Titre :

Modular Avionics Software Integration on Multi-Core COTS: Certification-Compliant Methodology and Timing Analysis Metrics for Legacy Software Reuse in Modern Aerospace Systems

---

"Libérée, délivréééééée..." – Elsa, *La Reine des Neiges*

"Let it go, let it gooooo" – Elsa, *Frozen*

Contents

List of Figures

List of Tables

”The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge” – Stephen Hawking

Acknowledgments

Creating a PhD contract is no piece of cake. It involves submitting an idea to the hierarchy, writing a subject and carefully choosing an academic environment that is relevant to the PhD. For me, it meant coming to a new city, and sacrificing personal paths that were laid out in front of me to shape my future. When starting a PhD involves such sacrifices, you better make your PhD worth it to compensate the personal pain by great work and professional perspectives.

This PhD has been the best, but also the worst thing that has happened to me so far. I even contracted serious back problems during that same period, which lightened once the PhD was over. I always thought the PhD would be the last guided, schooling environment before entering the workforce and all the challenges it brings. I was wrong. But I needed it. These three years of PhD have been a mix of technical research, social conventions learning, a bit of politics and diplomacy sensitization, along with psychological analysis of all the kinds of personalities that may exist in the world. I started as a weak student, lacking confidence, way too naive and gullible. I came out of it focused and more efficient, always true to myself and others.

I would like to thank Yamine Ait Ameur for agreeing to be the PhD director.

I would like to thank Wenceslas Godard for taking me as a 6-month intern, and then offering me the position for this PhD contract, which he initiated. Thank you for letting me write the PhD subject. Thank you for all the space and autonomy you have given me, and for always letting me decide what to do and how to react when my work was challenged. It allowed me to stay focused and find my way out of the biggest challenges encountered, would it be about technical difficulties, but also political, socially tough situations. You thus helped shape my own contributions but also my own character along the way.

I would like to thank Marc Pantel for saying yes to the proposition of Wenceslas of being the academic environment for this PhD, although Marc admitted later that the PhD was not in his field of study. Thank you for giving me enough autonomy to find information and contact relevant people myself. Thank you for your peculiar sense of humor too, including during meetings where you would make funny statements and propose a new challenge to take on every now and then. I eventually understood that you wanted me to not restrict myself to the sole role of student, but also take the lead, be my own safety net and scope-guard, and only rely on solid studies to prove my points to you during technical meetings.

Last but not least, I would like to thank both Wenceslas and Marc for offering me the best gift supervisors could ever offer to their PhD student: the surprise news that you have both been pushing for my PhD contributions to be reused in a multipartner industrial research project. There is no better proof of you being convinced by my work than such a great news.

Now, I would like to actually thank Stephan Stilkerich for his guidance and support which took different shapes over the years, for instance by providing advice on the manuscript breakdown, PhD title, etc. but also the industrial contacts that turned out to play an essential role in the test environment of this thesis. Without you, I would probably not have enough results to defend right now.

I would like to thank every contact I had at Absint, Freescale, and Wind River Inc. for helping me with tools usage and for being interested in my work. Thank you for your patience, especially Stefan Harwarth from Wind River, for always supporting me whenever I asked for an umpteenth free license extension to perform my tests on a real platform.

I would like to thank Isabelle Puaut and Pascal Richard for agreeing to review the manuscript and approve it for defense. I would like to thank Sebastian Altmeyer, Frederic Boniol, Claire Maiza and Christine Rochange as well for agreeing to be part of my jury and for the rewarding appreciation they delivered at the end of the defense. Special thank you to Stephan Stilkerich again, who could not make it for the defense and sacrificed himself off the jury so that I could defend in July rather than having to find a time slot matching every jury member in September, which can be very complex when it involves professors.

I would like to thank my friends from Paris for supporting me despite the distance. I would also like to thank all the friends I made during conferences and kept in touch with. I am grateful for your support, your advice but also all the fun we had after the lecture sessions of the conferences. I would also like to thank the PhD students with whom I briefly shared an office in the first year of my PhD, and in general other students and fellow researchers with which I shared some inspirational talks over a coffee.

I cannot thank enough everyone at Airbus Operations who accepted to join for questioning sessions from time to time, and provided me with all the information I needed to shape my contributions until it fit their industrial environment. I owe you the relevance of my PhD contributions.

In the same line of idea I will always remain grateful to Björn Lisper for inviting me to join the TACLe (Timing Analysis at Code Level) COST Action after I submitted my very first paper to a workshop he was chairing. You provided me with the environment I needed to get in touch with the biggest researchers and contributors of the real-time community, but also to attend technical meetings to remain up to date with the state of the art at no cost. This all was essential in the context of my PhD, and it meant a lot to me. Special thank you to my father for explaining to me what a COST Action was back then, and for encouraging me to join it. In general thank you to every member of the action with which I had the pleasure to interact with; for your advice, encouragement, but also kindness and the fun we had during conferences.

I would like to thank everyone at Airbus Group Innovations, where I spent almost all my time. Let's not forget the German side as well, for the team that welcomed me is dispatched between France and Germany. Thank you to any colleague that got interested in my work, and always took some time to discuss with me whenever they visited Toulouse for some other reason. I will not state any name by fear of forgetting anyone, but I believe they will recognize themselves. In general, thank you to every person at Airbus Group Innovations, Airbus Operations and Airbus Defense and Space who accepted to proofread my manuscript sometimes more than once, and who granted me their friendship; thank you for listening to me and supporting me whenever I took a hit I was not prepared for, and for always getting me back on my feet.

I would like to thank my several osteopaths for getting me back on my feet too, literally this time, more than once in three years.

Finally, I would like to thank my family for always supporting me through these hard times. I am still amazed that each and everyone of you made it to the defense, although it was not that simple at first. Including little Lilia, aged 7 months when attending my defense; you probably did not understand much, let alone remember it, but you were a delight to everyone and I very much appreciated your presence. Thank you for clearing your busy schedule of kindergarten workshops and toys reviewing committees, just to come to my defense.

To finish, a special thank you to my mother who took care of everything for the defense celebration, and who was even more stressed out than me at my defense. And a special thank you to my father who may even be happier than me over my doctors degree. Thank you for letting me handle everything, and refrain from asking questions even though you quickly sensed that something was seriously wrong with my PhD environment. I appreciated your patient supervising, and all the advice you gave me as a professor yourself.

”The best way out is always through” – Robert Frost

Acronyms

AMP:	Assymmetric Multiprocessing
ARINC:	Aeronautical Radio, Incorporated
ARP:	Aerospace Recommended Practice
ASIC:	Application-Specific Integrated Circuit
COTS:	Component Off-The-Shelf
CP:	Constraint Programming
CPIOM:	Core Processing and Input/Output Module
DAL:	Design Assurance Level
DDR:	Double Data Rate
DRAM:	Dynamic Random Access Memory
EASA:	European Aviation Safety Agency
EDF :	Earliest Deadline First
FAA:	Federal Aviation Administration
FAR:	Federal Aviation Requirements
GCD:	Greatest Common Divider
IMA:	Integrated Modular Avionics
I/O:	Inputs/Outputs
JAR:	Joint Aviation Requirements
JEDEC:	Joint Electron Device Engineering Council
LCM:	Least Common Multiple
LRU	Line Replaceable Unit
MAF:	MAjor time Frame
MIF:	MInor time Frame
NUMA:	Non Uniform Memory Access
PMC:	Performance Monitoring Counter
RM:	Rate Monotonic
RTA:	Response Time Analysis
SMP:	Symmetric Multiprocessing
SWaP:	Size, Weight and Power
TSP:	Time and Space Partitioning
UMA:	Uniform Memory Access
WCRT:	Worst-Case Response Time
WCET:	Worst-Case Execution Time

Symbols

A_p	function computing the maximum number of requests generated by core p depending on the considered time interval
a_{ij}	general term of the matrix defining the partition-to-core allocation in the one-to-one integration strategy
AS	set of suppliers responsible for the design of the applications integrated onto the same multicore platform
$c2mc_{ij}$	general term of the matrix specifying the core-to-memory controller – also referred to as core-to-memory path – allocation in the hardware platform
$cacheLat_p$	core p L1 cache access latency
$cacheSize_p$	size of the L1 cache of core p
C_i	vector gathering the execution durations in isolation of τ_i depending on the core on which it is executed
C_i^p	k^{th} execution duration in isolation of τ_i when running on core p
$coreAff_{ij}$	general term of the matrix defining the partition-to-core affinities for a given software and hardware platforms in the one-to-one integration strategy
$coreExcl_{ij}$	general term of the matrix defining the task-to-core exclusion constraints for a given software and hardware platforms in the one-to-one integration strategy
Clk_p	clock frequency of core p
Clk	core clock frequency whenever all cores are defined with the same value of clock frequency
C_{SW_p}	upper-bound of the context switch overhead of the core p
C_{SW}	context switch overhead whenever all cores have the same value of context switch overhead upper-bound
D_i	deadline of τ_i
d_{INT}	function computing, for a given task, its WCET and maximum number of memory access requests per execution, an upper-bound of the maximum total latency suffered by the task from the moment the corresponding memory access requests are issued by the corresponding core, to the moment they are serviced by the interconnect and transmitted to the corresponding memory controller
d_{RAM}	function computing, for a given task, its WCET and maximum number of memory access requests per execution, an upper-bound of the maximum total latency suffered by the task from the moment the corresponding memory access requests are transmitted to the corresponding memory controller, to the moment when the requests are serviced
$d_{RAM_{method_1}}$	definition of function d_{RAM} using a first computational method (see chapter 5)
$d_{RAM_{method_2}}$	definition of function d_{RAM} using a second computational method (see chapter 5)
$DRAMSIZE$	total size of the main memory

E_i^k	upper-bound of the maximum total CPU time budget required by the tasks of π_i for their respective executions in the k^{th} window of π_i
$gMAF$	MAjor time Frame in the one-to-all integration strategy
$gMIF$	MInor time Frame in the one-to-all integration strategy
H_i	vector gathering the maximum number of requests to the main memory τ_i can generate in one execution depending on the core on which it is executed
H_i^p	maximum number of requests to the main memory τ_i can generate in one execution when running on core p
$hp(\tau_i)$	set of tasks having an equal or higher priority than the priority of τ_i
ipc_{ij}	general term of the matrix specifying inter-partition communications between partitions
$isPreemptive_i$	boolean parameter defining whether tasks of π_i are all preemptive or non-preemptive
$isSharingMS_{pq}$	in the one-to-one integration strategy, general term of the matrix identifying couples of cores (p, q) of the multicore platform which contain tasks or partitions that share access to at least one memory controller
$isSharingMS_{pqj}$	in the one-to-all integration strategy, and for a given partition π_j, general term of the matrix identifying couples of cores (p, q) of the multicore platform which contain tasks of π_j that share access to at least one memory controller
JD_p^{inter}	inter-bank interference a task located on core p can suffer during its execution in the one-to-one integration strategy in the second computational method
JD_p^{intra}	intra-bank interference a task located on core p can suffer during its execution in the one-to-one integration strategy in the second computational method
JD_{pj}^{inter}	inter-bank interference a task of π_j located on core p can suffer during its execution in the one-to-all integration strategy
JD_{pj}^{intra}	intra-bank interference a task of π_j located on core p can suffer during its execution in the one-to-all integration strategy
J_i	jitter upon first activation of τ_i
L	row-conflict service time
$latest_i^k$	jitter upon activation of the k^{th} CPU time window of π_i
l_{bus}	latency corresponding to one main memory request traversing the interconnect
l_{max}	maximum memory request service latency, corresponding to the worst-case scenario (row conflict, etc)
MAF_p	MAjor time Frame on core p in the one-to-one integration strategy
$mcSize_i$	total size of the main memory area accessible from the memory controller i
MIF_p	MInor time Frame on core p in the one-to-one integration strategy
msg_{ij}	general term of the matrix specifying message-based communications between tasks
na_{ij}	general term of the matrix defining the task-to-core allocation in the one-to-all integration strategy

$nbActiv_i$	total number of executions of τ_i in one MAF
$nbCacheLines_p$	number of cache lines of the L1 cache of core p
$nbWindows_i$	total number of CPU time windows reserved for π_i in one MAF
N_C	number of cores available on the multicore platform and considered for active usage at runtime
$nFrames_i$	total number of frames in the partition cycle of π_i
$nFrames$	total number of frames in the MAF
N_{MC}	total number of memory controllers – also referred to as memory paths – available in the hardware platform
N_T	total number of tasks integrated onto the same multicore platform
$N_{T_{as_a}}$	total number of tasks developed by supplier as_a
N_P	total number of partitions integrated onto the same multicore platform
$N_{P_{as_a}}$	total number of partitions developed by supplier as_a
$overlapping_{ijkl}$	general term of the matrix identifying couples of task instances (τ_i^k, τ_j^l) scheduled at least partially in parallel on different cores of the multicore platform
$p2mc_{ij}$	general term of the matrix defining the partition-to-memory controller – also referred to as partition-to-memory path – in the one-to-one integration strategy
$\mathcal{P}^{as_a}$	set of partitions of all applications integrated onto the same multicore platform
$PART_{ij}$	general term of the matrix defining the task-to-partition allocation
π_i	i^{th} partition of the software platform, i e partition which identifier is i
P_i	period of repetition of π_i
pid_i	identifier of the partition to which τ_i belongs
pO_i^k	start date of the k^{th} time window of π_i
$pPrec_{ij}$	general term of the matrix specifying inter-partition precedence relations and/or partitions ordering preferences of system designers and integrators to be enforced in the partition-level schedule
$pRam_i^p$	size of memory space reserved for π_i in the main memory area accessible from the p^{th} memory controller
$prec_{ij}$	general term of the matrix specifying inter-task precedence relations
$prio_i$	priority of τ_i
R_i	worst-case response time (WCRT) of τ_i computed during the schedulability analysis
R_i^k	WCRT of τ_i^k
RD_p^{inter}	inter-bank interference a task located on core p can suffer during its execution in the one-to-one integration strategy in the first computational method
RD_p^{intra}	intra-bank interference a task located on core p can suffer during its execution in the one-to-one integration strategy in the first computational method
RD_{pj}^{inter}	inter-bank interference a task of π_j located on core p can suffer during its execution in the one-to-all integration strategy in the first computational method
RD_{pj}^{intra}	intra-bank interference a task of π_j located on core p can suffer during its execution in the one-to-one integration strategy in the first computational method

$reorder$	delay suffered by a memory request due to the reordering effect
$shared_{pq}$	general term of the matrix defining whether cores p and q share access to some main memory area in the one-to-one integration strategy
$shared_{pqj}$	general term of the matrix defining whether, for a given partition π_j, cores p and q schedule tasks of π_j that share access to some main memory area in the one-to-all integration strategy
$t2mc_{ij}$	general term of the matrix defining the task-to-memory controller – also referred to as task-to-memory path – in the one-to-all integration strategy
τ_i	i^{th} task of the considered task set, i e task which identifier is i
τ_i^k	k^{th} execution instance of τ_i
$\mathcal{T}$	set of tasks of all applications integrated onto the same multicore platform
$\mathcal{T}^{as_a}$	set of tasks of all applications designed by supplier as_a
T_i	period of repetition of τ_i
$taskCoreAff_{ij}$	general term of the matrix defining the task-to-core affinities for a given software and hardware platforms in the one-to-all integration strategy
$taskCoreExcl_{ij}$	general term of the matrix defining the task-to-core exclusion constraints for a given software and hardware platforms in the one-to-all integration strategy
tO_i^k	start date of the execution of τ_i^k
$tRam_i$	memory footprint of τ_i
u_i	utilization ratio of τ_i, also referred to as workload of τ_i
w_i	worst-case execution time (WCET) of τ_i
w_i^k	WCET of τ_i^k
$W(t)$	workload of a processing resource in a time interval $[0, t[$

Résumé

Les contentions apparaissant au niveau des ressources partagées par les coeurs d'un multicoeur sont problématiques pour les systèmes temps réel critiques. C'est en particulier le cas pour l'industrie aérospatiale, où il est impératif d'assurer un comportement temporel sain de tout système, et ce en avance de phase dans le cycle de développement. Etre capable de prédire le respect de toutes les échéances temporelles d'un système dans n'importe quelle situation pouvant être rencontrée en temps-réel dans l'environnement du système sous étude est indispensable pour obtenir les accréditations délivrées par les autorités de certification au niveau logiciel.

Le but de cette thèse est de proposer une approche pour le portage d'applications IMA (Avionique Modulaire Intégrée) préexistantes sur plateforme multicoeur, et ce sans modification majeure tant au niveau logiciel que matériel. L'objectif final de la thèse est de proposer une approche qui respecte les objectifs de certification appliqués au développement de calculateurs logiciels; cela implique aussi bien les contraintes de certification incrémentale, que le respect des concepts clés de l'IMA, à savoir le partitionnement spatial et temporel des applications intégrées sur le même module multicoeur.

Cette thèse intervient dans le cadre d'un contrat CIFRE (Contrat de d'Initiation à la Formation à la Recherche en Entreprise), à la demande d'Airbus Group Innovations Toulouse. Ainsi, une volonté additionnelle aux objectifs de la thèse et d'importance majeure est celle de suivre le plus possible les process intervenant dans les cycles de développement de logiciel IMA tels qu'ils existent aujourd'hui chez Airbus, pour des calculateurs basés single-core. Si l'on propose une méthodologie d'intégration multicoeur qui possède un maximum de similitudes avec le process d'intégration actuelle, cela augmente les chances des contributions de cette thèse d'être exploités en entreprise dans les plus brefs délais, l'effort à fournir pour proposer une nouvelle façon de travailler à des centaines de personnes expérimentées et habituées aux process existants depuis un certain nombre d'années étant généralement significatif. Il est notamment plus facile de convaincre des personnes de passer à une nouvelle méthodologie si l'effort d'adaptation est moindre, c'est-à-dire si les étapes impliquées dans la nouvelle méthodologie ne sont pas très différents de celles dans l'ancienne méthodologie. De plus, l'efficacité de la nouvelle approche est plus facile à prouver si l'approche ressemble aux process actuels, dont l'efficacité pour obtenir l'accord des autorités de certification n'étant en général plus à prouver.

Enfin, un objectif secondaire de la thèse est de chercher à optimiser au maximum les architectures intégrées résultant de l'étape d'intégration logicielle/matérielle. Si possible, une ou plusieurs des étapes de la méthodologie d'intégration multicoeur devraient être automatisées, de manière à accélérer les études de choix d'architecture tout en orientant la sélection finale vers des conceptions optimisant les critères de performance les plus pertinents pour l'industrie aérospatiale. L'automatisation permet également la réduction du temps et effort à fournir pour les tests et vérifications impliqués dans le cycle d'intégration, et ainsi de réduire le time-to-market du système avionique complet.

Cette thèse propose deux méthodologies complètes pour l'intégration IMA sur COTS (Component Off the Shelf, ou composant sur étagère) multicoeur. Toutes deux offrent des avantages différents et s'utilisent dans des situations complémentaires. Au final, il s'avèrera que les méthodologies proposées dépassent le cadre fixé originellement dans cette thèse: elle peuvent s'utiliser dans le cadre de développement de nouveau logiciel, qu'il soit IMA ou non, tant que

l'architecture visée est basée sur des multicoeurs.

L'une des deux méthodologies, appelée "one-to-all integration strategy", correspond à la situation où, à un instant donné, tous les coeurs sont utilisés au service d'une seule et même partition. Cette stratégie respecte tous les objectifs de certification, y compris le développement, la vérification et certification incrémentale, mais aussi le partitionnement robuste (spatio-temporel) des applications. La stratégie "one-to-one" reste pertinente pour tous les niveaux de criticité logicielle, y compris pour les applications DAL A (le plus haut niveau de criticité). Pour ces raisons, la stratégie "one-to-one" peut être vue comme la stratégie ayant le plus de chance d'être exploitée par l'industrie aérospatiale à l'issue de cette thèse.

La seconde stratégie, nommée "one-to-one integration strategy" correspond à la situation où chaque coeur du multicoeur a son propre schedule et ordonnance son lot de partitions indépendamment. Elle peut être utilisée pour des applications IMA jusqu'au niveau de criticité DAL C, pour une application multipartitions jusqu'au DAL A, ou encore pour tout logiciel temps-réel critique non IMA.

Les deux méthodologies d'intégration proposées sont qualifiées de "complètes" car elles contiennent:

- Une analyse temporelle statique qui borne les interférences inter-coeurs et permet de dériver des bornes supérieures de WCETs de manière fiable;

- Une formulation de problème de programmation par contraintes (PPC) pour l'allocation automatique et optimisée de logiciel sur matériel; la configuration résultante est correcte par construction car le problème de PPC exprimé exploite l'analyse temporelle mentionnée précédemment pour effectuer une vérification temporelle sur chaque configuration testée.

- Une formulation de problème de PPC pour la génération d'ordonnancement automatique et optimisé; la configuration résultante est correcte par construction car le processus exploite l'analyse temporelle mentionnée précédemment pour effectuer une vérification temporelle sur chaque configuration testée.

Abstract

Interference in multicores is undesirable for hard real-time systems, especially in the aerospace industry for which it is mandatory to ensure beforehand timing correctness and deadline enforcement in a system runtime behavior, to be granted acceptance by certification authorities.

The goal of this thesis is to propose an approach for multi-core integration of legacy Integrated Modular Avionics (IMA) software, without any hardware nor software modification, and which complies as much as possible to current, incremental certification and IMA key concepts such as robust time and space partitioning. The motivation of this thesis, supported by Airbus Group Innovations as an industrial PhD contract, is to stick as much as possible to the current IMA software integration process to maximize the chances of having avionics industries apply the contributions of this thesis to their future systems. Another reason is that the current process has long been proven efficient on aerospace systems currently in usage. A third motivation is to minimize the extra effort needed to provide certification authorities with timing-related verification information required when seeking approval. As a secondary goal, depending on the possibilities, the contributions should offer design optimization features, and help reduce the time-to-market by automating some steps of the design and verification process.

This thesis proposes two complete methodologies for IMA integration on multi-core Components Off The Shelf (COTS). Each of them offers different advantages and has different drawbacks, and therefore each of them may correspond to its own, complementary situations. One of the two proposed strategies fits all avionics and certification requirements of incremental verification and robust partitioning, and can therefore be used for applications with the highest criticality level, also referred to as Design Assurance Level (DAL) A. The other strategy proposed in this thesis offers maximum Size, Weight and Power (SWaP) optimization, and fits either up to DAL C applications, multipartition applications or even non-IMA applications.

The methodologies are said to be "complete" because this thesis provides all necessary techniques to go through all steps of the software integration process. More specifically, this includes for each strategy:

- Static timing analysis metrics for safely upper-bounding inter-core interference, and deriving safe WCET upper-bounds for each task. Both feasibility and schedulability analysis are considered in this thesis for multicore-based IMA systems.
- A Constraint Programming (CP) formulation of the software/hardware allocation problem for multicore-based IMA systems. Proposing a CP formulation enables the automation of the design space exploration and allocation configuration. The allocation selected at the end of the CP solving process is correct by construction since the CP problem embraces one of the proposed timing analysis mentioned earlier.
- A CP formulation of the schedule generation for multicore-based IMA systems. As for the allocation, proposing a CP formulation enables the automation of the design space exploration and schedule generation process. The schedule generated at the end of the CP solving process is correct by construction since the CP problem embraces the proposed timing analysis mentioned earlier.

Résumé Etendu

Motivations

Pour les systèmes temps-réel dur, il est tout aussi important d'obtenir des résultats exacts, que de les obtenir à temps. En particulier pour les systèmes avioniques, des résultats corrects obtenus en retard, soit après l'échéance qui a été défini pour le calcul de ces résultats, peuvent conduire à un dysfonctionnement majeur au niveau du système avion complet, avec des conséquences graves telles que la mort des passagers de l'avion et la perte de l'avion.

Pour que ce genre de situation ne se produise jamais, les systèmes avioniques font l'objet d'un cycle de conception et de vérification strictement réglementé, couplé à un processus de certification strict avant d'être jugés opérationnels puis commercialisés. Par exemple, en ce qui concerne les logiciels critiques embarqués dans un système avionique, chaque opération de logiciel doit être analysée au moment du design afin de dériver un majorant supérieur des temps d'exécution pire cas (aussi notés WCET, pour Worst Case Execution Time) de chaque traitement défini dans le logiciel, de manière à pouvoir vérifier la validité des échéances associées, configurer en avance de phase le plan d'ordonnancement, i.e. les enchaînements des traitements logiciels dans le temps, et vérifier que toutes les échéances seront toujours respectées à l'exécution pendant tout le temps de vol de l'avion. Une fois que le résultat d'une telle vérification montre le respect de toutes les échéances, le résultat et le processus de vérification lui-même peuvent être montrés aux autorités de certification comme justification d'un comportement temps-réel sain, prouvé en avance de phase.

La certification d'un avion est cruciale pour sa commercialisation. Au niveau logiciel, elle implique un respect d'un certain nombre d'objectifs définis par les autorités. Chercher à réspecter ces objectifs et le prouver représentent une part importante de l'effort de conception et vérification dans le cycle de vie d'un logiciel avionique, afin d'assurer sa conformité aux réglementations de certification. Cela s'applique non pas seulement au logiciel, mais à tout sous-système constituant l'avion. Chaque fois qu'une modification est effectuée sur une partie d'un système avionique, toutes les parties de ce système qui sont impactées par les modifications doivent être re-vérifiées, et leur conformité à la réglementation de certification à nouveau prouvée.

Une manière efficace d'essayer de réduire le temps et les efforts passés à gérer des modifications de conception consiste à adopter une approche de conception modulaire et indépendante. Par exemple, l'ensemble des logiciels intégrés dans un système avionique représentent plusieurs applications qui ne sont pas nécessairement développées par le même fournisseur. D'un autre côté, chaque application est développée indépendamment l'une de l'autre, peu importe le fournisseur, pour des raisons de sécurité. Cela permet également de s'assurer que chaque application peut être analysée indépendamment l'une de l'autre, de sorte que la modification d'une fonction n'ait aucun impact sur les autres fonctions. En tant que tel, seules les fonctions modifiées devront être ré-vérifiées; Le processus de développement, de vérification et de certification permettant ce genre d'indépendance est dit *incremental* [172].

En plus de la certification incrémentale, l'architecture IMA (Avionique Modulaire Intégrée) [1] favorise la modularité des systèmes avioniques. Dans les architectures IMA, une application logicielle est divisée en sous-éléments exécutés strictement indépendamment les uns des autres à l'exécution, ce qui assure une isolation temporelle et spatiale de ces sous-éléments entre eux. Une telle séparation est souvent appelée *partitionnement robuste*.

En résumé, les concepts de certification incrémentale et d'architecture IMA assurent la modularité d'un système avionique, et nécessitent une stratégie de vérification qui conserve l'esprit de séparation et isolation des applications et sous-élements des applications. Chaque fournisseur d'application décide comment découper une application en sous-éléments indépendants. Pour cette raison, les fournisseurs d'applications discutent avec les concepteurs de l'avion en cours de développement; la connaissance et l'expérience des fournisseurs d'applications et des concepteurs de systèmes avioniques sont essentielles pour certifier les systèmes avioniques.

Tout logiciel est exécuté sur une plateforme matérielle sur laquelle un processeur au moins est présent. En règle général pour des systèmes aussi critiques que les systèmes aérospatiaux, la tendance est d'exploiter des processeurs sortis sur le marché depuis déjà quelques années, dont les processeurs précédents de la meeêm famille ont en général un historique d'utilisation dans l'industrie aérospatiale conséquent, mais surtout qui sont jugés suffisamment fiables par des concepteurs de systèmes experts dans le domaine de l'embarqué temps-réel critique.

La motivation principale d'un tel choix est de pouvoir compter sur des plateformes électroniques qui ont eu le temps d'être testée de manière approfondie, et qui se sont avérées compatibles avec toutes les exigences des systèmes avioniques dans lesquels elles ont été exploitées. Une autre motivation est de faciliter la conception des futurs systèmes avioniques, car il est plus facile de réutiliser une plateforme matérielle dont les caractéristiques temps-réel et comportementales sont bien connues par expérience du fait de son utilisation dans un programme avion antérieur.

Jusqu'à présent, les plateformes matérielles embarquées dans les systèmes avioniques ne contiennent que des processeurs monocoeur. Depuis quelques années, les processeurs multicoeur – jusqu'à 8 ou 16 coeurs sur une même puce – et manycore – des centaines de coeurs sur une puce organisée en plusieurs tuiles et reliées par un réseau sur puce – ont fait leur apparition sur le marché de l'électronique. Chaque coeur supplémentaire sur une puce permet d'augmenter le nombre d'instructions logicielles traitées à un instant donné, améliorant ainsi les performances du système concerné. En l'occurrence, les derniers ordinateurs et téléphones mobiles proposés au grand public ont pu bénéficier d'un gain de performance significatif en embarquant de telles architectures. Le marché de l'électronique évoluant à une très grande vitesse, il suffit de six mois depuis la sortie d'un composant électronique sur le marché pour que celui-ci devienne obsolète. C'est le cas pour les processeurs; en particulier, maintenant que les fondeurs de processeur ont trouvé comment intégrer plus d'un coeur à une puce de processeur, les architectures monocoeur ne vont plus être poursuivies. Cela constitue un problème pour l'industrie avionique, dont tous les systèmes, les études de comportement temporel, l'expérience passée, repose sur l'utilisation de processeurs monocoeurs. Les systèmes aérospatiaux en général ne représentent pas une part de marché assez intéressante pour les fabricants de processeurs pour que ces derniers acceptent de poursuivre la production de processeurs monocoeurs exploités dans les systèmes avioniques. Par conséquent, l'industrie aérospatiale n'a guère d'autre choix que de s'adapter à l'évolution du marché électronique, et ainsi de passer à des architectures multi- ou manycoeurs.

Dans la mesure où le nombre d'applications logicielles, et plus généralement le nombre de lignes de code embarquées dans un avion augmente au fil des programmes, l'avènement des multi-manycoeurs peut s'avérer bénéfique pour l'industrie avionique. En effet, une unique plateforme multicoeur pourrait potentiellement remplacer autant de plateformes monocoeurs qu'il n'y a de coeurs dans le multicoeur exploité. Cela permettrait de réduire le poids et volume total de l'avion, sans compter le nombre de câbles à embarquer pour relier l'ensemble des plateformes embarquées, et le besoin en énergie nécessaire pour alimenter toutes ces plateformes matérielles tout le long de la mise en service de l'avion.

Cependant, passer au multicoeur dans les systèmes avioniques soulève un certain nombre de challenges [109]. En particulier, l'ensemble des procédures et standards intervenant dans les différentes phases de développement d'un calculateur embarqué doivent auparavant être mises à jour car elles sont adaptées aux architectures monocoeur et ne permettent pas de couvrir les cas multicoeurs, où plus d'un traitement logiciel est exécuté à chaque instant. En particulier, la vérification du bon comportement temporel et la preuve du respect des objectifs de certification associés sont plus complexes à mettre en oeuvre. L'utilisation d'un multicoeur dans un avion, avec plus d'un coeur exécutant des traitements applicatifs à chaque instant, ne sera pas possible tant que ces points ne seront pas réglés au préalable.

Un exemple concret de standard devant être mise à jour pour inclure le cas multicoeur est le standard définissant la manière dont un logiciel IMA doit s'interfacer avec toute plateforme matérielle sur laquelle il s'exécutera. En particulier, la mise à jour de ce standard devra prendre en considération l'indépendance des applications s'exécutant en simultané sur des coeurs différents, malgré le partage de ressources matérielles entre les coeurs d'un même processeur. Cela représente un challenge significatif. En effet, en raison des accès simultanés aux ressources partagées à l'intérieur de la puce multicœur qui se produisent au moment de l'exécution, l'isolement temporel est difficile à réaliser au moment de l'exécution sur une plateforme multicoeur.

De manière générale, toutes les activités d'analyse et de vérification habituellement menées dans un cycle de conception de logiciel temps-réel embarqué doivent être mises à jour avec des outils et des techniques capables d'analyser des plateformes multicœurs. Les interférences inter-core au moment de l'exécution doivent être majorées en toute sécurité lors de l'analyse des temps d'exécution pire cas des traitements logiciels, afin de pouvoir configurer en toute sécurité en avance le comportement d'exécution du système au moment du design. C'est seulement après de telles mises à jour que les concepteurs de systèmes seraient en mesure d'utiliser correctement les plates-formes multicœurs pour les systèmes aérospatiaux, mais aussi de pouvoir petit à petit reconstruire les connaissances et le savoir-faire nécessaires pour maîtriser les systèmes avioniques multicoeurs pour faciliter la réutilisation de plateformes similaires dans les programmes suivants.

L'apparition d'interférence inter-coeur à l'exécution pose problème à la vérification du comportement temporel d'un logiciel, principalement en raison du manque actuel d'outils ou de techniques permettant de produire un majorant des WCET de tâches exécutées dans un environement multicoeur [171]. Les demandes simultanées d'accès aux ressources générées par chaque coeur provoquent des délais d'attente d'accès aux ressources qui viennent s'ajouter au temps d'exécution du code associé à chaque tâche. Ces délais supplémentaires augmentent les temps d'exécution des tâches de manière plus ou moins significative suivant le multicoeur et le logiciel considéré. Le niveau de complexité de l'analyse à mener d'une part, et le couplage du partage des ressources dans les multicoeurs d'autre part, sont tels que l'analyse temporelle devient intractable, sinon trop pessimiste lorsqu'on essaie de réutiliser les techniques monocoeur actuelles en modélisant un environnement multicoeur. D'autant plus qu'il est encore impossible de modéliser une architecture multicoeur avec autant de précision qu'une architecture monocoeur même si on parvient à maîtriser la complexité des couplages impliqués par l'architecture. En effet, le manque d'information de conception par souci de protection d'IP par les fabricants de processeur, rendent la tâche encore plus délicate. Il ne suffit pas de trouver une technique d'analyse temporelle multicoeur; il faut aussi pouvoir contourner le manque d'information sur certaines ressources partagées à l'origine de délais d'interférence, comme les bus d'interconnexion entre les coeurs et les périphériques du processeur par exemple.

Aujourd'hui, il n'existe aucune solution pour l'analyse exhaustive de WCET sur multicoeur. Sans une telle solution, les contentions au niveau des ressources partagées ne peuvent pas être modélisées et majorées de manière absolue, de sorte qu'il n'y a aucun moyen de configurer à l'avance un plan d'ordonnancement dont on puisse apporter la preuve que toutes les échéances du système seront toujours respectées à l'exécution. Un tel manque de techniques d'analyse met en péril l'exploitation de platesformes multicoeur dans les systèmes aérospatiaux, ce qui en fait l'un des sujets de recherche les plus étudiés, dans le milieu académique mais aussi dans l'industrie aérospatiale et automobile. [90, 129, 119, 66, 64].

La littérature est pleine de diverses propositions pour aborder les problèmes posés par le multicoeur [64]:

- Certains travaux proposent de nouvelles couches middleware pour surveiller tous les accès aux ressources matérielles partagées, afin d'imposer des comportements d'exécution déterministes en éliminant les interférences inter-core à l'exécution [46];

- D'autres travaux proposent de nouvelles architectures multicoeur dédiées dans lesquelles chaque coeur possèderait ses propres ressources privées [7, 181, 149];

- Des travaux similaires se focalisent sur l'implémentation de composants hardware permettant de contrôler le partage de ressources, en espérant voir ces composants intégrés aux futures générations de COTS multicoeurs [111, 130, 170];

- Une toute autre stratégie également considérée dans la littérature est de proposer un nouveau modèle d'exécution suivant lequel les phases de calcul pur et les phases d'accès aux ressources partagées sont séparées de manière explicite pour réduire les congestions dans les multicoeurs [58];

- D'autres travaux proposent de nouvelles techniques d'analyse temporelle pour estimer ou majorer les congestions dues au partage de ressource, en se reposant ou non sur des modifications du logiciel et/ou du matériel constituant le système sous étude [19, 67, 85, 91, 111, 116, 118, 122, 127, 51].

La présente thèse a été initiée et financée par Airbus Group Innovations Toulouse, en tant que CIFRE (Contrat d'Initiation et Formation à la Recherche en Entreprise) en partenariat avec le laboratoire IRIT de Toulouse. L'objectif principal de la thèse est de proposer une approche pour la réutilisation d'applications IMA legacy (préexistantes) sur des processeurs multicœurs, sans modification matérielle ou logicielle.

La motivation principale de cette thèse est de maintenir autant que possible les processus industriels actuels, afin de maximiser les chances d'acceptation des contributions proposées par les industries aérospatiales et les autorités de certification en minimisant les changements et l'effort d'adaptation qui seraient nécessaires pour exploiter les contributions de thèse dans un cadre industriel.

Un objectif secondaire de cette thèse est d'automatiser et optimiser un maximum les étapes et sorties des phases conception logicielle, de manière à tirer profit du gain de performance que les mutlcioeurs peuvent potentiellement apporter.

Pour finir, la liste suivante regroupe les contraintes fortes imposées comme cadre de cette thèse, qui ont été idnetifiées lors des premiers mois de thèse comme lignes directrices des contributions, afin d'augmenter leurs chances d'être considérée comme une solution pour la prochaine génération de systèmes aérospatiaux:

- Les contributions proposées doivent être applicables à tout niveau de criticité, en particulier aux applications IMA de DAL A. En effet, comme nous l'avons déjà mentionné brièvement, toute tentative de solution proposée dans la littérature à ce jour n'est soit pas applicable aux systèmes IMA, soit nécessite une quantité importante de modifications et d'efforts pour être utilisée pour les systèmes IMA. De plus, aucune solution visant les systèmes IMA ne couvre l'ensemble des défis entravant l'utilisation de multicoeurs dans les systèmes avioniques.

- Les contributions proposées ne devraient pas être spécifiques à une plate-forme matérielle particulière ni s'appuyer sur uentechnologie ou le savoir-faire d'un fournisseur spécifique.

- En particulier, les contributions ne doivent pas dépendre de toute modification matérielle ou des caractéristiques d'architecture matérielle spécifiques. Il s'agit d'exploiter des COTS au lieu d'ASIC. Il s'agit également de ne pas s'appuyer uniquement sur une ligne de produits COTS spécifique ou sur un fabricant spécifique de COTS. Les contributions devraient reposer uniquement sur des mécanismes de configuration compatibles COTS, de sorte qu'aucun matériel supplémentaire spécial n'est nécessaire.
- Les contributions ne devraient pas impliquer une modification logicielle majeure, de sorte que les logiciels existants peuvent être réutilisés et aucune étude spéciale et / ou réglementaire supplémentaire n'est requise pour certifier la plate-forme logicielle considérée et l'approche globale proposée pour l'intégration multicouches peut être utilisée dans L'avenir le plus proche possible.
- Les contributions de thèse devraient réutiliser autant que possible les travaux existants de la littérature pour faire face aux défis multicœur, même si ces travaux ne représentent que des solutions partielles au défi multicore. Notre motivation est de promouvoir le transfert de la recherche à l'industrie; Cependant, il est également important que seules les techniques réalistes soient prises en considération, en adaptant les contraintes du monde réel aux systèmes et procédés industriels.
- Dans la même ligne d'idée, le réalisme des solutions proposées est impératif, même si cela implique de concevoir des systèmes sous optimisés. En particulier, il est important de ne pas faire une hypothèse qui n'est pas applicable aux multicores COTS actuels ou d'ignorer les situations actuelles inévitables au moment de l'exécution.
- Le résultat de la thèse devrait favoriser l'automatisation autant que possible dans les contributions proposées, afin de raccourcir le délai de mise sur le marché d'un système avionique basé sur multicoeur, mais aussi de réduire les efforts supplémentaires faits par les fournisseurs et les intégrateurs lors du traitement de nouvelles technologies telles que les multicores dans les futurs systèmes aérospatiaux.
- Enfin, les contributions de thèse devraient respecter autant que possible les processus de développement et de certification actuels. Même si ces derniers ne conviennent pas aux environnements multicoeur, respecter autant de règles de certification et de conception existantes que possible est susceptible d'aider à réduire le temps qui serait consacré à la négociation avec les autorités de certification afin d'étudier de nouveaux mécanismes et des choix de configuration.

Objectifs

La présente thèse s'attaque aux challenges rencontrés lorsqu'on envisage l'utilisation de plateformes multicoeurs dans un système avionique embarquant des applications IMA. Ces challenges ont été présentés et abordés dans un contexte d'utilisation industrielle, ce qui inclut des contraintes fortes comme la soumission aux contraintes de certification.

Après analyse de l'existant au sujet de l'utilisation de multicoeurs dans des systèmes temps-réel critiques, la cible de la thèse a été fixée autour de la minimisation des coûts de rework, coût, time-to-market, et adaptation des process, standards et étapes intervenant dans le cycle de développement d'un logiciel embarqéué temps-réel critique. De tels objectifs sont orientés industrialisation, et sont donc orthogonaux aux objectifs considérés dans la littérature. L'état de l'art des travaux autour des multicoeurs dans un environnement temps-réel critique montre des travaux impliquant soit:

- La proposition de puces multicoeur customisées, ce qui implique d'acheter la production en masse d'un design customisé si l'on considère leur usage dans l'avionique; cela représente un coût considérable par rapport à l'utilisation de COTS multicoeurs.

- La conception à partir de zéro des applications IMA existantes de manière à povoir les adapter à un environnement multicoeur en suivant les principes proposés dans certains travaux de la littérature. Cela représente également un coût important, en termes de temps et d'effort passés à tout refaire depuis le début, par opposition au portage de logiciel existant tel quel. Sans oublier le fait que tout re-design suivant de nouveaux principes implique de reprouver aux autorités de certification du bien fondé de ces principes et de leur sécurité.

- Des approches d'intégration qui ne conviennent pas aux architectures IMA. Les utiliser impliquerait de se débarrasser du concept de l'IMA dans les futures systèmes avioniques multicoeurs, ce qui va à l'encontre de la volonté d'isolation des applications, mais aussi de modluarisation et d'incrémentalité de manière générale.

Les objectifs de cette thèse sont les suivants:

- Minimiser autant que possible les dérivations de l'approche d'intégration proposée dans cette thèse aux processus actuels d'intégration IMA;

- Proposer des approches les plus indépendantes du matériel que possible. Toute dépendance ne devant pas être liée à une unique famille de processeurs, mais plutôt à une caractéristique présente dans la majorité des processeurs COTS proposés pour l'embarqué temps-réel critique. Ainsi, les contributions proposées pevent ne pas être applicables à absolument tout multicoeur COTS, mais doivent être applicables à tout COTS élligible à utilisation dans un contexte temps-réel critique d'après les études réalisées par des experts du domaine pour isoler les caractéristiques indispensables à l'avionique [123].

- Préserver le plus possible les concepts clés de l'IMA et respecter les exigences de certification majeures, comme par exemple le partitionnement robuste et la conception et vérification incrémentales.

Présentation Générale des Contributions de Thèse

Cette thèse propose deux méthodologies ou stratégies complètes pour l'intégration IMA sur le COTS multicore. Chacun a des avantages différents et des inconvénients différents, et peut donc être utilisé pour différentes situations et systèmes à concevoir. Une stratégie correspond à toutes les exigences en matière d'avionique et de certification, mais elle est susceptible d'entraîner des conceptions mal optimisées et l'autre génère des configurations aussi optimisées que possible, mais ne respecte pas certaines des exigences de certification clés actuelles, ce qui la rend pas adapté aux applications avec Les plus hauts niveaux de criticité.

Les stratégies d'intégration proposées sont "complètes" car nous fournissons toutes les mesures nécessaires pour passer à toutes les étapes des stratégies d'intégration proposées. Ils couvrent l'allocation de la plate-forme logicielle sur la plate-forme matérielle, la génération d'horaires, mais aussi les analyses temporelles. En résumé, les contributions de cette thèse consistent en:

- Modèles et techniques d'analyse de synchronisation tenant compte de l'interférence pour la vérification précoce du temps et l'application d'un comportement d'exécution déterministe du système: En particulier, nous proposons (i) un modèle mathématique de ressources partagées exploitées pour dériver une limite supérieure sécurisée sur les tâches au retard de l'interférence le plus défavorable en raison de chaque ressource partagée, et (ii) une approche pour effectuer une analyse du temps de réponse sécurisée pour IMA architectures dans des environnements multicouches. L'analyse résultante a été dérivée en deux approches différentes: l'une dédiée à la vérification de la validité d'une allocation logiciel / matériel en effectuant une analyse du temps de réponse et une dédiée à la dérivation des bornes supérieures sécurisées de WCET pour les tâches et les fenêtres temporelles CPU pour les partitions lorsqu'une allocation Et un calendrier a été défini. Cette contribution est présentée dans les détails du chapitre 5 et a été publiée dans [116].

- Une formulation de contrainte de programmation (CP) pour effectuer une allocation logicielle / matérielle automatisée, optimisée et sûre des applications IMA sur des plates-formes multidéveloppes. L'approche est sûre dans le sens où la faisabilité de la solution sélectionnée par le programme contraint est assurée puisque l'analyse de synchronisation implémentée dans cette thèse est intégrée comme contrainte du CP. Cette contribution est abordée dans le chapitre 6 et a été publiée dans [118].

- Une formulation de CP pour effectuer une génération de calendrier automatisée, optimisée et sécurisée pour les applications IMA sur les plates-formes multicore. On dit qu'il est sûr parce que le programme contraint incorpore l'analyse de planification proposée dans cette thèse comme une contrainte, afin d'assurer la validité de la solution sélectionnée par le programme contraint. Cette contribution est couverte par chapitre 6 et a été publiée dans [116, 118]. Cette contribution a été mentionnée dans nos pulations [116, 118, 117].

- Deux processus pour l'intégration sécurisée de l'IMA sur les plates-formes multicore: comme mentionné précédemment, l'un des deux aspects de toutes les caractéristiques de certification les plus importantes des systèmes IMA actuels, et l'autre offre une optimisation de conception maximale pour les applications IMA moins critiques. Chaque stratégie couvre l'allocation de la plate-forme logicielle IMA sur la plate-forme multicore et la génération d'un planning pour chaque noyau du processeur multicœurs. Pour le faire de manière sûre et automatique afin de gagner du temps et des efforts, les trois contributions mentionnées précédemment de cette thèse sont exploitées. Cette contribution est largement expliquée dans le chapitre 6, et a été publiée dans [117].

Le travail présenté dans cette thèse a été évalué sur une plate-forme réelle. L'évaluation a été effectuée sur le processeur Freescale / NXP QorIQ P4080 [9]. Une étude de cas de logiciel a été construite par l'auteur de cette thèse en exploitant le code source ouvert à partir de TacleBench benchmark suite [13]. Nous avons appliqué les stratégies proposées pour l'attribution et la planification de l'étude de cas IMA construite sur le processeur P4080 [9] à l'aide de Wind River IMA RTOS, VxWorks653 3.1 Multicore Edition [14]. L'évaluation consiste à mettre en œuvre sur la cible P4080 la configuration sélectionnée lors de l'application de l'une de nos deux stratégies et à vérifier qu'aucun délai n'a été manqué lors de l'observation du comportement d'exécution du système.

Les paragraphes suivants décrivent en plus de détails chaque contribution proposée.

Analyse Temporelle Multicoeur Statique et Majortion des Interférences Pire Cas

Cette thèse propose une condition suffisante de faisabilité d'allocation, qui inclut la prise en compte d'interférences multicoeurs, et qui est compatible à la fois avec les architectures logiciells IMA et les processeurs multicœurs COTS, homogènes comme hétérogènes. Pour ce faire, nous proposons une modélisation mathématique de la mémoire principale et du bus d'interconnexion reliant les coeurs à la mémoire, afin d'effectuer une analyse statique des pires délais d'interférence que chaque tâche puisse subir à l'exécution dû au partage de ces ressources.

L'analyse proposée est basée sur une extension de l'analyse du temps de réponse classique, de manière à s'adapter aux architectures IMA et aux environnements multicœurs. Avec la modélisation statique des interférences pire cas, l'analyse qui en résulte produit une borne supérieure du WCET et du WCRT de chaque tâche. Ces bornes sont essentielles pour prouver formellement la faisabilité d'une configuration d'allocation, mais aussi pour configurer un plan d'ordonnancement statique niveau partition. L'exploitation d'une telle analyse pour construire une allocation et un plan d'ordonnancement permet de faire vérifier au péalable tout choix de configuration, pour ainsi garantir un comportement dans lequel toutes les échéances temporelles sont respectées au moment de l'exécution. Des explications détaillées sur les analyses temporelles proposées sont données dans le chapitre 5.

Problème d'Allocation

Au cours du processus d'allocation, l'intégrateur module décide, pour un module donné constitué d'un processeur multicoeur donné, sur quel coeur chaque partition ou tâche sera exécutée. Une allocation se réfère alors à un mapping statique de chaque partition ou tâche des applications IMA aux coeur d'un processeur multicœur.

Dans cette thèse, nous utilisons l'expression *problème d'allocation* pour désigner les préoccupations concernant la manière dont la plate-forme logicielle doit être allouée spatialement aux coeurs d'un processeur multicoeur, et en mémoire principale. Cela comprend les questions suivantes: (i) sur quel coeur une partition ou tâche donnée sera exécutée, (ii) dans quelle zone son contexte mémoire sera stocké en DRAM et (iii) si plus d'un contrôleur mémoire est présent sur la plate-forme matérielle, quel contrôleur mémoire sera exploité par la tâche ou partition au moment de l'exécution. Aucune mention de la gestion du temps ni du découpage du temps n'est mentionnée dans de tels problèmes.

Dans le cadre d'une deuxième contribution, nous proposons une formulation du problème d'allocation d'applications IMA à une plateforme multicoeur. La formulation proposée correspond à une formulation mathématique d'équations, chaque équation représentant une contrainte du système. Pour résoudre un tel problème, nous exploitons la programmation par contrainte, bien que d'autres techniques puisse être exploitées à la place, comme des heuristiques de résolution par exemple.

Dans la formulation du problème d'allocation proposée dans cette thèse, nous exploitons l'analyse temporelle de faisabilité décrite dans le paragraphe précédent afin de proposer une approche pour la vérification d'une allocation en termes de respect d'échéances temporelles. L'analyse proposée consiste en une condition suffisante de faisabilité et est définie dans le problème d'allocation décrit dans cette thèse comme une contrainte à faire respecter par les variables du problème d'allocation.

Comme mentionné précédemment, la pertinence de l'allocation sélectionnée est garantie par l'analyse de faisabilité intégrée au problème d'allocation. Définie comme une contrainte du problème, elle permet de systématiquement vérifier l'existence d'un ordonnancement dans lequel toutes les échéances du lgiciel sont respectées, et ce pour chaque allocation en cours d'évaluation par le problème de programmation par contraintes. Finalement, l'utilisation de la programmation des contraintes pour définir la configuration d'allocation permet de gagner du temps mais aussi de réduire l'effort à fournir pendant le cycle de conception d'un système donné, par rapport à une technique manuelle de recherche d'allocation et d'analyse de faisabilité. En effet, le problème d'allocation est NP-complet, et les équations de calcul de majorant sûr de WCET sont suffisamment complexes pour provoquer facilement des erreurs d'inadvertance lorsqu'elles sont calculées manuellement.

De plus, de manière générale, exploiter la programmation par contraintes fait gagner du temps dans le sens où cela évite la détection tardive d'une allocation qui se trouve être non valide car ne respectant pas l'une au moins des échéances du système. L'analyse étant faite en simultané avec la recherche d'allocation dans cette thèse, toute allocation non valide est automatiquement rejetée des potentielles solutions d'allocation. Sans une telle détection au moment de la configuration de l'allocation, la durée du cycle de conception du système aurait été prolongée. En effet, la non-validité de l'allocation sélectionnée aurait été détectée plus tard seulement, voire ne pas être détectée du tout, ce qui aurait pu avoir des conséquences catastrophiques au moment de l'exécution, par exemple via la violation d'une échéance associée à une opération critique.

Problème d'Ordonnancement

Dans cette thèse, nous utilisons l'expression *problème d'ordonnancement* pour désigner les préoccupations concernant la manière dont les éléments de la plateforme logicielle (tâches et/ou partitions) sont temporellement allouées sur les coeurs d'un multicoeur, c'est-à-dire comment répartir des tranches de temps CPU aux tâches et/ou partitions de la plateforme logicielle.

Dans les systèmes IMA, et tout logiciel critique en général dans les systèmes avioniques, les plan d'ordonnancement sont statiques et configurés à l'avance lors du cycle de développement du système. Cela permet de s'assurer que le comportement du système à l'exécution a été vérifié et approuvé à l'avance au moment du design.

La définition d'un plan d'ordonnancement passe par: (i) la définition de dates d'activation pour chaque tâche et/ou partition, et (ii) la réservation d'une durée à partir de cette date pour la tâche ou partition concernée. Pour une partition IMA, l'ensemble de ces deux éléments est aussi appelé "fenêtre temporelle".

La sécurité des plans d'ordonnancement ainsi générés est garantie via l'exploitation de l'analyse d'ordonnancement proposée dans cette thèse. Cette analyse vérifie que toutes les contraintes temporelles définies pour le logiciel correspondant seront toujours respectées au moment de l'exécution. En particulier dans le problème d'ordonnancement tel que proposé dans cette thèse, une formulation de problème de programmation par contraintes recherche des dates d'activation pour chaque tâche et/ou partition, et l'analyse d'ordonnancement calcule les bornes supérieures des WCETs des instances de tâches dans une hyperpériode, ou MAF (MAjor time Frame). L'analyse détermine ensuite si toutes les échéances sont tenues pour les datdes d'activation et les WCETs calculés; si c'est le cas, le plan d'ordonnancement est dit valide. Le cas échéant, il est rejeté et le solveur de contraintes génère de nouvelles dates d'activation à tester, jusqu'à ce qu'un plan d'ordonnancement valide soit sélectionné, ou que le solveur annonce ne pas avoir trouvé de plan valide dans tout l'espace de recherche du problème défini.

En plus de garantir le respect des échéances dans le plan d'ordonnancement finalement sélectionné, l'analyse d'ordonnancement proposée dans cette thèse et intégrée dans la formulation du problème d'ordonnancement guide l'exploration de l'espace de recherche du problème. De manière analogue au problème d'allocation, l'utilisation de la programmation des contraintes pour générer des ordonnancements permet d'économiser du temps, mais aussi l'effort de l'intégrateur et/ou des fournisseurs d'applications, et aide à prévenir les détections tardives de configurations non valides.

Méthodologies pour l'Intégration Matérielle/Logicielle

L'ensemble des contributions décrites dans les précédents paragraphes ont été exploitées ensemble dans une contribution finale: la proposition de méthodologies pour l'intégration de logiciel IMA sur processeur multicoeur COTS.

Nous proposons deux stratégies. Notre première stratégie d'intégration est conforme à toutes les exigences industrielles: nous l'appelons la stratégie d'intégration *one-to-all.* Elle est basée sur une dérivation statique de l'approche de traitement SMP (Symmetric Multithreaded Processing) dans laquelle, au moment de l'exécution, une seule application est exécutée sur l'ensemble des coeurs d'une plateforme multicoeur par un OS unique. Avoir une unique application ordonnancée à chaque instant sur tout coeur actif d'un multicoeur préserve la notion de partitionnement robuste, en créant une situation dans laquelle le seul type d'interférence multicoeur existante est intra-application, par opposition à inter-application, ce qui est prohibé dans les systèmes IMA.

La stratégie d'intégration "one-to-all" peut être considérée comme une méthodologie pour la réutilisation d'applications IMA existantes sur COTS multicoeur, en toute sécurité et tout en respectant les exigences clés en matière de standard IMA et d'objectifs de certification des systèmes actuels (partitionnement robuste, certification incrémentale, bornage statique et sûr des WCETs, ...).

Les applications non critiques (DAL E à C) sont soumises à des exigences de certification moins strictes que les applications critiques. D'autre part, les processeurs multicœurs sont encore à l'étude par les autorités de certification cherchant à réglementer leur utilisation dans les futurs documents de certification pour l'avionique. Au vu de la difficulté d'analyse posée par les multicoeurs, les exigences de vérification temporelle peuvent éventuellement être assouplies pour les applications non critiques, au moins dans les premières tentatives de règlement. Sans compter que certains documents de certification différencient les objectifs suivant si un module avionique requiert un partitionnement robuste de ses applications ou non. Un exemple simple de module ne nécessitant pas ce genre de partitionnement est un module intégrant plusieurs partitions provenant toutes de la même application; toute interférence existante entre coeur reste alors intra-application.

Ces situations n'ont pas besoin d'une méthodologie d'intégration aussi stricte que la stratégie ”one-to-all”. A ce titre, nous proposons une seconde méthodologie d'intégration, nommée stratégie ”one-to-one”. Cette stratégie s'appuie sur une allocation statique de multi-traitement assymétrique (AMP) des partitions IMA aux coeurs d'une plateforme multicœur. Elle peut être appliquée dans le contexte d'applications IMA jusqu'à DAL C, ou pour des applications IMA multi-partition tant que toutes les partitions allouées à la plateforme multicoeur proviennent de la même application. Elle peut également être utilisée dans le cadre de l'intégration de logiciel critique non IMA à une plateforme multicoeur fédérée. La stratégie d'intégration ”one-to-one” permet d'obtenir une réduction maximale du nombre de calculateurs embarqués dans un système avionique possible grâce à nos techniques d'analyse temporelles et à la sélection de configurations suivant des objectifs d'optimisation. Cependant, le prix à payer pour ces systèmes optimisés est l'absence d'un partitionement temporel entre applications, et donc de partitionnement robuste, dans le système résultant.

Chacune des deux stratégies proposées est divisée en plusieurs activités, afin d'effectuer l'allocation, la génération d'ordonnancement, ainsi que toutes les vérifications correspondantes. Ces activités sont définies sous la forme de problèmes de programmation par contraintes afin de permettre leur automatisation, et donc de réduire le temps et l'effort passé à effectuer ces tâches. La programmation par contraintes permet également d'introduire des objectifs d'optimisation de manière à sélectionner, parmi les solutions du problème exprimé, celle optimisant le plus possible les critères de performance définis au préalable.

La description détaillée des activités menées dans chacune des stratégies, ainsi que la formulation des problèmes de programmation par contraintes correspondants, sont présentés dans le chapitre 6.

En résumé, en raison de la combinaison de toutes nos contributions dae cette thèse, les méthodologies d'intégration IMA proposées pour l'intégration multicoeur sont:

- Complètes, étant donné que les analyses temporelles proposées et les modèles exploités peuvent être considérés comme une justification fiable pour les autorités de certification du fait que les majorants supérieurs des WCETs calculés incluent les interférences intercoeur pire cas;

- Fiables, puisque toutes les techniques d'analyse mises en œuvre sont basées sur des techniques d'analyse statique produisant une enveloppe supérieure de tous les délais potentiellement soufferts par toute tâche à l'exécution dans le pire scénario;

- Simples d'application, puisque toutes les activités d'intégration sont automatisées en utilisant la formulation adéquate de programmation par contrainte proposée dans cette thèse;

- Flexibles, puisque tout modèle, analyse ou activité de chaque stratégie peut facilement être modifiée, de manière à s'adapter à une modification de l'environnement ou pour mise à jour suite à évolution de l'état de l'art des techniques d'analyse;

- Optimisées, puisque chaque étape comprend des fonctionnalités d'optimisation via l'expression de contraintes supplémentaires et de fonctions objectifs choisies en fonction des objectifs de performance industriels, afin de guider l'exploration exhaustive de l'espace de conception vers les solutions les plus optimisées suivant la propre définition de l'industrie aéronautique.

En particulier, la méthodologie d'intégration "one-to-all" est conforme aux principales exigences de certification courantes, ce qui en fait une véritable solution potentielle pour démarrer l'intégration IMA basée sur multicœurs dès aujourd'hui. Enfin, malgré leurs différences, les deux stratégies proposées peuvent être utilisées pour la réutilisation de logiciels existants sur COTS multicoeur, que ce soit pour des architectures IMA ou fédérées.

Stratégies d'Intégration SW/HW Proposées

Présentation Générale

Le principal challenge lorsque l'on souhaite intégrer des applications IMA à une plateforme matérielle donnée, est la détermination des besoins en temps CPU de chaque partition. Pour ce faire, au niveau partition, il y a besoin dans un premier temps de descendre au niveau tâches pour dériver les besoins en temps CPU de chacune des tâches d'une partition donnée. Une fois les besoins de chaque tâche connus, il est possible d'en déduire les besoins de la partition correspondante.

Comme mentionné précédemment, nous proposons deux stratégies pour l'intégration d'applications IMA sur des plateformes multicoeur:

- **La stratégie d'intégration "one-to-all" (cf. figure 1):** dans cette stratégie, une seule partition est exécutée à un instant donné sur le multicoeur considéré, en utilisant tout ou partie des coeurs disponibles. Pour ce faire, chaque fournisseur d'application se charge d'allouer les tâches à l'intérieur des partitions aux différents coeurs pour décider qui s'exécutera sur quel coeur. La migration n'est pas autorisée. Les tâches qui appartiennent à la même partition mais sont allouées à des coeurs différents seront ordonnancées de manière simultanée.

  De plus, dans la stratégie d'intégration "one-to-all", chaque fournisseur travaille indépendamment des autres fournisseurs d'applications, sans savoir quoi que ce soit des autres applications qui seront intégrées sur le même multicoeur. Cela correspond à la manière dont l'intégration a lieu dans le processus actuel d'intégration IMA.

  Ainsi, la stratégie "one-to-all" est en ligne à la fois avec le concept de partitionnement robuste, et d'incrémentalité des étapes de développement et vérification. Ces deux concepts sont très importants pour les systèmes IMA, en particulier du point de vue des objectifs de certification logicielle.

- **La stratégie d'intégration " one-to-one" (cf. figure 2):** dans cette stratégie, chaque coeur a son propre plan d'ordonnancement comme s'il s'agissait d'un processeur à part entière. Une partition ne peut être allouée qu'à un seul coeur; toutes les tâches d'une partition sont considérées allouées au même coeur que leur partition. Ainsi, le niveau partition peut être abstrait au moment du calcul des WCETs et le plan et l'analyse d'ordonnancement sont effectués au niveau tâche. Le plan d'ordonnancement niveau partition est ensuite déduit naturellement du plan d'ordonnancement niveau tâches via l'identification de la partition à laquelle chaque tâche appartient par définition.

  Dans la stratégie d'intégration "one-to-one", toutes les étapes – depuis l'allocation jusque la vérification d'ordonnancement – peuvent être effectuées par la même personne, qui possède toutes les informations de toutes les applications allouées au même module multicoeur. Cela pourrait par exemple correspondre à la nouvelle signification du rôle d'intégrateur module dans un contexte multicoeur.

Nous décrivons ici un peu plus en surface chacune des deux stratégies d'intégration proposées dans cette thèse. Plus de détails sont donnés dans le chapitre 6.

Stratégie d'Intégration ”One-to-All”

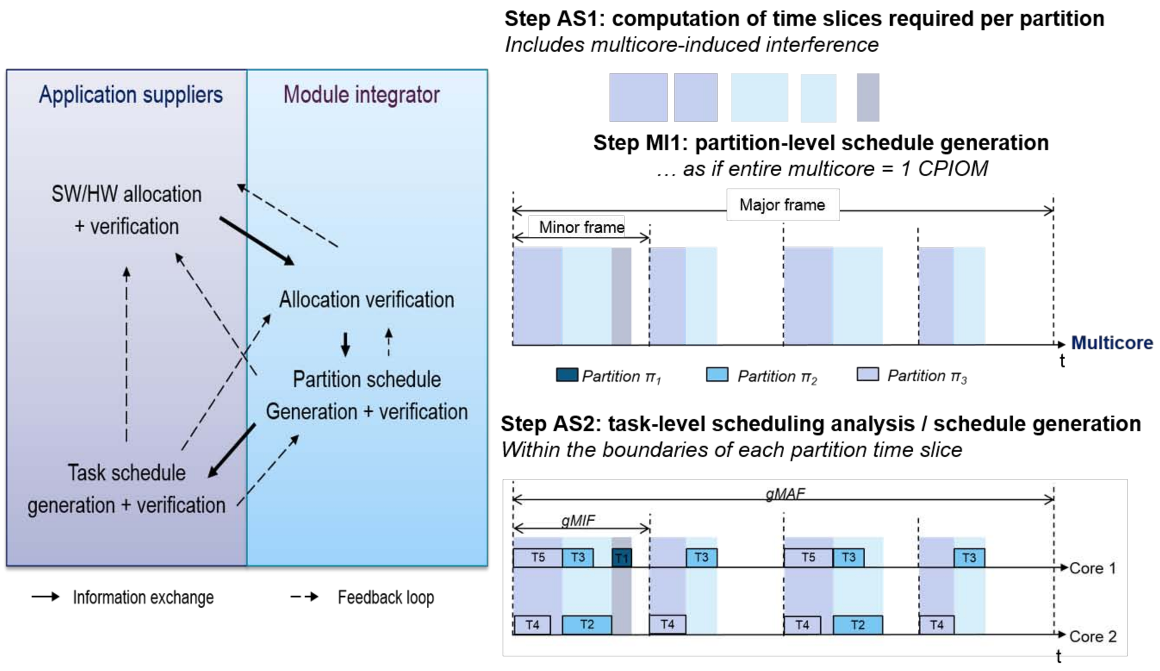

Step AS1: computation of time slices required per partition
Includes multicore-induced interference
Application suppliers
Module integrator
SW/HW allocation + verification
Allocation verification
Partition schedule Generation + verification
Task schedule generation + verification
Information exchange
Feedback loop
Step MI1: partition-level schedule generation
… as if entire multicore = 1 CPIOM
Major frame
Minor frame
Multicore
t
Partition π_1
Partition π_2
Partition π_3
Step AS2: task-level scheduling analysis / schedule generation
Within the boundaries of each partition time slice
gMAF
gMIF
T5
T3
T1
T4
T2
Core 1
Core 2

Figure 1: Stratégie d'Intégration ”One-to-All”

La stratégie d'intégration ”one-to-all” consiste à allouer de façon statique le contenu de chaque partition sur les coeur d'une plateforme multicoeur afin que chaque partition monopolise tous les coeurs du multicore pendant les intervalles de temps où la partition est exécutée. La figure 1 donne un exemple de plan d'ordonnancement résultant de la stratégie d'intégration ”one-to-all”. Le plan d'allocation proposé garantit que toute interférence inter-core correspond à une interférence intra-partition, la situation résultant devenant équivalente à des conceptions monocoeur. L'absence d'interférence inter-applications rend la stratégie d'intégration ”one-to-all” conforme avec l'exigence de partitionnement robuste des systèmes IMA.

Comme le montre la figure 1, la stratégie ”one-to-all” est divisée en trois étapes. Ces étapes ont été définies en fonction des activités à mener, et en fonction des délimitations des rôles des personnes actuellement impliquées dans un processus d'intégration IMA. Nous décrivons rapidement les trois étapes dans les prochains paragraphes. Une reprsentation graphique plus complète est éventuellement donnée via la figure 6.1 page 150. Le détail de la stratégie d'intégration ”one-to-all” est donné dans le chapitre 6.

Allocation Logicielle/Matérielle La première étape de la stratégie d'intégration ”one-to-all” consiste à allouer les tâches e chaque partition IMA aux coeurs d'une plateforme multicoeur, puis à dériver les besoins en temps CPU de chaque partition en fonction de l'allocation ainsi sélectionnée. Les entrées de cette étape sont les modèles du logiciel et de la plateforme multicoeur utilisée. La sortie de cette étape est une affectation de chaque tâche à un coeur et un expace mémoire, ainsi qu'unmajorant supérieur du WCRT de chaque tâche basé sur l'analyse de faisabilité proposée dans cette thèse et décrite précédemment.

Comme mentionné précédemment, l'allocation logicielle/matérielle est effectuée en allouant des tâches aux coeurs pour chaque partition; les tâches seront ordonnancées de façon à ce qu'à chaque instant, les seules tâches exécutées sur tout le multicoeur appartiennent toutes à la même partition. Cette étape est effectuée par chaque fournisseur d'application, sur leurs propres partitions et indépendamment des autres applications.

Dans cette étape d'allocation, chaque fournisseur d'application est également responsable de la vérification de la faisabilité de l'allocation sélectionnée. Cela revient à, pour l'allocation sélectionnée, vérifier s'il sera toujours possible de trouver au moins un plan d'ordonnancement dans lequel toutes les échéances temporelles de l'application concernée soient respectées. Le cas échéant, l'allocation est considérée comme non valide et doit être modifiée. Les contraintes temporelles à respecter pour chaque application peuvent être expriémes au niveau tâche comme ua niveau partition. Et chaque fournisseur ne connaissant que ses propres partitions, ils ne vérifient à cette étape que les contraintes au niveau tâche.

La vérification inclut le calcul de WCRT pour chaque tâche, via l'analyse de faisabilité proposée dans cette thèse. Les WCRT sont calculés avec prise en compte d'interférences multicoeur dues au partage de ressources entre les tâches exécutées en simultané sur des coeurs différents. Ces WCRTs sont également exploités pour déduire le temps CPU total que chaque partition requiert pour l'exécution de ses tâches.

Enfin, les entrées de cette étape sont les modèles du logiciel et de la plateforme matérielle. Les sorties sont les besoins en temps CPU de chaque partition.

Vérification de l'Allocation au niveau Module et Génération d'un plan d'Ordonnancement au niveau Partition La seconde étape de la stratégie d'intégration "one-to-all" consiste à vérifier que l'allocation de toutes les partitions dans leur ensemble pourront être attribuées assez de temps CPU comme demandé par chaque fournisseur d'application respectivement, tout en respectand les contraintes temporelles des partitions.

Si la vérification échoue, c'est-à-dire s'il n'est pas possible à l'intégrateur module d'allouer assez de temps CPU pour chaque partition sans violer l'une au moins des contraintes de périodicité des partitions, alors l'intégrateur module doit négocier avec tout ou partie des fournisseurs d'applications pour modifier les besoins en CPU des partitions. Plusieurs itérations peuvent être nécessaires, avant que la vérification d'allocation ne finisse par prouver qu'une allocation de partitions donnée respecte toutes les contraintes temporelles des applications.

La suite consite en la génération d'un plan d'ordonnancement des partitions. L'intégrateur module doit définir des fenêtres temporelles pour chaque partition selon leur périodicité respective. S'il ne parvient pas à trouver un ordonnancement valide dans lequel toutes les échéances au niveau partition sont respectées, l'intégrateur négocie des modifications avec un ou plusieurs fournisseurs d'applications. Plusieurs itérations peuvent être nécessaires jusqu'à ce que l'intégrateur module parvienne à générer un ordonnancement valide au niveau partition.

Génération et/ou Vérification d'Ordonnancement niveau Tâche La dernièere étape de la stratégie d'intégration "one-to-all" consiste soit:

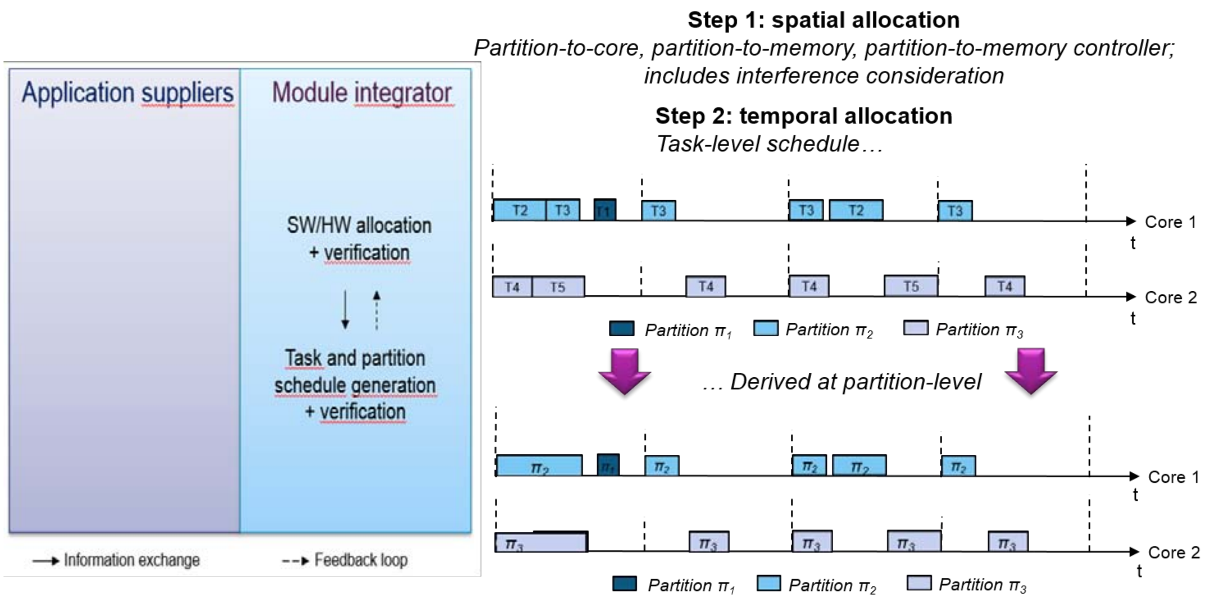
- Pour les partitions dont les tâches sont non-préemptives: générer un plan d'ordonnancement statique dans lequel toutes les échéances temporelles de l'application sont respectées; cela inclut le respect des fenêtres temporelles de la partition, c'est-à-dire vérifier que les exécutions de toutes les instances de tâches de la partition ont toujours lieu dans les limites des fenêtres temporelles de la partition.

- Pour les partitions dont les tâches sont préemptives: à vérifier qu'il existera toujours, pour les fenêtres temporelles définies au niveau partition, un ordonnancement valide dans lequel toutes les échéances de l'application seront respectées.

Les entrées de cette étapes sont le plan d'ordonnancement niveau partition. Il est important de rappeler que les fournisseurs d'application travaillent indépendamment les uns des autres. Ainsi, chaque fournisseur d'application ne connait que les fenêtres temporelles de ses propres partitions dans de plan d'ordonnancement niveau partition. L'intégrateur module est le seul à connaître le plan d'ordonnancement des partitions complet.

Chaque fournisseur d'application cherche à vérifier l'existence d'un plan d'ordonnancement valide niveau tâche. Le cas échéant, soit le fournisseur opère des modifications à son niveau pour trouver un arrangement valide, soit il négocie avec l'intégrateur module, pour obtenir plus de temps CPU pour sa partition par exemple. Plusieurs itérations peuvent être nécessaires avant que tous les fournisseurs ne parviennent à prouver l'existence d'un ordonnancement valide pour toutes leurs partitions respectivement.

Stratégie d'Intégration "One-to-One"



Step 1: spatial allocation
Partition-to-core, partition-to-memory, partition-to-memory controller; includes interference consideration
Step 2: temporal allocation
Task-level schedule…
Application suppliers
Module integrator
SW/HW allocation + verification
Task and partition schedule generation + verification
Information exchange
Feedback loop
T1
T2
T3
T4
T5
Core 1
Core 2
t
Partition π_1
Partition π_2
Partition π_3
… Derived at partition-level

Figure 2: Stratégie d'Intégration "One-to-One"

Comme illustré dans la figure 2, la stratégie "one-to-one" consiste en un processus linéaire pouvant être divisé en deux étapes: la génération d'une allocation logicielle/matérielle, et la génération d'un plan d'ordonnancement.

Allocation Logicielle/Matérielle La première étape d'intégration consiste à allouer la plateforme logicielle sur la plateforme matérielle, c'est-à-dire décider quelles partitions s'exécuteront sur quel coeur à l'exécution. Les tâches d'une partition donnée seront alors exécutée sur le coeur sur lequel leur partition a été allouée. Les inconnues fixées à la fin de cette étape représentent donc l'identification du coeur sur lequel chaque partition et/ou tâcheest allouée, mais aussi les WCETs et WCRTs des tâches. En effet, afin de s'assurer de la validité de l'allocation qui sera sélectionnée, l'étape d'allocation inclus une étape d'analyse des temps de réponse des tâches, basée sur l'analyse proposée dans cette thèse pour produire un majorant sûr des WCETs des tâches. L'analyse permet également de s'assurer que, pour l'allocation sélectionner, il existera au moins un plan d'ordonnancement dans lequel toutes les échéances temporelles des applications seront respectées.

Génération d'Ordonnancement Une fois l'allocation sélectionnée, l'étape suivante consiste à configurer le plan d'ordonnancement statique qui sera suivi à l'exécution. Dans la startégie "one-to-one", les tâches sont supposées non préemptives, et l'ordonnancement est généré au niveau tâche. Chaque tâche appartenant à une partition donnée, le plan d'ordonnancement niveau partition est directement déduit du plan d'ordonnancement niveau tâche.

Les données d'entrée de cette étape sont les sorties de l'étape d'allocation logicielle/matérielle. Les inconnues sont les dates de début d'exécution des tâches. Les WCETs de chaque instance de tâche sont également des inconnues, déterminées grâce à l'analyse d'ordonnancement proposée dans cette thèse et qui est exploitée lors de l'étape de génération d'ordonnancement. Cette analyse sert notamment à s'assurer que les dates d'activation des instances des tâches sont cohérentes avec les échéances temporelles des applications logicielles correspondantes. La sortie de l'étape de génération d'ordonnancement est alors un plan d'ordonnancement statique pour lequel l'analyse a vérifié qu'il mènera toujours au respect des échéances temporelles à l'exécution.

Evaluation

Les contributions de thèse ont été évaluées sur plateforme réelle, c'est-à-dire sur un processeur multicoeur identifié comme potentielle cible pour utilisation dans un système avionique. Nous avons sélectionné le processeur P4080 de Freescale [9]. Le RTOS exploité pour l'ordonnancement est le VxWorks653 3.1 Multicore Edition de Wind River Inc [14]. Les problèmes de programmation par contrainte ont été résolus en utilisant le solveur de contraintes de IBM/ILOG, CP Optimizer [6]. Pour effectuer l'analyse des temps d'exécution en isolation des tâches, nous avons exploité le logiciel aiT Analyzer d'Absint [2].

L'évaluation des deux stratégies a révélé leur pertinence et sûreté, par l'absence de violation d'échéance temporelle, dans toutes les configurations testées sur plateforme réelle et pendant toute la durée de l'observation du comportement du système.

Le gain en optimisation des systèmes via l'exploitation de la programmation par contraintes a été observé lors de la comparaison des résultats des stratégies d'intégration "one-to-one" et "one-to-all" à une stratégie basée allocation et ordonnancement ignorant complètement les interférences multicoeur. Cette dernière stratégie correspond à l'état de l'existant avant notre thèse, à notre connaissance. Les résultats montrent que les configurations sélectionnées en sortie de l'une ou l'autre de nos stratégies a toujours un taux d'utilisation total plus bas que dans les solutions exhibées par la stratégie ignorant les interférences.

Pour les deux stratégies proposées dans cette thèse, chaque problème de programmation par contraintes est résolu en quelques secondes. Une allocation et un plan d'ordonnancement générés manuellement auraient pris plusieurs jours, ne serait-ce qu'à cause du calcul de WCETs et WCRTs qui sont complexes. Sans compter qu'il serait très difficile de générer un ordonnancement optimisé manuellement, ce qui aurait pris plusieurs tentatives manuelles. Par opposition à une recherche manuelle, il est plus facile d'essayer différentes configurations, les valides et les invalides étant clairement et rapidement exposées par le solveur de contraintes. De plus, l'utilisation de l'une de nos stratégies d'intégration supprime le risque de détection tardive de configurations non valides, les vérifications temporelles étant menées en simultané avec l'allocation et l'ordonnancement respectivement.

Finalement, une autre nouveauté apportée aux concepteurs de systèmes par nos stratégies d'intégration est le gain de visibilité sur l'impact des interférences multicoeur sur l'existence de configurations valides. Le processus d'intégration est maintenant plus flexible, car le gain de temps et d'efforts permet aux concepteurs de systèmes d'effectuer des vérifications supplémentaire, d'essayer de nouvelles combinaisons d'allocations (par exemple allouer plus d'applications au même module), etc.

Pour résumer, les objectifs de la thèse en ce qui concerne la compatibilité des starégies d'intégration proposées avec les contraintes de certification ont été atteints. Les objectifs secondaires de l'optimisation de la conception et de la réduction du time-to-market ont également été atteints grâce à l'exploitation des techniques de programmation par contraintes pour automatiser les étapes de chaque stratégie d'intégration.

En plus des aspects fonctionnels des contributions de cette thèse, la généricité des stratégies proposées a également été vérifié. Tout d'abord, on peut remarquer l'indépendance des stratégies vis-à-vis de la plateforme matérielle exploitée et du RTOS: les seuls adhérences se trouvent au niveau de la modélisation de la plateforme multicoeur, et ces adhérences sont soit: (i) des éléments basiques de modélisation multicoeur, comme par exemple la fréquence CPU; (ii) soit des paramètres matériels caractéristiques du composant modélisé, présent dans tout composant du même type et dont la valeur est obligatoirement fournie par le concepteur de processeur. Ainsi, les travaux présentés dans cette thèse peuvent être utilisés pour tout multicoeur basé bus, et ne se restreignent donc pas à une unique famille de processeurs particulière.

Conclusions

La stratégie d'intégration ”one-to-all” est basée sur une approche statique SMP où, au moment de l'exécution, une seule application est exécutée sur tous les noyaux d'une plateforme multicoeur. La stratégie d'intégration ”one-to-one” repose sur une approche AMP statique où chaque noyau est considéré comme un processeur distinct à l'exécution, chaque coeur ayant son propre schedule.

Pour couvrir la vérification temporelle du système, les deux stratégies comprennent: (i) une analyse de faisabilité exploitée pendant la phase de configuration de l'allocation, et (ii) une analyse d'ordonnancement exploitée dans le problème d'ordonnancement. L'analyse de faisabilité est basée sur une dérivation de l'analyse du temps de réponse pour s'adapter aux architectures IMA et aux plateformes multicœurs. Un modèle d'interférence a été proposé afin de produire un majorant supérieur des délais d'interférence dans la situation la plus défavorable subies par chaque tâche. L'analyse d'ordonnancement réutilise le modèle d'interférence, ainsi que la connaissance des plans d'ordonnancement des autres coeurs respectivement afin de calculer des majorants supérieurs de WCET pour chaque instance de tâche dans une hyperpériode, et vérifie que toutes les échéances temporelles sont respectées.

Enfin, chaque étape de chaque stratégie d'intégration est exprimée sous la forme d'un problème de programmation par contraintes, afin de permettre l'automatisation des activités d'allocation, génération et vérification d'ordonnancement. De plus, l'exploitation des techniques de programmation par contraintes permet également d'économiser du temps et et raccourcir le cycle de conception du système, mais aussi de réduire les efforts à fournir pour effectuer les différentes vérifications. Cela permet également d'optimiser les choix d'allocation et ordonnancement, via la définition de fonctions objectives dans chaque problème exprimé, afin de sélectionner la solution de configuration la plus optimisée suivant des critères industriels.

Degré de Confiance des Analyses Temporelles Deux analyes sont proposées. La première est une analyse de faisabilité, ou analyse temporelle servant à déterminer si, pour une allocation des applications et tâches sur les coeurs d'un multicoeur donnée, il sera toujours possible de trouver un plan d'ordonnancement dans lequel toutes les échéances temporelles de la plateforme logicielle seront toujours respetées à l'exécution. La seconde est une analyse d'ordonnancement, ou analyse temporelle servant à déterminer si, pour une allocation et un algorithme d'ordonnancement donnés, toutes les échéances de la plateforme logicielle seront respectées à l'exécution.

Les deux types d'analyses proposées reposent sur un modèle statique d'interférence. Aucune hypothèse n'est faite sur les dates d'arrivée des requêtes mémoire au niveau de l'interconnect ou bien des contrôleurs mémoires. Le but des modèles construits est de dériver une enveloppe supérieure des situations pire cas d'interférence inter-coeurs de manière à produire un majorant absolu des WCET et WCRT des tâches d'une application IMA. Ainsi, les valeurs d'interférence calculées pour chaque tâche correspondent à une valeur toujours supérieure à la réelle durée d'interférence que cette même tâche peut subir dans la pire situation possible. On peut dire que les analyses proposées ont la même valeur qu'une analyse exhaustive, sous couvert de l'hypothèse de composabilité de la plateforme matérielle exploitée. A ce titre, l'analyse résultant est sûre, par opposition à une analyse basée sur un échantillon de mesures par exemple.

Modularité et Flexibilité des Modèles d'Interférence et d'Analyses Temporelles Proposés. Les analyses temporelles présentées dans cette thèse permettent de vérifier la validité des coix de configurations faits lors de l'intégration de la plateforme logicielle sur la plateforme matérielle. L'un des avantages notoires du modèle d'interférence proposé est la modularité des équations du modèle: il est possible de modifier ou raffiner les modèles d'interférences de manière rapide et sans nécessiter une refonte complète des aalyses temporelles exploitant les équations modélisant les interférences.

Une telle modularité rend également flexible l'utilisation des analyses proposées. Dans les premières phases de développement d'un nouveau logiciel IMA, il est possible de remplacer les équations détaillées du modèle d'interférence par un modèle gros grains pour se faire une estimation des interférences encore plus en avance de phase. L'analyse permet alors de donner, pour chaque allocation ou plan d'ordonnancement évalué, une réponse quasi instantanée quant à la faisabilité du respect des échéances temporelles.

La flexibilité des modèles d'interférence proposés se manifeste également de par le fait qu'ils peuvent être utilisés à la fois dans la vérification de faisabilité d'une allocation, et dans la vérification d'ordonnancement une fois des dates d'activations sélectionnées. Malgré le fait que les tâches et les partitions soient modélisées via des paramètres différents dans les étapes d'allocation et d'ordonnancement, aucune modification des modèles d'interférence n'a à être faite, la différence étant faite de façon logique au niveau des équations de calcul de WCET pour l'allocation et pour l'ordonnancement.

En conclusion, de telles caractéristiques confèrent aux techniques d'analyse temporelle proposées dans cette thèse une propriété significatie pour l'industrie: l'adaptation à l'évolution des plateformes, à la gestion d'osbolescence, mais ausi l'adaptation des modèles d'interférences aux avancées réalisées dans la littérature au fil des années futures.

Portabilité et Réutilisation des Contributions. Les deux principaux freins à l'extension des techniques d'analyse statique monocoeur aux architectures multicoeur sont:

- La complexité des architectures multicoeur. Les ressources partagées impliquentun certain nombre de couplage de chemins conduisant à l'explosion du nombre d'états à considérer dans le cadre d'une analyse temporelle exhaustive comme le sont les analyses statiques monocoeur exploitées dans l'avionique.
- Le manque d'informations sur certaines ressources partagées dû à des protection de propriété intellectuelle par les fabricants de processeurs. Ce manque vient augmenter le problème des ressources dont le comportement n'est pas déterministe et ne peut être modélisé de manière pertinente, par exmeple les ressources dont l'arbitrage exploite une heuristique basée sur les performances moyennes et les derniers arbitrages passés.

A notre connaissance, aujourd'hui, toute tentative pour réaliser une analyse WCET au niveau code et de manière statique a conduit à une explosion du nombre d'états à considérer dans ladite analyse, et donc à un problème intractable.

Les analyses temporelles proposées dans cette thèse sont des analyses statiques, et ne tombent pas dans ce problème d'explosion du nombre d'états ni d'intractabilité. Cela est dû au choix stratégique d'effectuer l'analyse à un degré d'abstraction plus élevé: au niveau ordonnancement plutôt qu'au niveau code. Les modèles des ressources partagées causant des interférences reposent uniquement sur des paramètres essentiels de ces ressources, en ce sens qu'il s'agit de paramètres dont le fabricant est tenu de donner la valeur lors d ela commercialisation de la ressource correspondante. Ainsi, cette thèse a su gérer à la fois le problème de complexité des architectures multicoeur puisque les contributions ne risquent pas de tomber dans le problème d'explosion du nombre détats et d'intractabilité, et le manque d'informations par protection intellectuelle.

Pourquoi Deux Stratégies d'Intégration Le fait de proposer deux stratégies différentes pour l'intégration a été motivé par les différentes propriétés des deux propositions. Bien que la stratégie d'intégration "one-to-all" respecte toutes les exigences obligatoires pour la certification des systèmes aérospatiaux, la stratégie "one-to-one" offre autant de réduction du nombre de plateformes embarquées dans un système avionique que possible (c'est-à-dire en utilisant des techniques d'analyse compatibles avec DO-178) pour les fonctions DAL A tout en utilisant l'analyse temporelle proposée dans cette thèse. Cependant, la stratégie d'intégration "one-to-one" ne premet pas de répondre à l'exigence de partitionnement robuste pour les fonctions DAL A, ce qui implique de déterminer un autre moyen que le partitionnement robuste pour empêcher la propagation de faute en cas de problème, et parvenir à convaincre les autorités de certification de la validité du moyen mis en place. Cela signifie qu'en cas d'usage pour des plateformes nécessitant un partitionnement robuste des applications, la stratégie "one-to-one" nécessite un savoi suppélmentaire non disponible aujourd'hui, contrairement à la stratégie "one-to-all", qui pourrait immédiatement être utilisée dans un contexte industriel.

La stratégie d'intégration "one-to-one" est tout de même proposée comme contribution de cette thèse, car elle peut être utilisée:

- Pour des fonctions moins critiques pour lesquelles il n'est pas obligatoire d'interdire les accès simultanés aux ressources partagées au moment de l'exécution, et pour lesquelles un processeur de développement incrémental n'est pas requis.

- Pour la conception de modules IMA dans lesquels une seule application par module est intégré, chaque application étant divisée en plusieurs partitions. Dans ces cas, toute interférence inter-coeur demeure intra-application. Ainsi, on peut considérer que les modules résultants respectent la contrainte de partitionnement robuste des applications IMA. Cela vise tout DAL, y compris les applications DAL A.

- Pour l'intégration non IMA, c'est-à-dire pour porter un logiciel non IMA – les commandes de vol par exemple – sur un processeur multicoeur.

Pour de telles fonctions, l'utilisation de la stratégie d'intégration "one-to-one" conduit à des conceptions optimisées dans lesquelles la réduction du nombre de processeurs à embarquer pour intégrer toutes les applications avioniques du système a été réalisée autant que possible. Dans toutes les autres configurations, c'est la stratégie d'intégration "one-to-all" qu'il faudra exploiter. Ainsi, les deux stratégies d'intégration proposées dans cette thèse sont complémentaires l'une de l'autre.

Perspectives

Les stratégies d'intégration et analyses temporelles associées dans cette thèse fournissent une base de travail pour l'allocation logicielle/matérielle et la génération de plan d'ordonnancement vérifiées en avance de phase pour des systèmes IMA basés sur multicoeur. A ce titre, certaines pistes de travaux futurs peuvent être identifiées, comme par exemple l'expérimentation de différents critères d'optimisation pour les problèmes degénération d'ordonnancement définis dans chaque stratégie d'intégration.

En dehors de la charge CPU, une préoccupation commune des concepteurs de systèmes avioniques est le nombre de changements de contextes de partitions impliqués par un plan d'ordonnancement donné. Ces changements de contexte représentent des délais venant s'ajouter à l'exécution des traitements logiciels de l'application concernée, et ils ne sont en général pas comptabilisés dans les budgets de temps CPU définis pour chaque fenêtre de partition. En effet, la commutation de partition n'est pas gratuite et le temps passé à changer les contextes peut entraîner des dépassements de fenetres temporelles des partitions dans une hyperpériode. Par conséquent, il pourrait être intéressant de chercher à optimiser les plans d'ordonnancement générés par nos problèmes de programmation par contrainte de manière à minimiser le nombre total de changement de contexte de partition. A notre connaissance, la littérature n'offre aucune approche – autre qu'empirique – pour la modélisation des changements de contexte [106, 75]. La raison principale tient à l'adhérence forte des délais de changement de contexte, à la plateforme matérielle exploitée. Les traitements réalisés lors d'un changement de contexte dépendent de l'OS et du processeur utilisés, et ne sont pas divulgués dans les détails par les fabricants correspondants. Pallier à ce manque pourrait correspondre à une perspective future intéressante d'amélioration des travaux de cette thèse.

Une autre optimisation possible des plans d'ordonnancement générés est la réduction des interférences inter-coeur. Celle-ci peut être réalisée en réduisant autant que possible le nombre total d'exécutions parallèles sur les différents coeurs du multicoeur exploité. Un objectif d'optimisation encouragerait la sélection de plans d'ordonnancement dans lesquels aucune tâche exploitant le même chemin vers la mémoire ne sont ordonnancées simultanément, les seules exécutions parallèles alors plannifiées visant des couples de tâche qui ne peuvent interférer les unes avec les autres. Deux tâches n'interfère pas si elles ne sont pas allouées au même contrôleur mémoire par exemple.

Définir une fonction objectif minimisant les interférences signifierait alors que, chaque fois que cela est possible, le solveur de contraintes guidera le processus de résolution du problème d'ordonnancement vers des plans d'ordonnancement dans lesquels aucune interférence inter-application n'a lieu. Il est important de noter que si une telle situation est rencontrée, cela signifie qu'il est éventuellement possible d'augmenter le nombre de partitions pouvant être intégrées sur le même module. Cela favorise encore plus la réduction du nombre de processeurs embarqués dans le système correspondant, et donc la réduction du volume et de la charge totale du système final.

Outre l'optimisation de la conception, une autre piste intéressante serait de raffiner davantage le modèle d'interférence exploité dans les analyses temporelles, de manière à couvrir plus de ressources matérielles partagées par les coeurs d'un multicoeur au moment de l'exécution. Compte tenu de la complexité et de l'explosion du nombre d'états à analyser lorsque l'on modélise toutes les ressources partagées, un compromis doit être trouvé entre la détermination d'un bon niveau d'abstraction du modèle d'interférence et le nombre de ressources partagées représentées dans le modèle d'interférence. Par exemple, pour encourager la réutilisation de l'état de l'art, la représentation des interférence multicœur proposée par Altmeyer et al. [19] est plus détaillée que dans cette thèse, et par conséquent pourrait produire des majorants superieurs moins pessimiste des WCRTs et WCETs des tâches. Cependant, ces travaux sont concentrés sur l'analyse d'ordonnancement, et partent d'une allocation et un plan d'ordonnancement prédéfinis en entrée du problème; d'autre part, l'ordonnancement hiérarchique telle que définie dans les architectures IMA n'est pas considérée. Ainsi, il serait intéressant de chercher à fusionner les travaux d'Altmeyer et al. avec les contributions de cette thèse.

Pour finir, comme mentionné précédemment, en tant que premiers travaux vers une stratégie d'intégration complète pour l'exploitation de multicoeurs dans les systèmes avioniques, nous nous sommes limités à un seul processeur multicoeur pour étudier les interférences, l'allocation et l'ordonnancement d'applications IMA. Les systèmes avioniques intègrent généralement plus d'un processeur à bord, reliés entre eux à travers différents réseaux d'interconnexion. Maintenant que cette thèse apporte des techniques d'analyse temporelle IMA multicoeur, il serait intéressant de généraliser nos contributions de thèse à une plateforme matérielle constitué de plusieurs multicoeurs interconnectés. La palteforme logicielle alors considérée consisterait en l'intégralité des applications logicielles devant être embarquées dans le système avionique à concevoir. Le problème d'allocation chercherait à allouer les applications aux différents multicoeurs, et l'ordonnancement comprendrait les délais de communication inter-modules.

Dans la littérature, les travaux cherchant à traiter ces deux activités font habituellement l'hypothèse qu'à l'échelle de chaque procésseur pris séparément, il sera possible de trouver une allocation et un ordonnancement valides. Ainsi, notre thèse est complémentaire de ce genre de travaux. Il serait alors intéressant de combiner les deux ensemble pour gérer l'intégration des systèmes distribués exploitant des multicoeurs.

Organisation du Manuscrit

Les premiers chapitres constituent les fondements de cette thèse. Le chapitre 1 présente les défis posés par les multicores pour l'industrie aérospatiale, la problématique de cette thèse, ainsi qu'une brève description des contributions de cette thèse.

Le chapitre 2 fournit les connaissances de base nécessaires pour comprendre les contributions de cette thèse: la terminologie est brièvement présentée avant de décrire le concept IMA et les réglementations de certification imposées sur les systèmes aérospatiaux. Le chapitre se poursuit ensuite avec une description des connaissances nécessaires pour comprendre la théorie de l'ordonnancement et les théorèmes exploités comme base des contributions de thèse.

Le chapitre 3 présente un aperçu de la littérature autour des challenges posés par l'intégration de systèmes temps-réel dans des environnements multicœur.

Les quatre chapitres suivants présentent les contributions de cette thèse. Sauf mention explicite dans ces chapitres, l'ensemble des travaux décrits dans ces chapitres est la seule contribution de l'auteur de cette thèse.

Le chapitre 4 présente tous les modèles et paramètres exploités dans les contributions de thèse, les hypothèses de cette thèse ainsi qu'une présentation générale des stratégies d'intégration proposées dans cette tèshe. Ensuite, sont présentés les modèles d'architecture logicielle et matérielle qui sont considérés dans cette thèse. Le chapitre termine par une brève description des contraintes considérées dans les problèmes d'allocation et d'ordonnancement abordés dans cette thèse.

Le chapitre 5 présente les méthodologies d'analyse temporelle qui ont été mises en œuvre dans cette thèse. En particulier, le modèle d'interférence et les équations exploitées pour le calcul des WCET et temps de réponse pire cas (WCRT) des tâches sont exposés. Deux types d'analyse temporelles sont propsées, en tant qu'analyses exploitées respectivement lors de la recherche d'allocation logicielle/matérielle et lors de la configuration d'un plan d'ordonnancement. Le chapitre se termine par une sous-partie résumant les analyses proposées et discutant de leurs avantages et de leurs limitations.

Le chapitre 6 présente les deux stratégies proposées dans cette thèse pour l'intégration IMA sur COTS multicoeur. Un bref aperçu de chaque stratégie d'intégration est donné en premier, avant de présenter séparément – dans les détails et dans l'ordre chronologique – les activités réalisées dans chaque stratégie. Le chapitre se termine par une sous-partie qui résume les stratégies d'intégration proposées, puis discute des avantages et des inconvénients de chaque stratégie.

Le chapitre 7 présente l'évaluation des travaux réaalisés dans cette thèse: des explications sur les expérimentations menées sont d'abord fournies, avant que plus d'information sur la façon dont la plateforme logicielle de test a été construite ne soient données. Le chapitre poursuit avec la présentation et l'analyse des résultats obtenus, et se termine par une section résumant les résultats généraux et discutant des réalisations et des limitations rencontrées lors de la mise en place des activités de test.

Pour terminer, le chapitre 8 propose un bref résumé des contributions proposées dans cette thèse, ainsi qu'une discussion globale en ce qui concerne le respect des objectifs principaux de cette thèse, avant de donner une conclusion générale sur les réalisations et de finir avec les travaux futurs que cette thèse propose en perspectives.

Chapter 1

Introduction

1.1 Motivations

For hard real-time systems, predictable delays for software runtime execution are as important as the correctness of the computational results. In particular for safety-critical systems, a failure to deliver correct results that are both correct and on time may lead to a system-level dysfunction with severe consequences, such as significant loss of money or human life. Aerospace systems fall into this category of systems, and must therefore undergo a strictly regulated design and verification cycle coupled with a strict certification process before being allowed to be operational and commercialized. For instance, regarding the safety critical software embedded in an avionic system, each software operation must be analyzed at design time in order to derive a safe upper-bound on its execution time duration at runtime, but also in order to configure accordingly and in advance the runtime behavior of the entire system, all the while being able to show the corresponding verification results as a justification to certification authorities to convince them of having successfully done so.

Certification is very important when designing an aerospace system, and represents a significant amount of effort and verification in order to achieve compliance to certification regulations. Whenever a modification is performed on a part of an avionic system, all parts of that system which are impacted by the modifications must be re-verified and their compliance to certification regulations re-proven.

One way to try and reduce the additional time and effort spent due to design modifications is to adopt a design approach favoring modularity and independence. For instance, the software embedded in an avionics system represents several applications, which are not necessarily developed by the same application supplier. However, each application is developed independently from one another for safety reasons. This also enables designers to make sure that each application can be analyzed independently from each other, so that modifying one function has no impact on all other functions. As such, only the modified function(s) will need to be re-verified and re-certified; the resulting development, verification and certification processes are then said to be *incremental* [172].

In addition to incremental certification, the modularity of a system is also enhanced by implementing an Integrated Modular Avionics (IMA) architecture [1], according to which software applications are further divided into sub-elements that are to be executed strictly independently from each other at runtime, thus ensuring a temporal and spatial isolation. Such a separation is often referred to as *time and space partitioning.*

To sum up, the concepts of incremental certification and IMA architecture both ensure the modularity of an avionic system, and require verification analyses to be performed in the same spirit of separation of concerns. In that sense however, the knowledge and experience of avionic system designers are crucial to certify avionics systems.

The software embedded in aerospace systems is preferably integrated on hardware platforms that have a long in-service history in the aerospace industry, and which are deemed reliable according to systems designers expertise. The main motivation behind such a choice is to rely on electronics which has had time to be extensively tested and has then been proven to be compatible with all requirements of the corresponding avionic systems. Doing so facilitates the design of future avionic systems as well, since it is easier to reuse hardware which behavioral characteristics are well known from experience due to usage in previous systems.

Until now, hardware platforms embedded in avionic systems contain only single-core processors. However, now that processor manufacturers are capable of producing powerful multicore processors, it is predicted that manufacturers will definitively stop producing single-core processors, including the ones embedded in current avionics systems. Avionics industries do not represent a market share big enough for processors manufacturers. As such, processors manufacturers would not benefit from keeping producing single-core processors only for avionics industries. As a consequence, avionics industries will soon have no other choice than to move to multi- and/or manycore processor designs.

The advent of multicores can be beneficial for design and cost optimization by reducing the embedded weight and energy in avionic systems, one core being able to handle the load of one previous single-core. However, all certification regulations and design procedures linked to software must be updated in order to take into account multicore-based hardware platforms [109]. For instance, a new definition of IMA architecture for multicore platforms must be set, since by nature multicore architectures go against the principle of independence of software execution at runtime. Indeed, because of the concurrent accesses to shared resources inside the multicore chip occurring at runtime, temporal isolation is difficult to achieve at runtime on a multicore platform.

Therefore, all analysis and verification activities usually taking place in a system design cycle must be updated with tools and techniques capable of analyzing multicore platforms. Inter-core interference at runtime must be safely bounded during the analysis in order to be able to safely configure in advance the runtime behavior of the system at design time. Only then, system designers would be able to properly use multicore platforms for aerospace systems and to rebuild the knowledge and know-how necessary to master multicore-based avionics systems.

Such changes are problematic in the case of software porting from single- to multicore systems, mainly because of the current lack of software analysis tools or techniques capable of upper-bounding tasks Worst-Case Execution Times (WCET) safely [171]. Concurrent resource access requests generated by each core cause additional interference at runtime, leading to significant waiting delays increasing tasks execution times. In addition, the level of complexity and coupling of the resource sharing in multicores makes it difficult or even impossible for current timing analysis techniques for safe WCET upper-bounding to be adapted to multicore environments without becoming intractable or overly pessimistic. Another difficulty adding up to the complexity of shared interference is the fact that information on the multicore architecture that is crucial for the WCET analysis is undisclosed in most Components Off-The-Shelf (COTS) multicores, due to protection of intellectual property rights and patented designs. Moreover, some request arbitration policies might be too complex to be properly represented as well, so that there is no certainty that any workaround proposed to cope with these issues will lead to a safe timing analysis for multicore COTS.

In the end, there currently exists no solution to the challenge of safe WCET analysis on multicore. Without such solution, resource contentions cannot be properly bounded beforehand, so that there is no way of configuring a safe runtime behavior in advance, nor of ensuring in advance that all deadlines of a system will never be violated at runtime. Such lack of safe WCET analysis techniques for multicore systems is jeopardizing the exploitation of multicore platforms in aerospace systems, which makes it one of the most studied research topic both in academia and industry [90, 129, 119, 66, 64].

As mentioned before, the difficulty of analyzing timing behavior on multicore systems exists for all real-time systems. The literature is full of various propositions to tackle the problem at hand [64]:

- Some works propose new middleware services to monitor every access to shared hardware resources, in order to enforce deterministic runtime behaviors by getting rid of inter-core interference [46];
- Other works propose new multicore designs where each core has its own private resources [7, 181, 149];
- Similar works focus on implementing new hardware devices to be added to an existing architecture in order to control the sharing of some resources, hoping that such new devices will either be implemented in custom multicore processors for industries or selected by platform vendors for their next COTS generation [111, 130, 170];
- A different kind of strategy that exists in the literature as well consists in proposing new execution models for multicore-friendly software development [58];
- Other works propose new timing analysis techniques for shared resources interference bounding or estimation, relying or not on some software and/or hardware modifications [19, 67, 85, 91, 111, 116, 118, 122, 127, 51].

Each approach has its own advantages. However, the first drawback is that all of them are not considered sufficient or valid for the certification of IMA architectures. The other main drawback common to all approaches in the literature is the non respect of key aerospace systems requirements, to which they consequently cannot be applied without high costs, dependence to some manufacturer or giving up some key principles of the current certification acceptance process.

This thesis has been initiated and funded by Airbus Group Innovations Toulouse, as a CIFRE (training contract for performing research in private companies) PhD with the IRIT lab in Toulouse. The main goal of the thesis is to propose an approach for integrating legacy IMA software – i.e. sequential software developed for single-core platforms – onto a multicore processor, without any hardware nor software modification: the approach much comply as much as possible with incremental certification acceptance process and key IMA concepts. To do so, multicore interference must be taken into account in the proposed contributions. In this thesis, we focus on inter-core interference due to sharing access to the main memory.

The motivation of this thesis is to stick as much as possible to current industrial processes, in order to maximize the chances of acceptation of the proposed contributions by aerospace industries and certification authorities by minimizing the changes and effort of adaptation that would be necessary to adopt the newly proposed process.

A secondary goal of this thesis is to improve design optimization by benefiting from the performance gain that can potentially be brought by multicore processors.

Finally, the following list gathers strong choices that have been selected as guidelines for our contributions, in order to increase their chances of being considered as a solution for the next generation of aerospace systems:

- The proposed contributions must be applicable to DAL A IMA applications. Indeed, as briefly mentioned before, every attempt of solution proposed in the litterature is either not applicable to IMA systems, or requires a high amount of modifications and efforts in order to be used for IMA systems; moreover, no solution targeting IMA systems covers all of the multicore challenges hindering the usage of multicores in avionics systems.
- The proposed contributions should not be specific to one particular hardware platform or rely on the know-how of some specific supplier;

- In particular, the contributions should not rely on any hardware modification or assume specific hardware architecture characteristics, so that COTS platforms can be used instead of implementing ASICs or relying only on one specific COTS product line or one specific COTS manufacturer. The contributions should rely only on COTS-friendly configuration mechanisms instead, so that no special additional hardware is needed;
- The contributions should not involve any software modification as well, so that legacy software can be reused and no special additional study and/or regulations are required to certify the software platform considered, and the overall approach proposed for multicore integration can be used in an as nearest future as possible;
- The approach proposed in the thesis should reuse as much as possible existing works from the literature to cope with multicore challenges, even if these works represent only partial solutions to the multicore challenge. Our motivation is to promote research-to-industry transfer; however, it is also important that only realistic techniques should be considered, fitting real world constraints of industrial systems and processes;
- In the same line of idea, the realism of the proposed solutions is mandatory, even if it means designing under-optimized systems. In particular, it is important not to make any assumption that is unapplicable to current COTS multicores or ignores current unavoidable situations at runtime.
- The thesis outcome should promote as much automation as possible in the proposed contributions, in an effort to shorten the time-to-market but also to reduce additional efforts to be made by suppliers and integrators when handling new technologies such as multicores in future aerospace systems.
- Finally, the thesis contributions should respect the current development and certification processes as much as possible. Even if they are not suited for multicore environments, respecting as many legacy design and certification rules as possible is truly likely to help reducing the time that would be spent negotiating with certification authorities to investigate new mechanisms and configuration choices.

The proposed contributions are based on a generic, bus-based multicore architectural model. All assumptions about affecting the hardware configuration can be implemented using classic configuration choices usually offered in COTS platforms.

1.2 Contributions

This thesis proposes two complete methodologies or strategies for IMA integration on multicore COTS. Each one has different advantages and different drawbacks, and therefore may respectively be used for different situations and systems to be designed. One strategy fits all avionic and certification requirements but is likely to result in poorly optimized designs, and the other one generates configurations that are as optimized as possible but does not respect some of the current key certification requirements, which makes it not suitable for applications with the highest levels of criticality.

The proposed integration strategies are said to be ”complete” because we provide all the metrics necessary to go through all steps of the proposed integration strategies. They cover the allocation of the software platform onto the hardware platform, the schedule generation but also timing analyses. To sum up, the contributions of this thesis consist in:

- Interference-aware timing analysis models and techniques for early timing verification and enforcement of a deterministic runtime behavior of the system: in particular, we propose (i) a mathematical model of shared resources exploited to derive a safe upper-bound on tasks worst-case interference delay due to each shared resource respectively, and (ii) an approach for performing a safe response time analysis for IMA architectures in multicore environments. The resulting analysis has been derived into two different approaches: one dedicated to verifying the validity of a software/hardware allocation by performing a response time analysis, and one dedicated to deriving safe WCET upper bounds for tasks and CPU time windows for partitions when an allocation and a schedule have been set. This contribution is presented in details in chapter 5, and has been published in [116].

- A Constraint Programming (CP) formulation for performing automated, optimized and safe software/hardware allocation of IMA applications onto multicore platforms. The approach is safe in the sense that the feasibility of the solution selected by the constrained program is ensured since the timing analysis implemented in this thesis is embedded as a constraint of the CP. This contribution is covered in chapter 6 and has been published in [118].

- A CP formulation for performing automated, optimized and safe schedule generation for IMA applications on multicore platforms. It is said to be safe because the constrained program embeds the schedulability analysis proposed in this thesis as a constraint, in order to ensure the validity of the solution selected by the constrained program. This contribution is covered by chapter 6 and has been published in [116, 118]. This contribution has been mentioned in our pulications [116, 118, 117].

- Two processes for safe IMA integration on multicore platforms: as mentioned before, one of the two respects all most important certification characteristics of current IMA systems, and the other offers maximum design optimization for less critical IMA applications. Each strategy covers the allocation of the IMA software platform onto the multicore platform, and the generation of a schedule for each core of the multicore processor. To do so, safely and automatically in order to save time and effort, the three previously mentioned contributions of this thesis are exploited. This contribution is extensively explained in chapter 6, and has been published in [117].

The work presented in this thesis has been evaluated on a real platform. The evaluation has been performed on the Freescale/NXP QorIQ P4080 processor [9]. A software case study has been built by the author of this thesis by exploiting open source code from the TacleBench benchmark suite [13]. We applied the proposed strategies for allocating and scheduling the constructed IMA case study on the P4080 processor [9] using Wind River IMA RTOS, VxWorks653 3.1 Multicore Edition [14]. The evaluation consists in implementing on the P4080 target the configuration selected when applying one of our two strategies, and verifying that no deadline was missed when observing the system's runtime behavior.

The following paragraphs describe in more details each proposed contribution.

Interference Bounding and Safe Multicore Timing Analysis

This thesis proposes a sufficient feasibility test that is interference-aware, and compatible both with IMA applications and both homogeneous and heterogeneous multicore processors. To do so, we propose a mathematical model of the main memory and interconnect to be exploited to perform a static timing analysis and derive safe upper-bounds of tasks interference delays due to shared resources. The resulting analysis includes safe WCET and WCRT upper-bounding. The model is based on an extension of the classic response time analysis in order to fit IMA architectures and multicore environments. It also includes the computation of safe upper-bounds on multicore interference. Such bounds are crucial to formally prove the feasibility of the system, but also to build a safe schedule which has been verified beforehand to guarantee a deterministic behavior at runtime. Detailed explanations on the proposed timing analyses are provided in chapter 5.

Allocation Problem

In this thesis, we use the expression *allocation problem* to address concerns about how the software platform should be spatially allocated to the cores and memory of a multicore platform, i.e. (i) on which core a given partition (resp. task) is going to be scheduled at runtime, (ii) where its memory context should be stored in DRAM and (iii) if more than one memory controller is present on the hardware platform, which one it will exploit at runtime. No mention of time management or time slicing is mentioned in such problems.

As a second contribution, we exploit the resulting IMA, multicore interference-aware timing analysis described in the previous subsection, in order to implement a CP formulation for the software/hardware allocation. The proposed timing analysis consists in a sufficient condition for feasibility, and is defined as a constraint to be verified by the variables of the allocation problem. During the allocation process, the module integrator decides, for a given module consisting of a given multicore processor, on which core each partition or task will be scheduled. In the case of partitioned scheduling on a multicore processor, the integrator should decide on which core each partition will be allocated. An allocation then refers to a static mapping of each partition or task of the IMA applications to the cores of the multicore processor.

As mentioned earlier, the safeness of the selected allocation is guaranteed by exploiting the proposed feasibility analysis to guide the design space exploration, so that the existence of a schedule where all timing requirements will always be met at runtime is guaranteed as soon as the allocation is set. Eventually, using constraint programming to set the allocation configuration enables to save time but also effort during the design cycle of a given system, the allocation problem being NP-hard and WCETs upper-bounding equations being complicated enough to easily cause inadvertent errors when computed manually.

In addition, potential time is saved when using constraint programming also because of the fact that expressing all timing constraints in the allocation problem reduces the risk of late detection of invalid allocations. Since all requirements of the system are expressed as constraints, and if one of them cannot be fulfilled by any solution of the allocation problem search space, then the expressed allocation CP is able to raise awareness of the situation to the software integrator and designers. Integrator and designers then know that some modifications must be performed until a valid solution can be found. Without such early detection of potentially problematic situations, the length of the system design cycle would have been extended. Indeed, the non validity of the selected allocation would have been detected later only, or may even not have been detected at all, which would have had catastrophic consequences at runtime, for instance a deadline miss in the context of a critical operation.

Scheduling Problem

In this thesis, we use the expression *scheduling problem* to address concerns about how elements of the software platform (tasks and/or partitions) should be scheduled on the cores of a multicore platform, i.e. how to distribute CPU time slices to the tasks and/or partitions of the software platform. In IMA systems and for any critical software in general in avionics systems, static schedules are built and configured in advance, to make sure the runtime behavior has been verified and approved in advance at design time. Such static CPU time slicing implies defining in advance time intervals, each interval being temporally defined by a duration (or time budget) and a start date (or activation offset).

The safety of the generated schedules is guaranteed by exploiting the timing analysis proposed in this thesis to verify that all timing requirements will be enforced at runtime. In particular in the scheduling problem, activation offsets and WCETs upper-bounds are computed in order to build a static schedule for partitions and/or tasks in one complete cycle of repetition of the schedule, also called MAjor Frame (MAF). In addition to guaranteeing the safety of the final schedules, the timing analysis embedded in the scheduling problem formulation guides the design space exploration when selecting activation offsets, so that the finally selected schedule is optimized as much as possible given the characteristics of the system. Eventually, analogously to the allocation problem, using constraint programming to build schedules saves time but also effort, and helps preventing late detections of invalid configurations.

Methodologies for Safe Integration of IMA

As a final contribution, we exploit the proposed timing analyses, allocation and schedule generation CP formulations to implement methodologies for the safe integration of IMA applications onto multicore platforms.

We propose two strategies. Our first integration strategy is compliant to all industrial requirements: we call it the *one-to-all integration strategy.* It is based on a static derivation of the Symmetric Multi-Processing (SMP) approach where at runtime only one application is scheduled on all cores of a multicore platform. Scheduling only one application on all cores at runtime preserves the concept of robust partitioning by creating a situation where the only kind of multicore interference is intra-application, and not inter-application, which is forbidden in current IMA systems.

The one-to-all integration strategy can be seen as a consolidated methodology for reusing legacy IMA applications on current multicore COTS, safely and while being compliant to key certification requirements of current systems, such as robust partitioning, incremental certification or static WCET upper-bounding.

Non-critical applications being submitted to less strict certification requirements than time- and safety-critical applications, and multicore processors still being under study by certification authorities in order to decide how to regulate the resource sharing challenge in future certification guidance documents for critical avionics, the requirements may eventually be relaxed for non-critical applications at least in the first regulation attempts. As such, we propose a second integration methodology which achieves as much SWaP reduction as possible while still computing safe WCET upper-bounds. This strategy relies on a static Assymetric Multi-Processing (AMP) allocation of IMA partitions to the cores of a given multicore platform. This strategy is referred to as the *one-to-one integration strategy.* It can be applied for non-IMA software, or in the context of multi-partition IMA applications, as long as all partitions allocated to the multicore platform are from the same application. It cannot be applied for allocating more than one IMA application onto the same multicore platform since tasks WCETs – and therefore, the ding partitions time slices – are computed depending on the knowledge of all cores schedules, which violates the concept of independence. The one-to-one integration strategy enables to achieve maximum SWaP reduction as can be possible using our timing analysis techniques. However, the price to pay for such optimized systems is the absence of a strong time partitioning in the resulting system.

Each strategy is divided into several activities, in order to perform the allocation, the schedule generation and all corresponding verifications. These activities are formulated as CPs in order to avoid having to perform them manually, and therefore to gain in effort, time and design optimization. The CP associated to each activity in each integration strategy proposed in this thesis will be exhaustively presented in chapter 6. As already explained earlier, all expressed CPs take into account multicore interference when performing timing verification during the corresponding configuration search process of the expressed CP.

To sum up, as a result of the combination of all our contributions in this thesis, the proposed IMA integration methodologies for multicore usage are:

- Complete, since the proposed timing analyses and the exploited models can be brought as justification to certification authorities of the fact that the produced WCET upper-bounds and schedule cover multicore interference;
- Safe, since all implemented analysis techniques are based on static analysis techniques;

- Seamless, since all integration activitites are automated using the adequate CP formulation proposed in this thesis;
- Optimized, since each CP includes optimization features through additional constraints or smart objective functions to guide the exhaustive design space exploration.

In addition, the first integration methodology is compliant to major certification requirements, which makes it a true potential solution to start multicore-based IMA integration beginning of today. Indeed, although being different, both strategies may be used for legacy software reuse on multicore COTS, would that be for critical applications, IMA or not.

1.3 Thesis Outline

The first chapters lay the fundations of this thesis. Chapter 1 introduces the challenges brought by multicores for aerospace industries, the problem statement of this thesis as well as a brief description of the contributions in this thesis.

Chapter 2 provides the background knowledge required to understand the contributions of this thesis: the terminology is briefly introduced before describing the IMA concept and the certification regulations imposed on aerospace systems. The chapter then continues with the background knowledge necessary to understand the scheduling theory behind timing and schedulability analysis.

Chapter 3 presents an overview of the literature revolving around hard real-time systems integration in multicore environments.

The next four chapters then present the contributions of this thesis. Unless explicitly mentioned in these chapters, their content is the sole contribution of the author of this thesis.

Chapter 4 introduces all models and parameters exploited in our contributions: the assumptions of this thesis and a general introduction of the proposed integration strategies are presented first. The software and hardware architecture models are then described, before presenting the constraints considered in the allocation and scheduling problems addressed in this thesis.

Chapter 5 presents the proposed timing analysis methods that have been implemented: this includes details about the interference model and the equations for the computation of tasks WCET and Worst-Case Response Times (WCRT), for the allocation and the scheduling problems. The chapter ends with a section summarizing the proposed analyses and discussing about their advantages and shortcomings.

Chapter 6 presents the two strategies we propose for IMA integration on multicore COTS. A brief overview of each integration strategy is given first, before separately presenting in details and in chronological order the activities performed in each strategy. The chapter ends with a section summarizing the proposed integration strategies and discussing about the advantages and drawbacks of each strategy.

Chapter 7 presents the evaluation of our work: we first explain which verifications are performed to evaluate our work and then explain how a case study was built for the tests. We then present the different test results and the corresponding conclusions that could be drawn. Finally, the chapter ends with a section summarizing the overall results and discussing about the achievements and shortcomings of the contributions and testing activities.

Finally, chapter 8 gives a brief summary of the proposed contributions, along with an overall discussion with regards to the primary objectives of this thesis, before giving a general conclusion on the achievements and concluding with future work related to this thesis.

Chapter 2

Background

This thesis exploits metrics and methods of the academic real-time computing world, in order to propose solutions for the industrial world. Both academic and industrial wordings are therefore employed throughout this thesis. This chapter presents all the background knowledge necessary to understand the work that has been done in this thesis, and covers both industrial vocabulary and academic scheduling theory. A brief definition of general concepts linked to the context of the thesis are presented in the ”terminology” section, while a more detailed presentation of the key concepts of this thesis are presented in dedicated sections. Readers who are already familiar with IMA systems, software integration processes and scheduling theory may skip this chapter.

2.1 Terminology

Allocation As mentioned in the introduction chapter, we use the expression *allocation problem* to address concerns about how the software platform should be spatially allocated to the cores and memory of a multicore platform, i.e. (i) on which core a given partition (resp. task) is going to be scheduled at runtime, (ii) where its memory context should be stored in DRAM and (iii) if more than one memory controller is present on the hardware platform, which one it will exploit at runtime. No mention of time management or time slicing is mentioned in such problems.

Scheduling As mentioned in the introduction chapter, we use the expression *scheduling problem* to address concerns about how elements of the software platform (tasks and/or partitions) should be scheduled on the cores of a multicore platform, i.e. how to distribute CPU time slices to the tasks and/or partitions of the software platform. Such CPU time slicing implies defining in advance time intervals, each interval being temporally defined by a duration (or time budget) and a start date (or activation offset).

Timing Analysis - Feasibility, Schedulability Analysis: in this thesis, the term timing analysis refers to the analysis of the temporal behavior of a system at runtime. Such analysis is often linked to scheduling-related activities, a schedule consisting in a chronograph of activations and executions of tasks at runtime.

Two kinds of analyses can be identified: ***feasibility analysis*** and ***schedulability analysis***. The feasibility analysis is usually performed ahead of the schedulability analysis for early verification and configuration. The classic definitions are the following.

A schedule is said to be *feasible* if it satisfies a set of constraints. A task set is said to be feasible if there exists an algorithm able to generate a feasible schedule for the corresponding task set. A task set is said to be *schedulable* with a given scheduling algorithm, if such algorithm is able to generate a feasible schedule. Performing a *feasibility analysis* consists in analyzing if a given task set is schedulable according to a given scheduling algorithm. Performing a *schedulability analysis* consists in verifying that a given schedule for a given algorithm satisfies all timing, resource and precedence constraints of the system.

As such in this thesis, a feasibility analysis is performed during the software/hardware allocation phases, and a schedulability analysis during schedule generation phases.

Worst-Case Execution Time (WCET) In this thesis, the Worst-Case Execution Time (WCET) of a task is referred to as the total time elapsed from the start of its execution and the end of the same execution, in a multicore environment. As such, the computed WCETs include the duration corresponding to the actual execution of the task entry point on the core processing unit, and also all additional overheads and waiting delays suffered by the task at runtime. One example of additional waiting delays is inter-task interference due to sharing access to the resources of the multicore platform with the tasks running in parallel on the other cores. Resources do not have an unlimited bandwidth. As such, when several tasks try to access concurrently to the main memory for instance, some of the tasks need to wait for their turn to access the memory, which results in additional overheads delaying the end of their execution.

Software Platform, Hardware Platform, Module In this thesis, all avionics software applications to be embedded in a system is referred to as *software platform*. In an analogous way, all hardware components and especially the multicore processor to be exploited as a computing resource in the corresponding system is referred to as *hardware platform*. In addition in avionics systems, a computer system composed of one or several applications embedded onto a hardware platform is often referred to as *module*.

System Designer, Application Supplier, Module Integrator: when designing an avionics system, three profiles interact iteratively with each other and in a complementary fashion. System designers are the owner of the final system to be designed, and decide of the high level requirements and configuration of the system, for instance the choice of the hardware platforms to be embedded in the system. Application suppliers are in charge of developing the software avionics functions assigned to them by system designers. The module integrator is responsible for getting all applications, and integrating them on the hardware platform so that the corresponding applications satisfy all timing requirements of the system.

Avionics Functions, Applications: several avionics functions are usually embedded in aircrafts, such as the autopilot or the fuel management system for instance. The software implementation of these functions is also referred to as applications. In this thesis, the terms avionics functions and applications are used interchangeably. Some functions may require to be implemented separately and independently from each other for safety reasons, especially to avoid common faults in both nominal and back-up systems due to fault propagation from a faulty application to a healthy one. As such, the functions to be embedded in one given aircraft can also be developed by different application suppliers, in charge of designing, testing and verifying them independently. The independence of the design and verification of each function ensures the global system to be designed in a modular fashion, limit the impact on other applications of a fault occurring in some application, and be able to seek incremental certification acceptance [172, 141].

Constraint Programming (CP): constraint programming consists in expressing a problem using mathematical equations to be solved in order to find a solution. The equations are referred to as *constraints*: a constraint is a logical relation between unknown variables. Solving a constrained program then corresponds to finding a value for each variable such that all constraints are satisfied by the variables. Each variable is defined with a predefined definition domain.

Formally, a CP consists in a tuple (X, D, C) where X is a set of variables $X = x_1, ..., x_n$, D is the function associating each variable x_i with its domain $D(x_i)$, and C is the set of constraints of the problem $C = C_1, ..., C_m$. Depending on the nature of the constraints (linear, quadratic, non-linear, etc.), different algorithms are used to solve the corresponding CP. It is important to note that the solving algorithms exploited for solving CPs are not the prior concern of this thesis, and will therefore not be discussed in this manuscript. Eventually, as more than one solution to a given CP might exist, an objective function can be defined, to express a quantity that must either be minimized or maximized. The objective function thus classifies the valid solutions so that the one with the best fitting value of the objective function will be chosen, among the solutions satisfying all constraints of the problem.

Multicore Processors: a multicore is a processor in which more than one Core Processing Unit – or CPU, or core – is embedded: the CPU is the processing unit responsible for code execution at runtime in a processor chip. To optimize average performance, multicore manufacturers often choose to embed smaller and faster memories in their processors designs, called cache memories. A cache is used to store some of the last data that have been used by a core. When a new data must be stored and the cache is full, the choice of which previous data to evict from the cache in order to store the new data in its place is done according to a specific eviction policy. Since the size of a cache is limited, COTS usually embed more than one cache, cascaded into levels and forming a hierarchy defining the order in which the cache levels are accessed. At runtime, each data to be used by the cores is first checked in the closest cache to the corresponding core; if it is not found, it is looked for in the next level, and so on until it is found either in some cache level or in the main memory.

2.2 IMA Systems in the Aerospace Industry

History

In the past, previous aircraft systems were implemented using federated architectures [167]: each avionics function was integrated on a separate computing resource, in a ”one function equals one computer” fashion. The isolation of applications implied by such an architecture promotes safe fault containment at runtime, but at the cost of aircrafts embedding excessive weight, volume and cables in order to supply enough power to and connect all the modules. Since the first aircrafts, avionics systems evolved into more sophisticated systems. Aircrafts embed smarter functionalities than in the first aircrafts, and the number of new functionalities added to modern systems compared to previous architectures keeps growing exponentially, so that the concept of ”one function equals one computer” - and thus the corresponding amount of computing resources, weight and power to be embedded in these aircrafts - could no longer be maintained.

The solution was found in a new type of architecture: the Integrated Modular Avionics (IMA) architecture [141]: in this architecture, several functions share the computing resource of the same computer at runtime. IMA successfully enabled to reduce the Space, Weight and Power (SWaP) embedded in the aircrafts in which it was implemented (for instance in the A380 and A350). In fact, by integrating more than one function into a module, IMA enables to share some components such as processing resources, power supply or I/O management services, which in turn reduces the thermal dissipation and fuel consumption of the system. Another main advantage of using IMA architectures is the ability to exploit generic modules: the IMA concept was conceived in parallel of the corresponding integration process, favoring the independence of a given application to all other applications and to the hardware platform during the design phase.

IMA cannot be applied to all avionics applications. In fact, applications integrated onto the same module are not allowed to cause interference to each other, and the mandatory condition for an aerospace system to implement IMA is to be able to guarantee a system behavior that is identical to the behavior of a system in a federated architecture. More precisely, one must guarantee the same level of segregation between applications in one module than if each function was integrated in a federated architecture, i.e. alone on a separate module. Functions of different criticality levels (or Design Assurance Level or DAL, as will be defined later in this chapter) can consequently be implemented on the same module without risking to influence one another at runtime.

Eventually, depending on the considered function, it may or may not be possible to exploit the IMA architecture, as some functions are considered too critical to be sharing CPU usage with any other function; it is the case for the fly-by-wire control command function for instance. To guarantee the same level of separation between IMA applications as in federated architectures, the concept of time and space partitioning has been implemented, also referred to as robust partitioning.

Definition

The Integrated Modular Avionics (IMA) architecture consists in a software architecture where several avionics functions share the same module. IMA software architectures are submitted to design and implementation rules stated in the ARINC 653 [1] (Avionics Application Standard Software Interface) specification, describing how to perform time and space partitioning in safety-critical avionics Real-Time Operating Systems (RTOS).

The IMA architecture divides avionics functions into ***partitions***, each partition containing one or several ***processes***. In the literature, IMA processes are equivalent to thread or ***tasks***; as such, to ease understanding of the reader by matching the IMA terminology to the literature's, IMA processes are often referred to as tasks throughout this thesis.

A function can be represented by one or several partitions, each partition defining the memory context of the tasks it contains. All processes of the same IMA partition share the same memory address space, and two processes from two different partitions cannot access each other's memory space. Such a property of IMA architectures is called ***spatial partitioning***.

Each partition is defined as periodic, the period value corresponding to the rate at which a partition is requesting CPU usage. It is linked to its own tasks respective periods. The partition period usually is equal to the Greatest Common Divider (GCD) of its tasks periods. At runtime, the execution of a task is constrained to happen within the boundaries of its partition CPU time slices. This corresponds to a strict *time partitioning*. Finally, according to the ARINC 653 standard [1], the RTOS exploited in IMA systems must support time and space partitioning in order to allow partitions of different criticality levels to be executed on the same module without affecting each other in any unexpected way.

The runtime behavior of an IMA system is managed at partition level, under the control of the system integrator, and is defined statically and maintained with configuration tables. In particular, each module has a configuration table to define the static schedule to be enforced at runtime. The module OS manages partitions, i.e., the entity scheduled by an IMA module RTOS at runtime is a partition. Partitions are scheduled according to a static, cyclic pattern defined at design time by the system integrator. The same pattern is repeated throughout the entire runtime operation of the module. The duration of that pattern is referred to as the MAjor time Frame (MAF) of the module.

During a MAF cycle, partitions are activated one after the other by being assigned one or several ***partition time windows*** or ***partition time slices***: a window is defined by (i) an offset, or activation date at which the module processing resources are reserved for a given partition runtime usage; and (ii) a duration, corresponding to a quantity of execution time exclusively reserved for the corresponding partition usage.

As mentioned before, the MAF pattern, i.e. order of activation of the partitions, as well as the partition time windows, are defined by the system integrator through configuration tables. The resulting pattern corresponds to the module schedule to be enforced at runtime. It is important to note that although the windows are defined by the system integrator, the duration of each window is computed separately by the applications suppliers and fed to the system integrator as input to be used to construct the module schedule.

At runtime during a given partition time window, the partition tasks execute concurrently on the module processing resources; a task cannot cross the boundaries of its partition time window. As such, the size of the time slices to be reserved for each partition must be assessed according to their respective tasks needs to ensure all tasks will have enough CPU time resource to complete their execution as defined by their normal behavior. However, tasks inside each partition are not directly visible outside of the partition from the point of view of the module OS and the system integrator [1]: it is therefore the job of each application supplier to derive their respective partitions CPU time needs (i.e. duration required in their time windows) according to the tasks runtime needs, in order to provide the system integrator with accurate information on the durations of each partition time window to be defined in the module schedule to be constructed.

To sum up, in IMA architectures, each function is implemented as one or several partitions and each partition contains one or several tasks. Different functions can then be integrated onto the same module provided that each partition is temporally and spatially isolated from each other at runtime. Temporal separation can be ensured by statically configuring the timing schedule to be enforced at runtime, a schedule in which each partition is given distinct time windows for its tasks execution. Spatial isolation is ensured by strictly separating memory context and hardware resources such as I/Os for each partition.

Finally, this thesis addresses the issue of software integration for IMA systems on multicores. As such, software functions implemented using the IMA architectural model will often be referred to as IMA software, functions or applications, and systems composed of IMA functions will be referred to as IMA systems. Similarly, software integration for IMA systems will often be referred to as IMA integration, module integration or simply integration.

Software Integration Process

The software integration phase of a real-time embedded system design consists in assigning the software platform onto the hardware platform of the system to be designed, and generating a static schedule to configure in advance the runtime behavior of the system.

The integration process of an IMA system involves: (i) system designers defining the general requirements of each function, (ii) application suppliers, in charge of designing one or several avionics function(s), and (iii) a module integrator, in charge of orchestrating the software/hardware integration of the system's module(s). The module integrator is in charge of ensuring the feasibility of the integrated architecture. To do so, he discusses with each application supplier to gather their partitions needs respectively, for instance in terms of CPU time, periodicity of execution, memory space, I/Os... Although application suppliers know which hardware platform their respective applications are going to be integrated onto, they do not know anything on the applications of the other application suppliers to be integrated onto the same module. As such, each supplier relies on discussions with the system designers and the feedback of the module integrator to make sure their respective applications needs are in line with the module capabilities. For instance, the integrator may ask a supplier for modifications, and the corresponding supplier must comply if possible, or discuss with the integrator in order to find an acceptable compromise.

Based on the information given to them by each supplier, the module integrator builds a static schedule of partitions activations on the corresponding module. The module integrator then verifies that it is possible to satisfy all time budgets and periodicity needs. If it is not the case, the module integrator discusses again with some or all the application suppliers to negotiate other CPU time budgets and/or partition periods until he is able to build a valid schedule satisfying all timing requirements. In case the negociation is not successful, the integrator may decide to integrate some applications to another module.

It is important to note that the module integrator does not know anything about the tasks inside each partition, so it is the job of application suppliers to perform timing analysis at task level on their respective applications. To do so once a schedule has been set for instance, the module integrator communicates the dates of the time windows of each partition to the corresponding application suppliers respectively, for them to perform some schedulability analysis at task-level.

In addition, application suppliers operate with no knowledge of the applications of other suppliers that will be allocated to the same module. As such, the timing analysis results are guaranteed to be independent from the other applications. Such an independence ensures the modularity of the platform/process, but also the possibility to follow an incremental certification process.

2.3 Safety and Certification Regulations

One major goal of aerospace systems is the safe transportation of passengers to their destination. As such, the safety of an aerospace system is ensured by guaranteeing the absence of catastrophic outcomes for the passengers and for the company. The safety ability of an aircraft to do so is guaranteed thanks to strong regulations guiding the design cycle of a system, and certification requirements that must be fulfilled according to a certification process in order to get the approval from certification authorities of the operationalization and industrialization of the corresponding aircraft. The certification authorities in Europe and the United States of America respectively are the European Aviation Safety Agency (EASA) [4] and the Federal Aviation Administration (FAA) [5]. Certification regulations published by the EASA are called Joint Aviation Regulations (JARs) [8]; regulations from the FAA are called Federal Aviation Regulations (FARs) [5]. Finally, another entity coordinating the development of standards and specifying Avionics Recommended Practices (ARP) for transport industries (automotive, aerospace and commercial vehicles) is SAE International (Society of Automotive Engineers) [12].

Certification Standards and Recommended Practices Documents

Certification regulation standards cover all parts of an aerospace system, from the selected hardware components to the final integrated system. Some of these regulations are complementary to one another, and are therefore interacting with each other. For instance, the main certification guidance documents impacting the software/hardware integration, verification and certification of IMA systems are the following:

- The DO-178 (*Software Considerations in Airborne Systems and Equipment Certification*) [10], which targets embedded software certification regarding the system's requirements, including timing considerations;

- The DO-297 (*Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*) [141], which provides insight on the activities to be conducted for software certification of IMA systems, and in particular, the analysis techniques that are considered relevant for certification authorities;

- The DO-254 (*Design Assurance Guidance for Airborne Electronic Hardware*) [140], which targets hardware processing units development for usage in avionics systems;

- The ARP4754 (*Guidelines For Development Of Civil Aircraft and Systems*) [11], which supports the DO-178 and DO-254 standards, and regulates the design and development of embedded systems for usage in avionics;

- The ARP4761 (*Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*) [139], which is used in conjunction to the ARP4754 in order to demonstrate compliance to airworthiness regulations for transport category aircraft.

The thesis target verification activities as requested in the DO-178 and DO-297. Figure 2.1 summarizes the relations between these documents. Finally, to obtain certification acceptance regarding a part or the entire system, one must abide by the corresponding regulations and provide all required proofs of correctness to certification authorities.

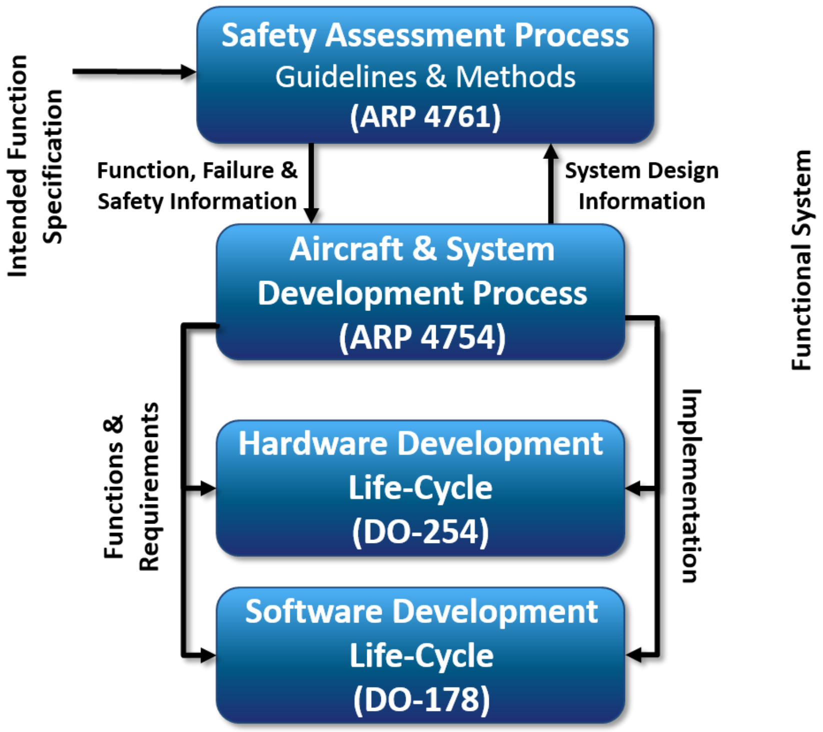

Intended Function Specification
Safety Assessment Process
Guidelines & Methods
(ARP 4761)
Function, Failure & Safety Information
System Design Information
Functional System
Aircraft & System Development Process
(ARP 4754)
Functions & Requirements
Implementation
Hardware Development Life-Cycle
(DO-254)
Software Development Life-Cycle
(DO-178)

Figure 2.1: Relations Between Avionics Certification Guidelines Documents [11]

Design Assurance Levels (DALs)

Software functions are classified into different levels of criticality, depending on the impact of a failure related to the function. Regarding software-level certification, the verification to be performed and the confidence level of the proofs to be brought to certification authorities depend on the criticality level of each function. Safety levels have been defined in order to formalize the certification process for each level of criticality, which are commonly called *Design Assurance Levels*. One DAL is assigned per function. DALs go from A (most critical level) to E (less critical level) according to the classification given in table 2.1: the severity of a failure is assessed according to its effect on passengers. Depending on that effect, different degrees of correctness verification are requested. For instance, DAL A applications are required to undergo a safe timing analysis at design time to safely bound execution times, whereas no such analysis is required out of DAL E applications.

The more severe the failure, the higher the DAL level, and the higher the effort that has to be made during the design, implementation and verification. In particular, the verification involves strong proof of correctness of the corresponding function's runtime behavior.

In IMA systems, robust time and space partitioning enables modules to be composed of functions with different DALs. The robust partitioning implemented in IMA architectures allows for each application integrated onto the same module to be qualified according to their respective DALs. Without IMA and robust time and space partitioning, all applications on a given module would have to be verified and certified according to the highest of their respective DALs. Finally, the complete module must be qualified according to the highest DAL of its applications.

DALs	A	B	C	D	E
Severity of a failure	Catastrophic	Hazardous	Major	Minor	No Effect
Effect on passengers	multiple fatalities	serious or fatal injuries to a small number of persons	physical distress, possibly including inuries	physical discomfort	none
Probability of occurence	$< 10^{-9}$	$< 10^{-7}$	$< 10^{-5}$	$< 1^{-3}$	–

Table 2.1: Design Assurance Levels (DALs) as Defined in [10, 11, 139, 140]

2.4 WCET Analysis

The validation of the runtime behavior of a given system includes a phase where execution times are evaluated. In particular, designers often try to upper-bound the Worst-Case Execution Time (WCET) of a task in advance, in order to make sure all implemented safety measures and runtime configuration remain applicable in the worst-case timing situation. WCET analysis techniques consist in identifying the worst possible situation leading to the maximum execution time for each task at runtime, then identified as WCET of each task respectively. If despite these WCETs, all deadlines of the system are still respected, then the system will always be able to respect its deadlines at runtime. Indeed, WCETs are maximum upper-bounds on the tasks actual execution times at runtime; as such, if a task WCET respects its deadline then any other execution time duration will respect the corresponding deadline, since it will remain lower than the WCET by definition.

Classification

Many WCET computation techniques exist, and can roughly be divided into three categories [169]:

- When performed while the code is running, the timing analysis is said to be measurement-based; each task is executed on the hardware or in a time-accurate simulator, while its execution time is measured. Various input data values are tested, to cover multiple scenarios that may lead to the maximum execution time of the task. However, such analyses cannot cover all possible scenarios and provides no certainty that the worst-case scenario has actually been observed [168].

- When performed on a model of the system, the timing analysis is said to be static; the hardware and software architectures, including tasks, are represented by several parameters in order to compute an upper bound of the corresponding WCETs. Such analysis does not actually execute the task on a hardware platform or in a simulation environment, but is rather based on an abstract representation of a task program. A classic example is static code analysis, as performed by the commercial tool aiT Analyzer [2] to perform WCET analysis at code level in single-core environments. Another example of static analysis metrics are response time analyses [137], where tasks respective entry points are abstracted by a higher level model of the tasks.

- When relying both on measurements at runtime and models analysis, the timing analysis is said to be hybrid. Concepts from both measurement-based and static-based approaches are combined. Examples of application of a hybrid timing analysis to compute WCET bounds is the commercial tool SymTA/S [74], or the MBPTA (Measurement Based Probabilistic Timing Analysis) as done in the PROXIMA project [51] for instance.

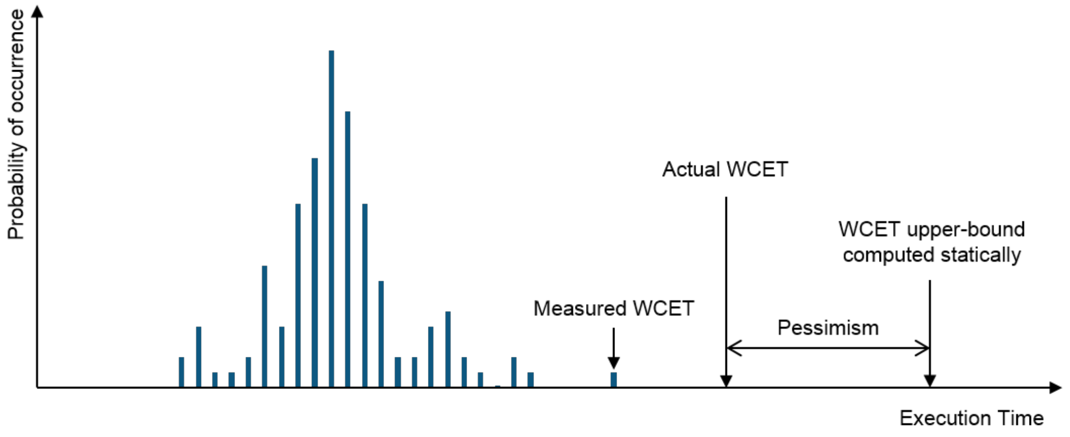


Figure 2.2: Execution Times Distribution for a given Task [169]

Figure 2.2 illustrates the distribution of execution times that can be observed for a given task at runtime, the corresponding real WCET, and the WCET bounds computed using a static timing analysis. Static analyses are safe in the sense that they produce upper-bounds on the execution time that are guaranteed to never be exceeded at runtime, since all possible input data values of the task are considered during analysis.

A static analysis covers all possible traces of execution that could happen at runtime, without actually needing the system to run in order to do so. These analyses are usually focused on identifying the worst-case runtime scenario, in order to produce a maximum upper-bound of the WCET of the given task, which will then correspond to a maximum upper-bound that is applicable to absolutely all situations that can happen at runtime for that task.

The static timing analysis is the only method that guarantees safe WCET estimates. Measurement-based and hybrid techniques are usually less complex than static techniques. However, the produced upper-bounds are not ensured to be safe since there is no proof that the actual worst-case runtime situation has effectively been observed and therefore that the corresponding actual WCET has indeed been measured. As such, as illustrated in figure 2.2, only static timing analysis techniques can be considered as producing safe WCET upper-bounds for certification authorities.

Moreover, static analysis techniques are considered formal proofs of correctness for certification authorities since they are focused on exhaustive analyses based on the worst-case situation. However, such analyses inevitably produce pessimistic timing upper-bounds, as the considered worst-case actually corresponds to a situation that is always worse than the actual worst-case situation that could happen at runtime. As illustrated in figure 2.2, the corresponding pessimism corresponds to the difference between the maximum WCET bound produced using static timing analysis and the actual WCET. On the other hand, pessimistic predictions are not prohibitive for the aerospace industry. In fact, the embedded pessimism can be seen as additional safety margins added to the computed WCET upper-bounds, provided that tasks are forced to hold the CPU until the end of the duration of the corresponding pessimistic bound computed by the analysis, even if their real execution finishes earlier. Safety margin is a concept commonly accepted in the aerospace industry, who would rather work with slow designs embedding too large safety margins but which runtime behavior is deterministic, rather than with highly optimized but non-deterministic systems.

Eventually, for a given static analysis technique, if the embedded pessimism is too important, the resulting schedules built using the corresponding WCET upper-bounds lead to CPU under-utilization at runtime. The consequence is the design of systems embedding much more hardware modules than could have been achieved if the WCET bounds were less pessimistic. As such, even though pessimism is tolerated, the WCET upper-bounds produced by a static analysis should be as close to the real WCET as possible. This is also referred to as tightness of the analysis.

Because aircraft certification is the main challenge of the present work, this thesis is focused on static timing analyses. As such, the rest of this chapter will be focused on presenting the main static timing analysis techniques of the literature that are commonly used in single-core environments.

Code Analysis

Static WCET analysis at code level derives its timing information from the analysis of the corresponding program code, while using abstract models of the hardware architecture. Unlike the measurement-based and hybrid timing analysis approaches, static code analysis performs an exhaustive analysis of all possible input values to single out the longest control flow path of the program leading to the maximum execution time. The control flow graph of a program is usually built on the assembly or machine level of the code, in order to represent low level instructions individually and give them a corresponding maximal execution time duration. Basic blocks of instructions are identified, as a maximal sequence of instructions that can be entered and exited only at the first and the last instruction respectively. To construct the corresponding control flow graph, the program is identified as a set of connections between these basic blocks, in which each end-to-end path corresponds to a possible execution trace at runtime.

The efficiency and reliability of WCETs produced by a static analysis highly depend on the abstract models used to represent the hardware platform. The more detailed the hardware model, the more accurate – and thus, the less pessimistic – the produced WCET upper-bounds. However, the price to pay for a detailed hardware model and accurate WCET bounds is unrealistic analysis times due to the complexity of programs to analyze and exponential numbers of state spaces to evaluate when exploring programs control flow graphs. In general, the level of abstraction of the system model influences the tightness and quality of the produced WCET bounds.

Scheduling Theory

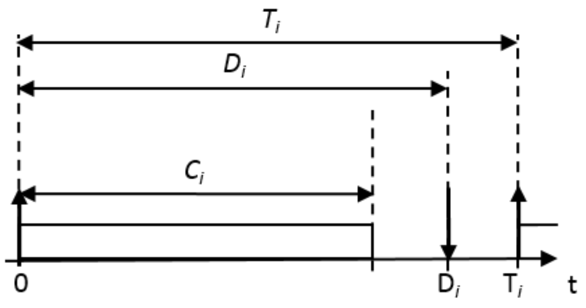


Figure 2.3: Classic Task Model [105]

Real-time software applications are often assimilated to a set of periodic tasks, which interact in real-time according to their environment. Tasks have temporal constraints, often represented as a certain amount of time within which they respectively must have completed their execution in order to ensure a correct runtime behavior of the resulting system. As such, Liu and Layland [105] proposed to represent a task τ_i as: an execution duration C_i; a relative deadline D_i corresponding to the amount of time it has from the moment it started running to complete its execution; and a period T_i corresponding to the time interval between two successive activations of the task. A simplified representation of such a task model is given in figure 2.3. In this model, C_i represents the maximum amount of time for a processing resource to execute the software code – also called ***entry point*** – of a task τ_i in isolation, i.e. when τ_i has exclusive access to all resources necessary to its execution. C_i is also referred to as ***execution duration in isolation*** in the literature as well as in this thesis. It is important to note that C_i is not the WCET of τ_i in a multicore environment, but rather a single-core environment. In this thesis, in a multicore environment, the WCET includes environment-related delays, such as intercore interference for instance.

From the point of view of a processing resource, a task τ_i represents a workload of $u_i = \frac{C_i}{T_i}$. The task τ_i requires to be scheduled every T_i time units, i.e. a task wakes up every T_i time units. When a task is selected by the OS to be scheduled on a given processing resource, the task is said to be ***activated***.

An execution of a task is often referred to as ***instance***. For a given task instance, the ***response time*** refers to the time duration elapsed from the moment the instance awakes, to the end of execution of the instance. The ***Worst-Case Response Time (WCRT)*** of a task τ_i is the biggest response time of all its instances; it is denoted R_i in the scheduling theory, as well as in this thesis. A task is then said to be ***schedulable*** if and only if all instances of that task are able to complete their execution within the deadline limitation, i.e. if $R_i \leqslant D_i$. A set of tasks is said to be schedulable if all tasks are schedulable.

In the scheduling theory, two types of analysis can be identified: feasibility analyses, and schedulability analyses. Both analyses view a task according to its runtime behavior in order to determine whether the task will always have enough processing resource to be executed as expected according to a behavior predefined during application design. These analyses belong to the scheduling theory, and abstract a task code by only representing the task according to runtime parameters such as priority level or periodicity for instance [105].

Feasibility Analysis A feasibility analysis focuses on determining whether a set of tasks, given a scheduling algorithm, can be scheduled on a given processing resource without any task missing some deadline at runtime.

In fact, feasibility analyses can be used in advance to help setting design choices such as deciding which tasks will be executed on which processing resources for instance; it identifies which combinations of task-to-processing resource allocation will lead to systems where some tasks cannot respect their deadlines for instance. Scheduling analyses are performed once a task-to-processing resource allocation choice has been made and once a schedule has been statically set for each processing resource.

Three main techniques exist to perform a feasibility analysis:

- *processor utilization analysis*, relying on the computation of a processing resource utilization or workload, $\sum_{i=1}^{n_T} \frac{C_i}{T_i}$ where n_T is the number of tasks executing on that processing resource. If the total workload of a task set on a given processing resource remains smaller than a certain value depending on the scheduling policy, the system is ensured to be schedulable.

- *processor demand analysis*, in which the cumulated sum of executions of task instances in a given time interval on a processing resource is computed. The resulting function depending on the considered time interval is called the *demand bound function*, and is plotted in order to verify that it always remains under the curve of the corresponding processor availability function.

- *response time analysis*, which determines the WCRT for each task R_i and compare them with the corresponding deadline D_i to check whether all deadlines can still be enforced at runtime despite the worst-case scenario.

While the first two techniques analyze busy periods of a processing resource, the third approach implicitly leads to the computation of an upper-bound of tasks WCRTs including blocking times due to potential runtime overheads and dependencies between tasks. One of the main contribution of this thesis is to propose a response time analysis fit for IMA architectures and multicore platforms. As such, the next section of this chapter will be focused on defining in greater details the response time analysis.

Scheduling analysis Compared to code-level WCET analyses, the primary goal of schedulability analyses is to assess beforehand whether a task will always be able to respect its timing requirements at runtime. In fact, part of validating real-time systems consists in proving that all tasks of the system respect their deadlines and other timing-related constraints such as communications and synchronizations for instance. Finally, an analysis performed in order to evaluate the possibility that all tasks will do so when using a given processing resource is called *feasibility analysis.* An analysis performed on a given schedule for a given task set and processing resources is called *schedulability analysis.* Both analysis metrics belong to the scheduling theory.

2.5 Response Time Analysis

This subsection presents the necessary background knowledge to understand the task-level timing analysis approach presented in this thesis. Every aspect of the analysis presented in this subsection presents models of the scheduling theory as currently commonly admitted among the real-time community; as such, it is important to keep in mind that none of the aspects presented in this subsection is intended for – and as such, applicable to – IMA architectures nor multicore processing resources.

Classic definitions of the response time analysis are given for software functions defined as a set of tasks only, with no representation or any equivalent of the partition level that exists in IMA architectures. As mentioned earlier, the adaptation of the techniques presented in this subsection to IMA architectures is in fact one of the contributions of this thesis, as will be presented in chapter 5.

In the rest of this section, the words ”processing resource or ”processor” are used to refer to an environment consisting in one single-core processor. Finally, the response time analysis will first be presented for preemptive task sets, and later on for non-preemptive task sets.

Classic Response Time Analysis

The response time analysis is a static analysis focused on determining whether the WCRT of a task τ_i, denoted R_i, is smaller than the corresponding task deadline, denoted D_i. The worst-case situation being considered to be the one leading to the longest response times, R_i is also often assimilated to an upper-bound on all possible values of τ_i response time that can occur at runtime. As a consequence, if R_i is smaller than D_i, then all other response times will be smaller than D_i as well. The response time analysis is therefore said to be a sufficient condition of schedulability for a given task set.

In order to understand how R_i is computed for each task τ_i, it is necessary to introduce the notion of busy period for a given processing resource.

In this thesis, all tasks are periodic. As such, two successive executions of a task occur according to its period. When scheduling a set of periodic tasks on a given processing resource, there exists a pattern of execution that is repeated infinitely at runtime. The smallest pattern repeated over time corresponds to a time interval called *hyperperiod.* In order to verify if a system respects all its deadlines at runtime, it suffices to perform a timing analysis on one hyperperiod in order to cover the entire runtime behavior of the corresponding system. As such, the hyperperiod is also sometimes referred to as *feasibility interval.* The length of the feasibility interval corresponds to the Least Common Multiple (LCM) of the tasks periods.

Within the hyperperiod, time intervals where the processing resource is busy executing one of the tasks are analyzed. More specifically, in order to consider the worst-case situation, the notion of *busy period* for a given processor has been introduced, as a time interval in which the processor in fully utilized [137]. In single-core processors, the longest busy period for a given processor occurs when all tasks are initially released at the same time date. This time date can be considered as the time origin for simplification purposes: all tasks are awake and ready to be scheduled, and must be scheduled so that they end their execution before their respective deadline.

The length of the busy period of a given processing resource depends on the considered task set. For a given time interval $[0, t[$, a periodic task τ_i is activated $\left\lceil \frac{t}{T_i} \right\rceil$ times, where T_i is its period. If the execution duration of τ_i is denoted C_i, then the workload $W(t)$ of the processor in a time interval $[0, t[$ is computed as follows:

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{2.1}$$

where n is the number of tasks scheduled on the processing resource.

W is a step function. The affine function $f(t) = t$ represents the maximum workload capacity of the processor. The first time instant t at which the equality $f(t) = W(t)$ is verified corresponds to the processor having completed the execution of all tasks awoken before t. More information, along with illustrative examples, can be found in [137].

Such a reasoning can be exploited in order to compute the response time R_i of a given task τ_i released at the time origin 0. Let t be the time instant corresponding to the end of execution of τ_i in the worst-case situation. Then the time interval $[0, t[$ represents the processor busy period limited to tasks having an equal or higher priority than τ_i. Let $hp(\tau_i)$ represent the set of such tasks. Then the cumulated workload corresponding to tasks of $hp(\tau_i)$, W_i, is computed as follows:

$$W_i(t) = C_i + \sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i)}}^{n} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{2.2}$$

Since τ_i was released at the time origin and t is the instant when τ_i completed its execution, the smallest solution of the equation $W_i(t) = t$ corresponds to the WCRT of τ_i, R_i. As a consequence, R_i is computed as a solution of the following equation [81]:

$$R_i = C_i + \sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i)}}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.3}$$

Equation (2.3) is usually solved iteratively: R_i is the smallest fixed point solution of the equation $W_i(R_i) = R_i$. Another way to solve equation (2.3) is to find the smallest integer value which satisfies equation (2.3). In this thesis, the second option is exploited in order to compute tasks WCRTs.

Response time analysis for non preemptive task sets

As implicitly accounted for in equation (2.3), the classic response time analysis covers task sets which are preemptive.

When non-preemptive task sets are considered, one must take into consideration the fact that a task becomes ready to be scheduled while a task form a lower priority level is already running. The task cannot start its execution before the end of the currently active task since preemptions are not possible.

In the worst-case situation, a task τ_i is blocked by the task of lower priority than itself that has the longest execution duration, which corresponds to the additional delay $\max\limits_{\substack{j \in [1;n] \setminus \{i\} \\ \tau_j \notin hp(\tau_i)}} C_j$

being suffered by τ_i at runtime. The corresponding equation to compute a task WCRT in a non-preemptive environment is then the following [137]:

$$R_i = C_i + \sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i)}}^{n} \left\lceil \frac{R_i}{T_j} \right\rceil C_j + \max_{\substack{j \in [1;n] \setminus \{i\} \\ \tau_j \notin hp(\tau_i)}} C_j \tag{2.4}$$

To simplify the explanations, the rest of this section presents derivation of the classic response time analysis for preemptive task sets only.

Context Switch Overheads

Every time a task is about to start running on a CPU, its context is loaded into the memory. This corresponds to operations performed by the OS between each task switch at runtime in order to save the context of the task that just finished executing, and to load the context of the next task to be scheduled. The time elapsed while doing so at runtime is called a *context switch overhead*, and is often denoted C_{SW} in the literature. To include context switch overheads in the computation of tasks WCRTs, the response time analysis equation is usually modified as follows:

$$R_i = (C_i + C_{SW}) + \sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + C_{SW}) \tag{2.5}$$

where C_{SW} is a maximum bound of the overhead occurring at runtime to switch from the task that just finished executing to the task starting its execution right afterwards.

Waiting Delays Consideration

Contrary to context switch delays which can be represented as constant over time, some additional overheads occurring at runtime are variable and cannot be represented as constants. These overheads may actually correspond to blocking delays suffered by tasks during their execution. These additional delays depend on the complete system's schedule and configuration. To include such delays in the classic response time analysis, additional terms are often added to w_i's equation, each new term corresponding to a specific overhead caused by a resource during a task execution. The resulting equation to compute the WCRT is then the following:

$$R_i = C_i + \sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{R_i}{T_j} \right\rceil \times C_j + \sum_{k=1}^{\#Resources} I_{i,k}(R_i) \tag{2.6}$$

where $\#Resources$ represents the number of resources causing waiting delays during τ_i's execution, and $I_k(R_i)$ is a maximum bound of the waiting delay caused to τ_i by the k^{th} shared resource.

Jitters

In theoretical models of the scheduling theory, every blocking action – access for granting/releasing a semaphore, setting/waiting for an event, etc. but also messages reading and writing – happening during a task execution is assumed to occur either at the beginning or at the end of execution of the task. For instance according to this model, a task accesses all the messages it consumes as soon as it wakes up, and produces all messages at the end of its execution. Analogously, if at some point in its entry point, a task needs some event or other synchronization resource to be available, then the test on whether the event or resource availability condition is verified or not is performed as soon as the task wakes up.

This is usually not true in practice, as tasks make calls to blocking actions in the middle of their executions. However, such a model enables to simplify the representation of dependence,

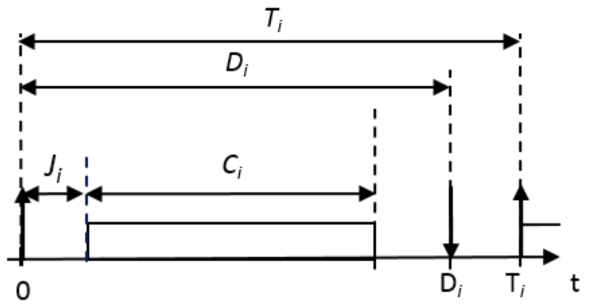


Figure 2.4: Tasks Defined with Jitters upon First Activation

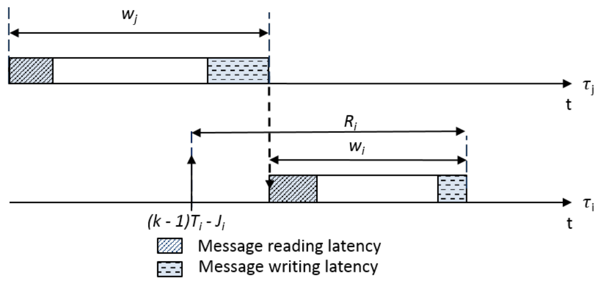


Figure 2.5: Classic Holistic model

along with the schedulability analysis. In this model, a task that just woke up can be scheduled only once all input messages, event or synchronization resource it exploits are available. To link such a model with the reality of industrial software, one makes the assumption that the software is broken down into tasks entry points so that each function call leading to a blocking action such as taking a semaphore, reading a message, verifying the occurrence of an event, etc. corresponds to the beginning of some task entry point, and the corresponding function call leading to releasing a semaphore, writing a message, setting an event, etc. corresponds to the end of the respective task entry point.

Finally, the delay between the moment the task woke up and the moment it can be activated by the OS is called ***release jitter***, or ***jitter upon activation***. A task jitter is denoted J_i in the literature as well as in this thesis.

As illustrated in figure 2.4, in presence of jitters the WCRT R_i of a task τ_i is decomposed into: (i) the jitter J_i suffered by τ_i, and (ii) the time elapsed between the moment when τ_i started its execution and finished its execution. In this thesis, we refer to the latter as the Worst-Case Execution Time (WCET) of τ_i and we denote it w_i. Equation (2.3) stands for independent tasks, i.e. tasks that are not involved in any precedence relation or consuming data produced by another task. In order to take into account eventual jitters upon first activation, equation (2.3) is modified as follows:

$$\begin{cases} R_i = J_i + w_i \\ w_i = C_i + \sum\limits_{\substack{j=1 \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j \end{cases} \tag{2.7}$$

Message passing

As illustrated in figure 2.5, and since no multi-threading is implemented regarding real-time avionics software, the time spent reading and writing messages are considered to be part of tasks execution, for they correspond to the CPU being busy handling the read/write requests of the corresponding task. This means if a task τ_i sends messages to other tasks, the time during which each message is written at runtime is considered to be a part of the task execution time. As a consequence, a task execution time now includes two additional durations:

- The time needed by τ_i to read all messages it receives, $msgsread_i$;
- The time needed by τ_i to write all messages it sends, $msgswrite_i$.

where $msgsread_i$ and $msgswrite_i$ compute the corresponding interference delay when τ_i respectively reads/writes its messages in the main memory. To our knowledge, no particular model of the $msgsread_i$ and $msgswrite_i$ functions respectively have been commonly admitted by the community as of today.

In scheduling theory, these two durations are added to the tasks response times according to the classic holistic model illustrated in figure 2.5: for all messages, the reading phase happens at the beginning of each task, and the writing at the end of each task. To match such a model, w_i's equation can be updated as follows:

$$\begin{aligned} w_i = \ & msgsread_i \\ & + \left(C_i + \sum_{\substack{j=1 \\ \tau_j \in part(i) \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j + \sum_{k=1}^{\#Resources} I_{i,k}(w_i) \right) \\ & + msgswrite_i \end{aligned} \tag{2.8}$$

where the term in parentheses corresponds to the right-hand side of w_i's equation; the first and the third term respectively refer to the duration of the reading and the writing phases described in the previous paragraphs.

2.6 Summary

In this chapter, we presented the background knowledge required to understand the extent of our contributions. The chapter began with a definition of the main general terms employed throughout this thesis, before presenting the basis of IMA and certification concerns in the aerospace systems design life cycle. A classification of existing timing analysis for WCET computation has then been given, with special attention to static techniques since it is at the core of the contributions of this thesis. The chapter then concludes with the current summary.

Chapter 3

State of the Art

The multicore challenge has been around for several years. Many works thoroughly studied the origin of the difficulty of performing safe WCET analysis when using complex multicore hardware architectures such as current COTS. As a consequence, many different approaches were taken in order to tackle the multicore challenges at hand. This chapter presents the related work of the literature dealing with the challenges brought by multicores in hard-real-time systems.

3.1 Overview

The multicore problem is vast and multifold. Resource sharing coupled with non deterministic architectures and arbitration policies result in difficulties to predict safe WCET upper bounds. Multiple roles and industries involved in hard real-time systems design are impacted, from the software developer to the manufacturer, the system designer to the provider of RTOS. As such, there exists more than one potential approach to tackle the multicore challenge.

This chapter is an attempt at presenting an overview of the literature related to the multicore challenge addressed in this thesis, i.e. allocating, scheduling and performing timing analysis on multicore platforms for hard real-time systems. Related works are roughly classified according to the approach chosen by the respective authors. Each identified category of approach is presented in a dedicated section. Categories remotely related to- but not in the scope of this thesis are briefly presented as well. A brief statement is given at the end of each section to compare with the scope of this thesis.

3.2 Execution Models

When dealing with multicores, some works of the literature propose to control the access to shared resources in a multicore platform by redesigning tasks software into several pieces of software explicitly separating shared resource access phases and non-access phases. The goal is to compute interference-free schedules by never overlapping shared resource access phases of two active tasks on different cores.

Schranzhofer et al. [151] studied different shared cache accessing models and compared them in terms of schedulability. They draw conclusions in favor of separating resource accesses from computation phases, and consequently propose to use a dedicated access model with superblocks executed sequentially.

Pellizzoni et al.[131] proposed the Predictable Execution Model (PREM) as an execution model where each task has a data acquisition phase during which data necessary to the task execution are fetched either from the memory or from the cache, followed by an execution phase.

Durrieu et al. [58] proposed a similar execution model then exploited to implement a predictable Flight Management System (FMS) on a multicore COTS. The execution model contains three phases: (i) a resource access phase at the beginning of a task execution in order to fetch all data necessary to its execution; (ii) a pure computation phase during which no access to any shared resource is performed; (iii) and another resource access phase at the end of execution of a task during which all produced and updated data is written back in memory. A schedule is built so that two resource access phases of two scheduled tasks on two different cores never overlap in time, in order to make sure that only one task is accessing a given shared resource at any given time. These works are later reused in [68] coupled with memory partitioning and static scheduling in order to propose a toolchain for a safe and interference free deployment of avionics applications on multicore systems.

Boniol et al. [32] proposed the same kind of execution models, but considered that multiple resource access phases – interleaved with pure computation phases – can happen throughout the execution of a task. The approach also includes cache consideration in order to improve the determinism of tasks runtime execution. To do so, resource access phases – also referred to as communication phases – are preceded with actions to flush the shared cache and load in cache the context of the next task to be scheduled, along with all data it may access in its first resource access phase.

Kim et al. [87] proposed a similar distinction between resource access and pure computation, but at partition level for IMA systems. In a given partition-level schedule, the time allocated to each partition is divided into exclusive resource access phases and pure computation phases. The identification of such phases is exploited when building a schedule, in order never to schedule two resource access phases in overlapping time intervals on two different cores, but allowing ”computation” phases to overlap.

Statement

New execution models are very promising for future multicore based systems, as they will enable to fully benefit from the performance gain that can be achieved with multicores. However, proposing new ways for software development exclusively targeting multicore architectures is out of the scope of this thesis, as our goal is to reuse legacy software. Moreover, the proposed execution models are not mature yet for industrial usage; one of the objectives of this thesis is to propose an approach to enable multicore usage without having to wait for some approach involved in the proposed works to be mature enough to be used in the avionics industry. For these reasons, new execution models are not further considered in this thesis.

3.3 Dedicated Designs

As mentioned earlier, many different approaches were taken in order to tackle the multicore challenges at hand. Among them, many works of the literature emerged as propositions of predictable multicore architectures, hardware components, monitoring and/or hypervising frameworks.

Multicore Processor Designs Some proposed custom multicore designs aimed at simplifying WCET analysis by proposing simpler, white box multicore architectures exploiting predictable arbitration policies and components.

For instance, Edwards and Lee [59] presented an architecture focused on providing repeatable timing behaviors through constant instruction execution times and programs path balancing. Later on, Zimmer et al. [181] designed a new PRET machine architecture called Flex-PRET for mixed criticality systems support. The proposed architecture relies on fine-grained multithreading and scratchpad memories in order to provide isolation capabilities for hard real-time tasks while offering quality of service for soft tasks.

Schoeberl [149] designed ”Patmos”, a java processor architecture for embedded real-time systems and provides the complete handbook and datasheet of the corresponding architecture in order to enable to perform timing-accurate analysis. The T-CREST European project (2007 - 2013) [150] then exploited Patmos in order to build a time-predictable multicore platform guaranteeing predictable and low WCETs for hard real-time systems.

The RECOMP (REduced Certification cOst for trusted Multicore Platforms) [133] project (2010 - 2013) targeted the design of a multicore architecture for enabling cost-efficient certification and re-certification of safety-critical systems.

Similarly, the MERASA (Multicore Execution of parallelized hard Real-time Applications suppoRting Analyzability) [164] project (2007 - 2010) aimed at designing a multicore architecture fitting all hard real-time industrial needs, for usage in avionics, automotive and railway industries. To do so, hardware components showing interesting features for hard real-time systems were identified first in order to be exploited in the resulting multicore architecture proposed.

The parMERASA (parallelized MERASA) [163] project (2011 - 2014) then started as a follow-up project on MERASA, and was focused on proposing a timing analyzable manycore architecture and enabling software parallelization for hard real-time industrial systems.

The PREDATOR (Design for Predictability and Efficiency) [170] project (2008 - 2011) proposed a generic methodology of conception fitting industrial requirements by identifying guidelines for the definition of the hardware platform to be used for hard real-time systems, according to the "architecture follows application" concept.

Hardware Component Designs Other similar works of the literature are focused on proposing custom component designs in order to control – or get rid of – interference when accessing shared resources in a multicore platform. The goal of these works is to inspire multicore manufacturers with designs potentially approved by hard real-time systems industries for their future multicore COTS. For instance, Obermaisser and Weber [124] introduced a system model with gateways and end-to-end channels for mixed criticality systems on network-based heterogeneous multicores. Kim et al. [88] presented an approach for WCET-aware dynamic code management using scratchpads for software-managed multicores.

Several works of the literature focused on proposing predictable memory controllers able to monitor the access to the main memory and enforce a predictable access scheme at runtime [15, 128, 77, 136]. Akesson et al. [15] proposed a predictable SDRAM memory controller named Predator. The proposed design is capable of enforcing a minimum bus bandwidth, along with a maximum latency for each memory access request generated by the cores of a same multicore thanks to a non-fair arbitration that requires knowledge about requestors. Paolieri et al. [128] proposed an analyzable memory controller that implements fair round robin arbitration. Hassan and Patel [77] presented a criticality-aware bus arbitrator for mixed-criticality systems. Reineke et al. [136] introduced the PRET DRAM controller, based on memory bank privatization in order to enforce predictability and temporal isolation at runtime.

Other propositions in the same line of idea focus on finding new arbitration policies for shared resources requests servicing. For instance several works focused on cache block eviction policies [98, 135, 92, 179], either to minimize tasks interference delays [135] or to prioritize high criticality tasks resource accesses over non-critical tasks in mixed criticality environments [98, 92, 179].

Interference Monitoring based on Middleware Layers Modifications Some propose approaches based on hypervising techniques in order to reduce or get rid of interference when accessing shared resources of a multicore processor.

For instance, Liu et al. [107] proposed to suppress memory interference by modifying the OS memory management subsystem; the approach consists in implementing a page coloring based bank level partition mechanism to allocate specific DRAM banks to specific cores or tasks. Similarly, 2014 Yun et al. [177] proposed a DRAM bank-aware memory allocator named PALLOC, aiming to achieve resource isolation on multicore platforms. PALLOC relies on cache and memory partitioning at application-level by allocating memory banks to applications by leveraging the page-based virtual memory system.

Rajkumar et al. [134] proposed a resource kernel for real time resource management for multimedia applications. Jean [80] implemented hypervising techniques for controlling resource accesses by cores in a multicore environment and enforce runtime determinism for future usage in avionics systems. Desenfants et al. [56] presented HIPPEROS, a multicore embedded real-time operating system kernel targeting safe WCET analysis and predictable runtime behavior. The European project MULTIPARTES [162] focused on resource isolation requirements that are mandatory in IMA architectures. Xtratum [180], the outcome of the project, exploits hypervising and virtual machines concepts in order to isolate clusters of cores and RTOS from each other so that each cluster can use its own areas of shared resources, the access to which is partitioned and monitored by a hypervising layer at runtime.

Van Kampenhout [165] proposed to create hardware partitions in order to allocate time windows for the usage of all shared resources to only one software application on the entire multicore at any given time. To do so, the author proposed to allocate the software partitions to hardware partitions, and then allocate them on cores.

Nowotsch [122, 123] pre-computed CPU time budgets to be allocated to each task and relies on monitoring techniques to enforce these bounds at runtime. A linear model for computing task interference and WCET bounds based on the number of resource access requests is proposed. Any task CPU time budget overrun is prevented at runtime by a hypervising layer that enforces the computed time budgets.

Finally, the approach in [111] proposed a single-core equivalence to implement legacy IMA applications on multicore, but does not offer interference-aware tasks response times computation. Instead, it relies on hardware-specific components to manage a server-based time sharing of applications.

Statement

The proposed architectures are targeting hard real-time systems and therefore usually present the main characteristics and safety components traditionally sought out by avionics industries in hardware platforms. However, although related, such works are not in the scope of this thesis, since the thesis aims at using COTS multicores.

The same statement can be made about the propositions of custom hardware components designs or requests arbitration policies. These works would require multicore and/or RTOS manufacturers to implement them in their designs in order for the corresponding works to be exploitable. For these reasons, dedicated designs are not further considered in this thesis.

3.4 Software-Based Resource Access Monitoring Approaches

Aside from dedicated designs, other works propose to control (monitor or limit) the resource accesses times or impose budgets for each resource and each application. To do so, such approaches rely either on specific scheduling policies, or on specific, custom hardware features. The difference with works presented in the previous section is that they rely on hardware or hypervising features that are either currently available in current COTS, or assume they will be in future COTS; they are focused on proposing a way to use them, and not a way to actually implement them, contrary to the previous section.

The goal of these works is not to characterize but rather get rid of multicore-induced interference, so that single-core analysis techniques can be exploited. The proposed works often consist in restricting access to shared resources, either physically (partitioned space allocation for instance), temporally (by controlling when a core can access a specific resource for instance), or both.

Several works proposing resource access monitoring techniques target shared cache usage in multicores [86, 20, 161, 42]. Kim et al. [86] proposed to partition the cache and main memory accordingly in order for the entire memory hirerarchy to be partitioned at task-level. To do so, the approach relies on OS-level cache management capabilities. An algorithm for finding the best task-to-core allocation regarding the minimization of core utilization is then proposed. The approach proposes a static model for intra-core cache interference, including warm-up delays and preemption-induced delays. The resulting delays are then included in a sufficient condition of schedulability according in a fixed priority preemptive context, either based on utilization bounds or response time analysis.

Altmeyer et al. [20] presented an algorithm to find the best task level cache partitioning configuration for which the existence of a feasible schedule is guaranteed. The proposed algorithm is based on the branch and bound heuristic, and takes a schedule as input. Tasks sensitivity to each cache partitioning configuration is taken into account depending on the size of the cache partition.

Tan and Mooney [161] claimed to be the first ones to link tasks priorities with cache lines usage. Each cache line is assigned a priority, and a task can use all cache lines with a priority equal or lower than the task priority. They propose a cache controller in order to dynamically monitor the priority and usage of each cache line.

Chisholm et al. [42] proposed criticality-aware optimization techniques for last level shared cache areas allocation to mixed criticality software systems allowing for schedulability improvement in multicore environments.

Suhendra and Mitra [159] proposed to combine cache partitioning and locking in order to cope with the problem of interference characterization in WCET analysis. The final goal of the approach is to achieve predictable L2 cache configuration and performance preservation. To do so, all levels of partitioning (none, core-level, task-level) have been evaluated for comparison purposes, both with and without cache locking (dynamic or static).

Resource monitoring techniques were also proposed for bus requests arbitration, in order to separate each core access to the shared interconnect and therefore to get rid of inter-task interference at runtime [138, 43, 36, 47, 48, 50].

Rosen et al. [138] presented an approach for bus access optimization for TDMA bus-based mutlicore platforms. A TDMA schedule is generated regulating the time intervals within which each core is allowed to generate requests to the bus; for a given TDMA schedule, the resulting additional delays caused by cache misses are evaluated and used as feedback to refine the TDMA schedule. Another implementation of TDMA-based bus requests scheduling is presented by Cilku et al. [43].

Similarly to [138], Burgio et al. [36] proposed an adaptive TDMA bus allocation and elastic scheduling, the elasticity characteristic being due to the fact that tasks periods are variable. The approach focuses on finding the best set of periodicities leading to a schedulable system on multicore.

Dasari et al. [47, 48, 50] proposed a response time analysis considering the contention of the shared memory bus based on simulating the distribution of tasks requests to the front-side bus. Observation points are included in tasks codes in order to check the number of cache misses that occurred up to these corresponding points, and the corresponding tasks WCETs are progressively increased depending on the requests emitted on other cores that are potentially interfering at runtime, according to a custom algorithm. Once WCETs stopped increasing, the algorithm performs a timing analysis to check the feasibility of the system.

Finally, another class of related works proposing resource monitoring tries to consider more than one multicore shared resource at a time. For instance, Yoon et al. [176] proposed to monitor both cache and bus access, along with computing a TDMA bus schedule, in order to optimize performance. Their approach determines the optimal allocation and scheduling of tasks on cores, bus and cache of a multicore platform; they then propose a WCET analysis based on the knowledge of the implemented TDMA bus schedule.

Statement

These works cope with the difficulty of modeling interference in multicore systems by controlling resource access, for example by identifying time windows during which each core is allowed access to the corresponding resource. Although such techniques are very interesting, their application to an avionics system depends on the existence of a firmware layer with such capabilities. COTS RTOS and multicores usually do not propose such monitoring capabilities, and any custom layer added to a design must undergo an additional certification acceptance process, which implies an extension of the time-to-market of the system to be designed. As such, the type of approaches presented in this section are not further investigated in this thesis.

3.5 Mixed Criticality Approaches

Some work of the literature came up with new task models distinguishing soft real-time tasks from hard real-time tasks, and building systems where hard real-time tasks are prioritized over soft real-time tasks so that (i) no critical task suffers any deadline miss, and (ii) soft tasks are scheduled as best as possible given the time left after safely scheduling critical tasks.

To the best of our knowledge, [166] was the first to propose to rename as ”mixed criticality systems”, systems where some applications were prioritized over others according to criticality levels. Since Vestal, the expression ”mixed criticality systems” is employed whenever tasks or applications of different criticality levels are integrated onto the same hardware platform, and a distinction is made between critical and non-critical applications when performing scheduling analysis and/or generation.

An overview of the research on the topic of mixed criticality systems has been written by Burns and Davis [37], covering the period since Vestal's paper up to – and including – july 2016. In particular relatively to multicores and allocation/scheduling concerns, the first paper addressing mixed criticality in multiprocessor environments was by Anderson et al. [21] according to [37]. Anderson et al. [21] proposed to use slack times not exploited at runtime by high criticality tasks for increasing quality of service of low criticality tasks. To do so, the authors proposed five levels of criticality matching the DALs of avionics systems, to which they assigned a separate scheduling policy. Applications with the highest criticality level (corresponding to DAL A applications) are scheduled statically, contrary to applications of all other criticality levels: DAL B applications are scheduled using preemptive EDF, DAL C and D global preemptive EDF and DAL E global best-effort. Each core is expected to have its own hierarchical scheduler implementation, and slack times are allowed to be passed along to other servers. Such an approach of slack reuse by other servers was then followed by several other works of the literature [21, 115, 72, 71].

Another line of works using the mixed criticality model propose to compute an offline schedule with arbitrary WCETs, and dynamically perform changes to that schedule depending on runtime outcomes [99, 67, 31, 91, 156]. Li et al. [99] proposed to exploit global scheduling policies and migration capabilities in order to schedule mixed criticality tasks on homogeneous multiprocessors.

Bletsas and Petters [31] addressed semi-partitioned scheduling, i.e. proposed a mix of partitioned and global scheduling within the same multicore processor.

Kritikakou et al. [91] relied on runtime monitoring to dynamically adapt a scheduling plan, thanks to regular comparisons of the remaining time available for each task before its deadline, and its theoretical remaining execution time required to finish, computed in isolation at observation points. High criticality tasks execution times are therefore monitored at runtime in order to identify when no further interference can be tolerated by the task; when it is the case, low criticality tasks running in parallel with the high criticality task are aborted until the latter completes its execution. Socci et al. [156] later extended the approach in [91] in order to take into account precedence constraints between task instances.

Some other works propose to build offline schedules depending on tasks respective criticality levels [67, 160, 70, 82, 79]. For instance, Giannopoulou et al. [67] exploited model checking techniques to generate a reliable schedule, and thus safe bounds on tasks worst-case interference.

Selicean and Pop [160] proposed an approach for the generation of an allocation and a schedule for applications on a multicore platform. Tasks are allocated to partitions first, which are then allocated to cores and scheduled. Tasks of different SILs (criticality levels for automotive systems) cannot be allocated to the same application, for reduction of certification costs purposes. For the same reason, partitions having different SILs cannot be allocated to the same core. System designers have the possibility to control part of the task-to-partition allocation, by defining mutual exclusion constraints between tasks of the same SIL, or by asking for some task to be allocated to a partition of a higher SIL. Finally, tabu search is used for the allocation, and list scheduling techniques are exploited to buid offline schedules.

Kelly et al. [82] proposed a SIL-dependent WCET analysis technique, while Goswami et al. [70] proposed an Integer Linear Programming (ILP) algorithm to generate schedules for CPU time usage by tasks but also bus accesses in order to optimize runtime performance.

Huang et al. [79] proposed to derive mutual exclusion constraints on tasks depending on their criticality, to be applied when scheduling the corresponding tasks.To do so, the authors propose to group tasks in classes according to mutual exclusion constraints, so that mutually exclusive tasks are never scheduled in overlapping time intervals in the same multicore.

Other mixed criticality related works proposed to generate more than one schedule offline and perform mode changes online depending on runtime performances [53, 54, 26]. For instance, Cortes et al. [45] proposed to compute several schedules offline, and activate one of them online depending on when tasks finish their executions at runtime. The selection of schedules is done so that no deadline of any hard real-time task is missed, and the minimum possible number of deadlines of soft-real time tasks are missed. The same concept is later implemented by Saraswat et al. [143] using a constant bandwidth server, the capacity of which is optimized in order to maximize the quality of service of the soft tasks.

De Niz et al. [53, 54] proposed an algorithm for optimal generation of multi-modal mixed-criticality schedules on multiprocessor platforms. The approach first exploits an algorithm for the allocation of tasks to cores based on a derivation of bin-packing problem, called "vector mixed-criticality packing algorithm" (vMCP). A second algorithm of schedule generation based on the rate monotonic policy is then used, called "Zero Slack Rate Monotonic" (ZSRM) and which includes mode commutation capabilities; however, no interference consideration is made in the proposed approach.

Baruah and Fohler [26] proposed to generate multiple schedules for mixed criticality systems; one schedule is computed as if all tasks were hard real-time tasks and corresponds to the schedule to be shown to certification authorities at design time for certification acceptance. All other schedules are less pessimistic and correspond to "engineering modes": at runtime, one of these secondary schedules is exploited, and as soon as a low criticality task overruns its WCET, a mode change is triggered in order to switch to the certification schedule.

Finally, another remarkable line of works proposed to search for the best priority levels assignment to tasks in mixed criticality systems so that the overall system is feasible on a given hardware platform [27, 148]. Baruah et al. [27] proposed a response time analysis for mixed criticality systems coupled with a smart selection of tasks priorities depending on their respective criticality levels.

Schneider et al. [148] proposed an offline priority assignment algorithm using an ILP formulation before generating schedules for mixed criticality applications on multicores, and is focused on QoC-oriented efficiency for cyber-physical systems.

Statement Mixed criticality techniques take advantage from breaking down tasks according to their criticality level in order to secure high criticality tasks execution over low criticality tasks, the mixed criticality concept did not convince aircraft industries and/or certification authorities yet. In fact, all works based on this concept would require convincing them first before being exploitable, which can take some time before being achieved. Such models would in fact require to modify current standards for software certification, and probably require additional precautions to be taken by application suppliers when developing avionics software. As a consequence, works on mixed criticality are out of the scope of this thesis.

3.6 Hierarchical Scheduling Considerations

The IMA scheduling problem can be treated as hierarchical scheduling problems. In such problems, the overall system is seen as several sub-systems consisting of different applications to be scheduled separately from each other at runtime. Hierarchical scheduling frameworks guarantee independent execution of each subsystem, and prevents a given subsystem from causing a failure in another subsystem via providing enough CPU resources to each subsystem [28]. Several server models currently exist to model the CPU resource allocation to subsystems in order to determine the CPU budget needs of each subsystem according to their properties (periodicity, scheduling policy, etc.) [95]. The CPU resource allocated to a given subsystem is exploited in order to schedule the applications constituting that subsystem. The resulting schedules can then be considered to have two levels: the top level corresponds to the CPU time allocation to subsystems, while the bottom level corresponds to scheduling tasks inside each subsystem, within the boundaries of the time windows allocated to their respective subsystems. As such, exploiting a hierarchical scheduler to arbitrate CPU resource usage in partitioned environments such as IMA systems is relevant.

Deng et al. [55] proposed a two-level hierarchical scheduling framework for open environments, each critical application corresponding to a separate server and all non-critical applications sharing the same server.

Kuo and Li [93] proposed an exact schedulability condition for such a two-level, rate monotonic scheduling framework for task sets in which all periods are harmonic. Lipari and Baruah [104, 101] later presented a framework for enforcing inter-application isolation for hard real-time systems under an EDF scheduling policy for servers. An exact schedulability condition is proposed, and each server is assumed to have knowledge of their task-level deadlines.

Mok et al. [114] introduced the notion of real-time resource model: the goal is to transfer task-level resource needs to the server level through the resource model, so that the two levels can exploit different scheduling policies while still being able to be analyzed separately and guarantee the schedulability of the task-level when performing server-level analysis. To analyze the resource model, the authors proposed a sufficient schedulability condition using a bounded-delay model [62, 113]. Shin and Lee [154] later extended the approach to cover rate monotonic and EDF scheduling schemes.

Aside from the bounded-delay model, a periodic resource model has been proposed, that targets hierarchical systems with periodic needs of resource allocation [153]. Several works then addressed the schedulability analysis of hierarchical systems exploiting the periodic resource server under various scheduling policies [142, 102, 153, 18, 52, 152]. In particular, Saewong et al. [142] introduced a worst-case response time analysis for rate monotonic scheduling; Almeida and Pedreiras [18] and Davis and Burns [52] presented the notion of worst-case response time of periodic resource servers; Shin and Lee [152] proposed schedulability conditions and utilization bounds for periodic tasks under EDF and RM scheduling over both the bounded-delay resource model and the periodic resource model.

Lipari and Bini [103] presented a hierarchical scheduling approach targeting schedule optimization by proposing an algorithm to find the minimum CPU budgets needed by each server for their task-level schedule needs. Schiendorfer [145] later proposed to use constraint programming in order to find optimized resource allocation and scheduling designs in hierarchical systems.

Chatha and Vemuri [39] proposed a framework for software/hardware co-design of hierarchical systems, named MAGELLAN, for multimedia applications. The framework uses an iterative approach for partitioning and scheduling working on tasks control flow graphs. To do so, the approach uses techniques such as clustering, and pipelining in order to optimize the execution of control loops in the final design solution.

Some works related to hierarchical scheduling target IMA architectures. Lee et al. [96] proposed a partition and channel-scheduling algorithm for partitions and messages scheduling in an IMA system. The approach uses a two-level hierarchical schedule to activate partitions (resp. message transfer channels) following a distance-constraints guaranteed cyclic schedule, and then dispatch tasks (resp. messages) according to a fixed priority schedule. To enhance schedulability and help optimize the system to be designed, the authors include heuristic algorithms for deadline decomposition and message channel combining.

Al Sheikh et al. [16] proposed a CP formulation for the allocation and scheduling of IMA applications onto multiprocessors, with strict end-to-end latency constraints due to partitions periodicity and communications. The approach takes into account memory capacity constraints, but also communications via AFDX networks: a tree of all possible virtual links routing and corresponding schedules is drawn before the tree is progressively reduced until a valid path is finally selected. The approach in [16] is focused on inter-modules communication delays, all modules being based on single-core processors. As such, no interference such as what can be experienced in multicores is occurring and therefore taken into account.

Kim et al. [87] proposed a CP for allocating and scheduling IMA partitions on homogeneous multicore processors. The task-level is not considered in the proposed approach, but the partitions characteristics are assumed to have been derived by designers according to their task-level requirements. As an optimization feature, the CP objective function is set on minimizing the number of cores to be used for a given IMA software platform. Yoon et al. [175] proposed to address the same problem of IMA scheduling on multicores by exploiting holistic techniques to achieve better design optimization.

Ouhammou et al. [126] propose a response time analysis for IMA applications scheduled using a hierarchical periodic server. The approach is based on supply bound and resource demand functions in order to propose a theorem for schedulability, to be integrated in the model-based scheduling analysis MoSaRT framework [125].

Finally, [23] proposes two multicriteria algorithms for the allocation and scheduling of IMA applications on distributed architectures. The approach focuses on ensuring end-to-end latency constraints depending on computing resources characteristics.

Statement Works focused on hierarchical scheduling techniques rely on the knowledge of the entire software platform when performing timing analysis and deriving the time budgets required per server. Relying on such a knowledge is however contrary to the spirit behind the IMA concept, promoting independence of applications development and verification within the same module.

In addition, although related works are said to be hierarchical, every proposed schedule generation process only considered the top-level schedule generation even when explicitly targeting IMA systems. Usually, the top-level schedule is built based on bottom-level timing requirements in order to determine the accurate size of time windows budgets to be reserved at the top-level. As such, a bottom-level analysis for each partition is required prior to computing a top-level schedule. However in the literature related to proposing a hierarchical scheduling analysis, either the bottom level – i.e. the task level in an IMA system – is an input to the proposed approach and assumed to be correct, or it is configured at the same time as the top-level – i.e. the partition level in an IMA system. In other words, in order to ensure that a feasible schedule at bottom-level exists, schedulability analyses are proposed at both top-level and bottom-level simultaneously.

In contrast, this thesis needs to check whether, for a given top-level schedule that is set as an input, there exists a bottom-level schedule in which all timing requirements of the corresponding subsystem are respected. And on the other hand, the bottom-level must have been characterized first prior to proposing top-level schedules. In fact, top and bottom levels must be handled separately. The top level is configured without knowledge of the bottom-level schedule, and the bottom level is constrained by the top-level schedule defined beforehand. As a consequence, approaches for hierarchical scheduling in the literature do not match IMA integration process in avionics industries.

Finally, to the best of our knowledge, the majority of the works of the literature about hierarchical scheduling targets single-core environments, or multiprocessor environments based on single-core chips. As such, these works do not consider interference delays due to sharing access to hardware resources at runtime. No multicore interference consideration is present in the works quoted in this section: they all target either distributed systems based on single-core processors, or simply ignore multicore interference. As such, although the works presented in this section could be inspiring for building CP formulations in this thesis for the allocation or the schedule generation topic, they are not comparable to the contributions of this thesis.

3.7 Multicore Scheduling Approaches

This section presents related works of the literature which include some interference considerations without relying on any modification of the software and/or hardware platform(s), nor on specific capabilities enabling to control the time instants when tasks on each core emit requests to a shared resource.

Multicore processors can be seen as a hardware platform with multiple processing units capable of executing several software simultaneously at runtime. As such, and as a hot topic of research among the real time community, many works in the literature emerged about scheduling applications on multi-processing platforms [33, 61, 38, 78, 174, 17, 16, 83, 60, 112]. For instance in [65, 33, 16, 127, 17] as well as some scheduling tools like Cheddar [155], SchedMCore [44], MAST [69, 24] or SymTA/S [74], schedule generation capabilities are presented for multicore environments. However, these works are either ignoring inter-core interference due to sharing hardware resources, or aimed at multiprocessor systems.

In a multiprocessor system, several applications are executed in parallel and they each have exclusive access of hardware resources of their own while running on their respective processors. In a multicore environment, several applications run in parallel and share access to the same hardware resources, which leads to waiting latencies at runtime which delay the end of execution of each task in the software platform. Such a difference between multiprocessor systems and multicore systems makes any scheduling approach designed for multiprocessor systems, not applicable to multicore systems. The resulting timing analysis is indeed performed without interference consideration, which leads to unsafe WCET upper-bounds, and therefore, unsafe designs.

To fill such a gap, many works later focused on the allocation and schedule generation problem while considering interference due to sharing access to one or several hardware resources in multicore environments.

Paoleri et al. [127] proposed to generate an allocation in multicore environments according to inter-task interference using CP techniques. Interference consideration is based on a selection of measurements of execution times at runtime in various allocation configurations, and using a non-preemptive EDF scheduling policy. Tasks are placed on the cores first, and the schedulability of the resulting system is verified. If successful, the solution is stored and a WCET sensitivity analysis is performed, corresponding to the measurement of tasks WCETs. These measured execution times are then compared to the WCET obtained in the last solution previously found; if the WCET of a task is bigger in the new solution, the task is allocated according to the last previously found solution instead. A schedulability analysis is performed to check again that the system is feasible at the end of the process, before repeating it until all tasks have been allocated and the system is feasible.

Bui et al. [35] defined a genetic algorithm for determining a task-level cache partitioning so that the worst-case total utilization ratio of the system is reduced as much as possible. To do so, various cache partition sizes are evaluated: the size of the entire task memory context is considered at first, and then reduced until the overall cache partitioning corresponds to a feasible system.

Finally, other works propose to regroup tasks according to a common characteristic and perform the allocation according to the so constituted groups [25, 22]. Anderson et al. [22] proposed to group tasks according to the intensity of cache usage. In particular, tasks that are intensively using the cache are grouped together into megatasks, and scheduled on the same set of cores. A scheduling algorithm is proposed in order to minimize parallel execution of such megatasks. Baruah and Brandenburg [25] targeted the allocation of tasks to processors or groups of processors according to their affinities, and provided the corresponding schedulability analysis. Processors defined as common affinities to several tasks are grouped in order to form a cluster on which the corresponding tasks are scheduled globally, a cluster consisting in maximum three cores each.

Statement

The works presented above are indeed in the scope of this thesis: they tackle the problem of allocating and scheduling existing software on multicore platforms. However, some of them do rely on some software modification, which is prohibited in this thesis. Moreover, no IMA consideration is being made, and tasks response times are not evaluated safely using an over-estimation of the worst-case situation. Finally, all software applications are considered altogether during the allocation and/or schedule generation and analysis, which leads to an overall approach that is not compatible with the current IMA development and verification process.

3.8 Non-Intrusive Static Timing Analysis Techniques

A last category of works related to timing analysis on multicore can be identified: static analysis techniques for WCET upper-bounding, without requiring software or hardware modifications, even though some of these works may be hardware architecture-specific.

Indeed in these approaches, the software and hardware platforms are untouched inputs of the integration of multi-core processors problem. In that sense, such works are referred to as *non-intrusive*. The goal of these approaches is to characterize the implicit resource sharing between the cores of a multicore architecture. This part of the literature is the one that is the most in the scope of this thesis of the entire state of the art around multicores.

Cache Interference

Several works of the literature are focused on cache-related delays in multicore environments.

For instance Lee et al. [94] proposed a schedulability analysis based on identifying the number of useful cache blocks encountered between in order to derive a preemption cost. Luniss et al. [110] proposed another approach for cache-related preemption delays consideration in hierarchical systems.

Other than cache-related preemption delays, several works propose cache analyses covering multicore interference in general [57, 94, 121, 158, 38, 110, 120].

Sasinowski and Strosnider [144] presented a dynamic programming algorithm for cache memory partitioning for real time systems. The proposed approach consists in a search algorithm for determining how to partition the caches and the main memory in order to increase performance at runtime. The search algorithm tries several configurations, and evaluates the system schedulability thanks to a test embedded in the search algorithm. The approach relies on a cache analysis in order to derive the maximum number of cache misses and a mathematical model of the cache in order to derive an upper-bound of cache-related interference delay.

Pellizzoni and Caccamo [132] proposed an approach for evaluating tasks WCETs according to their requests to the cache memory.

Guan et al. [73] proposed an approach for high-level cache aware scheduling analysis. Tasks are scheduled online on the first core that becomes idle; the timing analysis is based on the identification of tasks running in parallel on two different cores and which share the same cache space.

Nagar and Srikant [120] presented a CP formulation for determining points in a task program that would lead to worst-case interference from the perspective of a shared cache. Once identified, these points are exploited in a WCET analysis. This approach is based on the assumption that considering the worst-case interference scenario at these program points for a given tasks suffices to cover the task worst-case scenario, instead of having to consider the worst-case scenario as the one in which interference occurs at all points of the task execution at runtime.

As multicores usually embed more than one cache level, some works of the literature tried to consider multilevel cache hierarchies [63, 173, 76, 97].

Yan et al. [173] proposed a WCET analysis which considered both the L1 cache and shared instruction L2 caches. Memory access requests are analyzed and classified as always hit or miss, and the L1 and at the L2 level. The shared usage of the L2 is taken into account, in the sense that each request targeting a cache set shared by at elast two cores is considered to be a miss. Li et al. [100] later extended the work of Yan et al. by identifying tasks on different cores with potentially overlapping life times in order to reduce the pessimism of the approach.

Schliecker et al. [146] proposed an approach for computing inter-task interference due to shared resources based on a minimum distance between memory accesses. To do so, the maximum number of cache misses per core within a certain time interval is computed, and then exploited as inputs to the computation of inter-core cache interference but also tasks respective response times. While the approach in [146] targeted preemptive task sets, a similar approach for non-preemptive sets is proposed by Dasari et al. [49]. Both [146, 49] focused on online scheduling, trying to minimize tasks response times depending on the temporal occurrence of requests.

Hardy et al. [76, 97] proposed to identify where memory blocks are cached at each cache level, with the knowledge of whether they are going to be reused in the future and by which task.

Memory Interference

Hyoseung Kim [84] proposed an extension of the response time analysis to cover main memory interference delays. To do so, a detailed behavioral model of a DRAM memory is implemented in order to distinguish the different situations that can influence the order of satisfaction of a request issued by a task on a core to the main memory (last opened memory region, reordering effects, etc.).

The work in [84] has later been derived by Yun et al. [178] in order to consider memory-level parallelism, by acknowledging the fact that each core is able to generate more than one request at a time. Both approaches in [84] and [178] assumed a single channel DRAM controller, prioritizing reads over writes, and maintaining separate buffer queues for read and write requests.

Multi-Resources Interference

Chattopadhyay et al. [41] proposed a shared cache and bus analysis in order to transpose the effect of the cache analysis to the shared bus. The shared cache analysis is based on inter-task conflicts and the individual task life times. The authors later extended their work in [40] in order to further transpose the effect of the shared cache and bus analyses to other parts of the processors pipeline, such as the branch prediction.

Bradford et al. [34] proposed a framework for modeling and computing schedules for real-time systems using simulation techniques. Interconnect latencies are analyzed using network calculus, even though it may not correspond to the functional behavior of internal buses of a multicore chip. In order to take into account all other multicore interference, one has to provide its own behavioral model for each hardware resource.

Altmeyer et al. [19] proposed a framework for static multicore scheduling analysis with detailed interference consideration. The proposed analysis targets multicore architectures and includes inter-task interference due to resource sharing. It includes multiple resource models in order for the proposed approach to be generically applicable to all COTS architectures, and relies on code level analysis in order to compute tight WCET bounds.

Statement

The approaches presented in this subsection rely on mutual analysis of in-parallel scheduled tasks. However, for that purpose, information on all considered applications is required in order to perform timing analysis and derive tasks WCET upper-bounds. In the case of IMA applications, the analysis to be performed for one application partitions must be independent from other applications, as required by the incremental verification process promoted for IMA systems. As the software architecture considered in the related works mentioned in this section are not IMA, the incremental process is not respected, and modifications to one application may impact the analysis of all other applications.

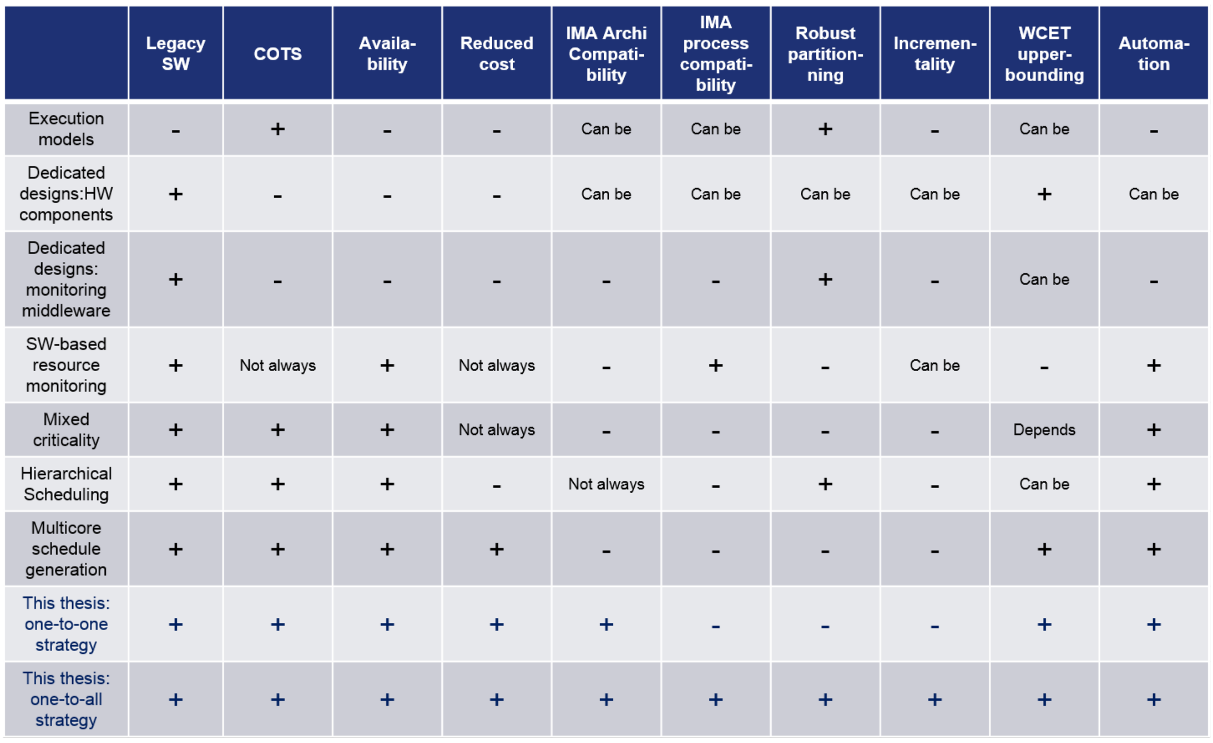
The work in this thesis falls in the category of this section; it relies on the approach in [84] as a basis of memory interference model. It has been modified in this thesis in order to fit IMA architectures, but also to refine the interference model and tighten the produced WCET upper-bounds. Finally, since this thesis is also about providing a methodology for certification-compliant allocation and schedule generation, an adequate IMA, interference-aware timing analysis is proposed to fit the resulting integration strategies. Further details will be given in chapter 5 page 123.

3.9 Summary and Conclusions

In this section, we reviewed the literature focused on proposing techniques to analyze resource sharing when integrating hard-real-time software onto a multicore platform.

As shown in this chapter, the literature is rich of works focused on one or several issues related to integrating hard-real-time software onto multicore COTS platforms. However, to our knowledge, in the literature, no consolidated approach to guide the transition of IMA applications from single- to multicore platforms without involving any software or hardware modification has been put forward so far. Some target the allocation problem, others target the scheduling problem. Some ignore all interference, and some try to take them into account, safely or not. To sum up, all works of the literature present at least one of the drawbacks that can be identified on the matter of integrating up to DAL A IMA software onto multicore COTS platforms. These drawbacks are listed hereafter.

IMA Hierarchical Level Considerations: The specific IMA architecture is mostly considered in scheduling analyses. However in these works, either only the top-level scheduling is considered, or top-level and bottom-level timing requirements are evaluated altogether in order



	Legacy SW	COTS	Availa-bility	Reduced cost	IMA Archi Compati-bility	IMA process compati-bility	Robust partition-ning	Incremen-tality	WCET upper-bounding	Automa-tion
Execution models	-	+	-	-	Can be	Can be	+	-	Can be	-
Dedicated designs:HW components	+	-	-	-	Can be	Can be	Can be	Can be	+	Can be
Dedicated designs: monitoring middleware	+	-	-	-	-	-	+	-	Can be	-
SW-based resource monitoring	+	Not always	+	Not always	-	+	-	Can be	-	+
Mixed criticality	+	+	+	Not always	-	-	-	-	Depends	+
Hierarchical Scheduling	+	+	+	-	Not always	-	+	-	Can be	+
Multicore schedule generation	+	+	+	+	-	-	-	-	+	+
This thesis: one-to-one strategy	+	+	+	+	+	-	-	-	+	+
This thesis: one-to-all strategy	+	+	+	+	+	+	+	+	+	+

Figure 3.1: Summary of the State of the Art

to derive a schedule fitting every application needs. Such an approach is not compliant with the key concepts of IMA, namely the independence of applications at runtime.

Rework Effort: The majority of works in the literature propose new multicore designs, new hardware components or features to hopefully be included in future multicore designs, new execution models, new software architecture designs, etc. The majority of these works are academic solutions to an industrial problem; as such, they rely on the hope that industries – processors manufacturers, avionics systems manufacturers, RTOS vendors, ... – will exploit their work in their respective future products. As they imply a significant amount of rework, they cannot correspond to a solution in the near future for immediate usage of multicore COTS in aircrafts.

Certification Compliance: In all presented works, certification considerations are either non existent or not sufficient for acceptance. For instance, the majority of the proposed timing analyses violate the incremental verification concept. This is also true for some approaches offering schedule generation techniques for hierarchical systems.

Even when some work is explicitly addressing IMA systems, no consideration of the incremental certification concept, but also the separation of concerns as implemented in the current engineering process – as described in chapter 2 subsection 2.2 page 61 – is made. For instance in the current integration process, the task-level timing properties are analyzed first, and for each application separately; then the resulting information is forwarded to the module integrator which computes a top-level schedule according to these information, and then forwards partial information to each supplier separately for feedback on the corresponding bottom-level schedule. Such a process has consequences on the way the timing analysis, but also the allocation and the schedule generation can be performed. Without consideration of such an engineering process, any approach for one of these three topics cannot be applied to IMA systems.

To address these drawbacks, the approaches presented in the following chapters are focused on covering every aspect of the integration of IMA software onto multicore COTS platforms.

The IMA architecture is central to all contributions proposed in this thesis. The rework effort is minimized by considering COTS only, not relying on any modification of the software architecture, the hardware platform or eventually the firmware layer either. Multicore interference are modeled using static models so that tasks WCETs can be safely upper-bounded. A schedulability analysis and a scheduling analysis rely on such interference models in order to compute such bounds and determine in advance if a given allocation or schedule respects all timing requirements of the system. The resulting analyses are compliant with the corresponding DO-178C verification objectives expressed for DAL A applications. Certification compliance is further ensured thanks to an adequate allocation and scheduling methodology proposed as the one-to-all integration strategy. Finally, the complete integration industrial process is covered, and in the case of the one-to-all integration strategy, enforces the current integration process followed by application suppliers and module integrators in current avionics systems.

Chapter 4

Integration Strategies Overview and System Model

This chapter presents all models used throughout this thesis. The software integration phase of a system design includes two major activities: (i) the allocation of the software partitions on the hardware platform; (ii) the generation of a static MAF schedule deciding in advance the reservation of time slots for partitions at runtime at each MAF cycle. Depending on which integration strategy and/or which activity is considered, some elements of the system model are represented differently. However, it is important to keep in mind that the goal of this chapter is to provide a complete presentation of all models and parameters used throughout this thesis depending on the considered strategy or process. Extended details on how each element is computed will then be given in chapters 5 and 6.

This chapter is organized as follows. First, the assumptions are introduced, before a general overview of the two integration strategies proposed in this thesis is given. The current chapter then proceeds with the description of the models of the software and hardware architectures considered in this thesis.

In the rest of this thesis, the software/hardware allocation activity will be often referred to as allocation, and the partition- and/or task-level schedule generation as scheduling. In addition, the methodologies proposed for the integration will be referred to as ”strategies”, and the allocation and schedule generation as ”processes” or ”activities” in the rest of this thesis.

4.1 Assumptions

In this section, we introduce all assumptions made in our work.

Core Allocation: we consider the case of static, fully partitioned scheduling on multicore, where IMA partitions and/or tasks are statically assigned to the cores of the same multicore processor. Extended details on how the allocation is done will be given in chapter 6.

Core Synchronization: we assume all cores of the same multicore platform to be synchronized at runtime, and have core clock frequencies that are even multiples of each other respectively.

This assumption is essential for the validity of the computed schedules, which rely on the knowledge of the other cores schedules (cf chapter 5 for detailed explanations). In fact, the WCETs and schedules generated using the approaches described in this thesis rely on this assumption, as they are computed while setting and/or assuming knowledge of runtime activity on other cores. If cores are not synchronized, it is more complicated to build a schedule and one must consider highly pessimistic scenarios in order to compute a safe WCET upper-bound for each task.

AMP Real-Time Operating System (RTOS) We assume the work presented in this thesis to exploit AMP RTOS, even for an SMP usage of multicore processors.

Such an assumption has been made due to the fact that, at the time this thesis work has been started, only AMP versions of ARINC 653 compliant RTOS where available. However, this does not limit the usage of this thesis to AMP RTOS only; it only means additional considerations for configuration have been taken into account in this thesis, which will be useless when using an SMP RTOS for instance. As such, this thesis is able to adapt to both AMP and SMP RTOS that will be made available in the electronic market in the future.

Scheduling policy: IMA partitions are scheduled statically, and are not preemptive. Inside each partition, tasks can either be all preemptive or all non-preemptive, according to the choice of system designers and function suppliers. Tasks are scheduled according to a fixed priority policy, using priority levels assigned to them at design time. The partitions are scheduled according to a schedule plan elaborated by the module integrator, without any priority or allocation consideration.

This assumption is based on current IMA architectures definition.

Monolithic Tasks Executions All tasks must complete their respective execution within the time window where the corresponding execution instance started running.

Such an assumption may be relaxed in current systems; however, such an assumption is made in this thesis in order to simplify the problem, and be able to focus on multicore-related challenges.

Timing Analysis during Schedule Generation: This thesis proposes an approach to automaticaly generate static, partition-level schedules for IMA systems based on multicores. However, as mentioned previously, tasks inside a partition can either be all preemptive or all non preemptive. It is important to note that the CP proposed for schedule generation then applies only for non-preemptive tasks. If one or several partitions implement preemptive tasks, the generation of schedule is replaced with the verification of existence of a valid schedule, as will be explained in details in chapter 5 section 5.1.2 page 126.

Harmonic Periods: we assume the periods of all partitions to be implemented on the same multicore module to be harmonic.

In fact in IMA systems, all partitions periods are harmonic on each separated module.

Timing Composability: tasks WCRTs and WCETs are computed as if the hardware architecture was anomaly-free.

Current COTS multicores are usually not anomaly-free. To cope with anomalies and allow our WCET model to be valid despite the anomaly-free assumption, we propose to enforce the computed WCET for each task at runtime by not allowing any task to hold the CPU for the exact amount of time represented by the computed WCET bound. The produced WCETs being safe upper-bounds except for the advent of an anomaly occurring at runtime, the situation where a task tries to overrun its WCET at runtime could be considered as the occurrence of an anomaly. As such, although timing anomalies are not faulty events, we assume them to be treated as such at runtime: the computed WCETs are assumed to be enforced through health monitoring mechanisms, and any task trying to overrun its WCET raises a faulty event. To do so, a specific recovery action to be undertaken must be defined in the context of health monitoring.

Module and Hardware Platform Definition: we assume the hardware platform to consist in only one module, embedding one multicore processor.

This assumption does not limit the use of our work, since it is complementary to works tackling the problem of inter-module communications in IMA systems on multicore platforms, such as in [23] for instance. Such works usually make the assumption that a schedule has already been safely derived, which corresponds to the work done in this thesis for instance. As such, our work can be exploited in addition to the works in [23] in order to add consideration of inter-application communications in multicore-based, distributed IMA systems.

Inter-Application Communications. As mentioned earlier, inter-application communications are not addressed in this thesis. According to the ARINC 653 standard [1], these communications must be handled using ARINC 653 ports following one of the two protocols – queuing or sampling – described in the standard. This implies additional memory considerations for the ports memory context storing, and additional traversal time latencies and freshness considerations regarding the transfer of the corresponding message from the producer to the consumer.

Such additional considerations represent significant work to be done regarding message scheduling, processor and interconnects speeds, etc. They are therefore usually tackled as a topic of their own. The complexity of such a problem explains why it has been identified as out of the scope of this thesis for simplification purposes: indeed, our work can be seen as the first step towards legacy IMA transfer to multicore COTS, our main focus being on the difficulty of computing safe WCET bounds in multicore environments.

On the other hand, state of the art works on inter-application communications usually assume that tasks and/or partitions allocation, schedulability and WCET analysis, schedule generation, etc. – i.e. all activities covers in this thesis – have already been performed beforehand. In our thesis, we make the assumption that inter-application communications will be handled once the activities tackled by our work have been addressed. As such, our work is complementary to – and therefore can be used in addition to – state of the art work related to inter-applications communications.

Shared Data and Message-Based Communications: intra-applications message-based communications are implemented using shared memory. All communications are implemented using dedicated shared memory spaces, the size of which is specified by the corresponding application supplier and is assumed to be large enough for one message. Each shared memory space is strictly limited to the storage of the content of the corresponding message.

Tasks involved in communications may have different period values and/or belong to two different partitions.

We consider the sending of a message by a task to another task to be completed with certainty at the end of execution of the sender task only. As such, the receiver task considers the message to be available only at the end of execution of the sender task.

Finally, partition-level communications are deduced from task-level communications. As such, if two tasks from different partitions exchange messages or share data, the two corresponding partitions are said to be involved in some inter-partition communications.

In the rest of this thesis, we use the term ***communications*** to refer to both message based communications or data sharing between tasks allocated to different cores.

It is important to note that communications are assumed to be the only valid justification to implement shared memory in the system. Management of message queuing or sampling is done as currently described by the ARINC653 standard [1].

Precedences Intra-application precedence relations can be defined, either between tasks or between partitions of the same application.

The IMA standard forbids inter-application precedences. As such, this assumption corresponds to an accurate feature of current IMA systems. However, if two partitions are involved in some data exchange, we assume the system designer may want to enforce an ordering of the partition windows occurence in the MAF schedule, as a preferred design choice rather than a precedence requirement. Although it is not an actual precedence relation specified by system designers, such design preferences fit the representation of precedence relation. As a consequence, they are represented as such in this thesis for the sake of simplicity.

Core-Level Main Memory Privatization In this thesis, ideal multicore based designs are the ones where the main memory is privatized at core level. By definition, each core is assigned memory address ranges corresponding to the address spaces containing the memory contexts of the partitions that have been allocated to these cores respectively. Although the IMA definition requires out of partitions to have disjunct memory address spaces, in order to share some memory areas. Two situations are then possible when two partitions are communicating: either they are both allocated to the same core, or they are not. If all communications fall into the first situation, then the main memory is indeed privatized at core level. If not, then at least two cores share access to some memory space, in which case the main memory is not privatized at core level. Strict main memory partitioning at core level can therefore truly be achieved only if no inter-partition communication is implemented within the considered module, or if all couples of partitions involved in inter-partition communications are allocated to the same core. As a consequence, if there exists inter-partition communications, one would then only have to carefully allocate shared memory areas to the main memory in order to achieve core-level memory privatization whenever possible with regards to the communications, partition context sizes and memory layout.

It is important to note that this reasoning is applicable to COTS multicores. As a consequence, the main memory is to be privatized at core level whenever possible with regards to the inter-partition communications and hardware core-to-memory path wiring.

Multicore Interference Multicore interference considered in this thesis are related to the usage of the main memory by all the cores of a multicore processor. The corresponding shared resources taken into account are the main memory space itself, but also the usage of main memory controllers and the interconnect linking cores to the memory controllers. As such, we abstract all devices of a multicore platform other than the cores, the main memory and the interconnect, and focus on the interactions of the cores of a multicore processor with the main memory.

Multicore Architectures Targeted in this Thesis We consider a hardware platform with only one bus-based multicore processor. The study proposed in this thesis handles activities and verification concerning the configuration of one multicore only. Therefore, every multicore present on the hardware platform of the system to be designed must be considered separately. As such, in the rest of this thesis, whenever we talk about partitions of a module, we actually talk about the partitions allocated to the multicore of the module on which the software platform has been allocated to.

Finally, we only consider one cache level in the hardware platform, corresponding to each core's L1 cache. As such, there is no shared cache considered in our contributions.

Scope of the CP formulation The efficiency and optimality of the algorithms used to solve the implemented CPs are not the concern of this thesis, and are therefore out of scope. In addition, any mention of optimization achieved in this work is expressed with regards to SWaP reduction and/or minimization of the objective function, which is therefore independent from eventual techniques and algorithms to optimize the CP search process.

4.2 Proposed Integration Strategies

4.2.1 Brief Overview

The main challenge when integrating IMA applications onto a hardware platform is the derivation of time slices for each application according to their respective CPU time needs. In order to derive accurate CPU time needs for each partition, one must first assess CPU time needs of the tasks of each partition respectively.

We propose two strategies for the integration of IMA applications onto multicore platforms, each strategy having its own advantages and drawbacks (see table 4.1):

- **The One-to-All Integration Strategy (cf. figure 4.1):** in this strategy, each supplier is responsible for statically allocating tasks of each partition to the cores of the multicore platform. Tasks of the same partition but on different cores are expected to be scheduled in parallel, so that in the final partition-level schedule, only one partition is scheduled on the entire multicore at any given time (cf figure 4.1). The outcome of the one-to-all integration strategy is a static SMP-like scheduling table, similar to the one illustrated in figure 4.1.

  In this strategy of integration, the steps addressing activities performed by each application supplier is assumed done by each of them independently of other application suppliers, on its own partitions and without knowledge of any partition of any other supplier that will be integrated onto the same module. This corresponds to the way the software integration is currently carried out in IMA systems.

  This strategy is in line with the robust partitioning and incremental certification acceptance requirements of avionics systems. It therefore currently represents a potential beginning of solution for legacy IMA software porting to multicore modules.

- **The One-to-One Integration Strategy (cf. figure 4.2):** in this strategy, the software/hardware allocation is performed at partition-level. All tasks belonging to the same partition are allocated to the core to which the corresponding partition has been allocated to. Then, during the schedule generation, the partition level is ignored as tasks are scheduled independently of the partition they belong to (cf. figure 4.2). The outcome of the one-to-one integration strategy is a static AMP scheduling table, similar to the one illustrated in figure 4.2.

  In this strategy, every step is performed by a single person, for instance either a system designer or a module integrator. However, it cannot be used for critical applications requiring strict robust time partitioning, as it does not prohibit inter-application interference at runtime, unless all partitions actually belong to the same critical application. This version is still proposed as a potential solution for future systems, for instance for integrating onto a multicore platform:

  - One multi-partition application alone on an entire multicore platform,
  - Applications for which strict time partitioning may not be required and for which system designer would accept the risk of having to reverify all applications if some modification is applied to the corresponding module,
  - Critical, non-IMA avionics software, i.e. in a multicore-based federated architecture.

Each strategy will be extendedly presented in chapter 6. However in the next two subsections, each strategy is presented briefly. The following subsection then compares both strategies in order to better identify their differences;

4.2.2 One-to-All Integration Strategy

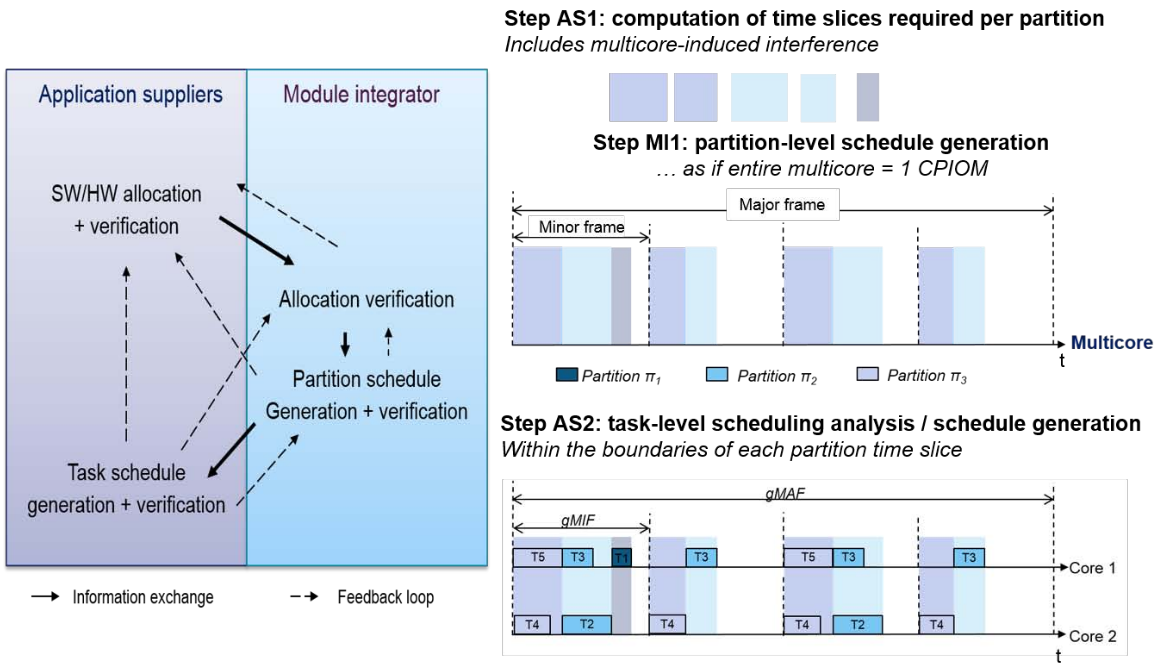

Step AS1: computation of time slices required per partition
Includes multicore-induced interference
Application suppliers
Module integrator
SW/HW allocation + verification
Allocation verification
Partition schedule Generation + verification
Task schedule generation + verification
Information exchange
Feedback loop
Step MI1: partition-level schedule generation
… as if entire multicore = 1 CPIOM
Major frame
Minor frame
Multicore
t
Partition π_1
Partition π_2
Partition π_3
Step AS2: task-level scheduling analysis / schedule generation
Within the boundaries of each partition time slice
gMAF
gMIF
T5
T3
T1
T4
T2
Core 1
Core 2

Figure 4.1: Overview of the One-to-All Integration Strategy

The one-to-all integration strategy consists in statically dispatching the content of each partition on the cores of a multicore platform so that: (i) each partition uses all cores of the multi-core and (ii) at runtime, only one partition is active on all the cores at a given time. Figure 4.6 gives an example of scheduling table resulting from the one-to-all integration strategy. The proposed allocation scheme ensures any inter-core interference to correspond to intra-partition interference, the resulting situation becoming equivalent to single-core designs provided that such an allocation can be enforced.

Such an allocation scheme leads to one partition being scheduled on more than one core at runtime. Unfortunately, there currently exists no IMA real-time OS able to handle such global scheduling yet. As an alternative, we propose to implement such an allocation by breaking down each partition into as many sub-partitions as they are cores available on the multi-core platform, and allocate each sub-partition to a different core. Each sub-partition will be seen by the OS as a separate partition, which enables the usage of AMP RTOS to implement an SMP-like allocation strategy.

Eventually, the allocation, verification and schedule generation activities are dispatched among the roles intervening in an IMA system design, according to the current integration procedure defining the separation of concerns among the different roles and regarding the activities to be held at design time.

As implied by figure 4.1, we divided the integration strategy into three steps according to the activities to be held and the roles implementing them. We briefly describe in the next paragraphs the content of these three steps; a graphic representation is also given in figure 6.1 page 150.

It is important to note that if some partitions are defined with preemptive task sets, the task-level schedule generated in the last step of the integration process only serves as a proof of existence of a valid task-level schedule, and will not correspond to the runtime task-level schedule. On the contrary, if some partitions are defined with non-preemptive task sets, then the generated task-level schedule may be kept by the corresponding application supplier as the task-level schedule to be enforced at runtime by the system. In the rest of this section, we describe in detail the three steps of the proposed strategy illustrated in figure 6.1.

SW/HW allocation The first step of the one-to-all integration strategy is about allocating IMA partitions to the cores of a multicore platform, and deriving the needs of each partition in terms of CPU time so that accurate time windows can be reserved to each partition in the schedule to be configured. The input of this step are the models of the software and the hardware platforms respectively, and the output is a task-to-core allocation along with an upper-bound on the CPU time budget that must be allocated to each partition at runtime.

As mentioned earlier, the software/hardware allocation is done by allocating tasks to cores for each partition, with the condition that each partition uses all cores. These steps are performed by each application supplier, on her own partitions and independently from all other partitions. This implies to implement sub-partitions as separate partitions at configuration time if legacy software is reused, for an AMP OS to be able to reproduce the selected task-to-core allocation. As the tasks of a same partition usually communicate through shared data in the partition memory context, inter-task has to be taken into account when searching for a task-to-core allocation. Two situations can occur: either (i) each partition can easily be divided into sub-partitions according to the defined communications, in which case simple module configuration at OS-level will suffice; or (ii) there exists no allocation combination where there is enough memory to define additional necessary shared areas for all communications involving two tasks allocated to different cores.

Aside from selecting a task-to-core allocation for each partition, each supplier is responsible with the verification of the enforcement of all associated timing requirements in the selected allocation. This includes estimating tasks WCETs in order to check whether the associated deadline can be enforced even in the worst-case situation. Combined with the allocation setting, WCETs are necessary to derive each partition CPU time budget as well. Finally, once a supplier has selected an allocation, she can derive the CPU time budget needed by each partition for the corresponding tasks execution. IMA partitions are often defined as periodic, and all partitions periods are assumed to be multiple of each other. This enables the runtime schedule to be broken down into frames, one frame corresponding to the minor time frame (MIF) and a complete pattern of the schedule corresponding to the major time frame (MAF) (see figure 44). As a consequence, for a given partition, an application supplier must provide one CPU time budget per frame for a complete MAF.

The first step of the integration strategy illustrated in figure 4.1 would have to be performed by each supplier separately. The inputs of this step are the software and the hardware models. The variables are the tasks to cores and memory allocation, but also the tasks WCETs and the partitions CPU time windows. The output of this step is a task-to-core allocation, proven to be correct. The correctness is ensured by embedding the interference-aware response time analysis proposed in this thesis for static WCET upper-bounding. Doing so expresses the requirement that all tasks are allocated exactly on one core, all cores are used by each partition, and every task would always be able to respect its timing requirements, including deadline, communications, precedence relations, etc. Eventually, tasks WCETs and core allocation are used in order to derive the corresponding partitions CPU time budget needs per cycle.

Global allocation verification and partition-level schedule generation The second step of the one-to-all integration process consists in verifying that the overall allocation and the corresponding time budgets per core is feasible in practice, and also the generation of a partition-level scheduling table. To do so, the module integrator relies on the information gathered from the output of the previous step by each supplier, and then proceeds with the verification of the global allocation. If the verification fails, i.e. if it is not possible to satisfy all time budgets in one MAF, the module integrator negotiates some changes with some or all application suppliers until a valid allocation can be found.

The module integrator then performs design space exploration in order to generate a valid partition-level schedule. If no valid schedule can be found for the partitions, the module integrator negotiates some changes with some or all application suppliers until a valid schedule can be found. The partition-level schedule being the output of this step, the module integrator provides, to each application supplier separately, the information of the schedule related to their respective partitions.

Task-level schedule generation and/or verification The last step of the one-to-all integration process is the verification of existence of a valid schedule at task-level, for each partition separately and given the partition-level schedule defined in the previous step. To do so, each supplier takes as input the time windows allocated to its partitions by the module integrator, and verifies if a valid schedule can be found for the tasks of the corresponding partition, within the boundaries of the time windows of its partition. By the end of this step, if all partitions schedules lead to the existence of valid task-level schedules, then the integration process ends successfully with a valid partition schedule for the corresponding module.

4.2.3 One-to-One Integration Strategy

In this section, we present the integration strategy we propose for integration IMA applications on multi-core COTS, without robust partitioning guarantees. This strategy is also named ”one-to-one integration strategy”.

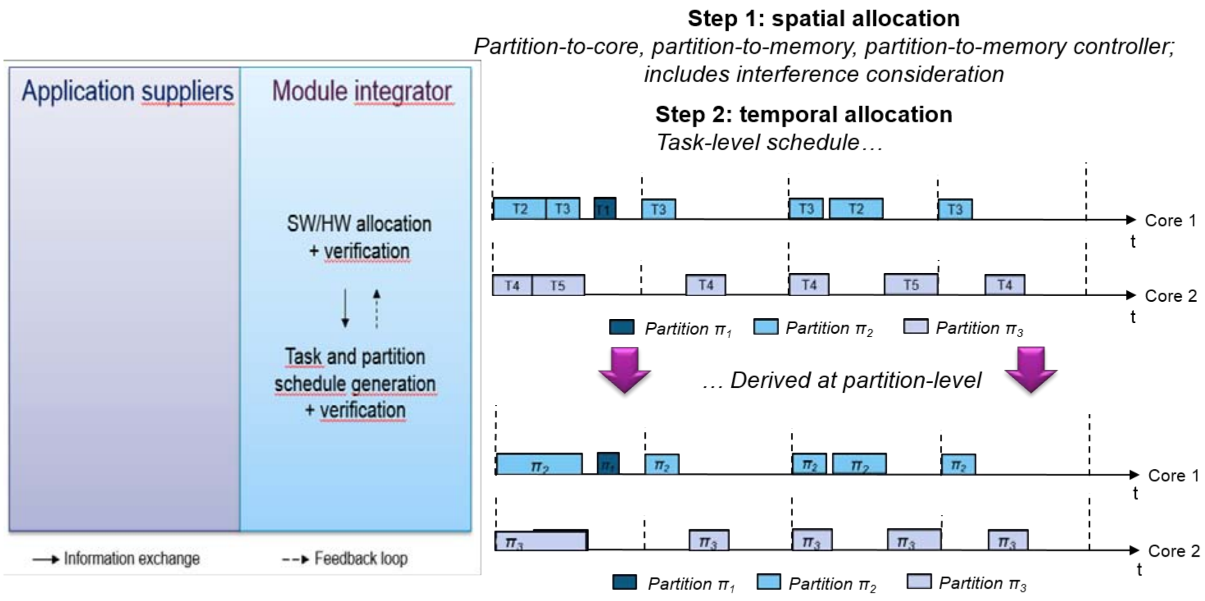

Step 1: spatial allocation
Partition-to-core, partition-to-memory, partition-to-memory controller; includes interference consideration
Step 2: temporal allocation
Task-level schedule…
Application suppliers
Module integrator
SW/HW allocation + verification
Task and partition schedule generation + verification
Information exchange
Feedback loop
T2
T3
T1
T4
T5
Core 1
Core 2
t
Partition π_1
Partition π_2
Partition π_3
… Derived at partition-level
π_1
π_2
π_3

Figure 4.2: Overview of the One-to-One Integration Strategy

As illustrated in figure 4.2, the one-to-one integration strategy consists in the allocation, schedule generation and WCET verification to be done in a linear process. The strategy can be divided into two steps: allocation, and schedule generation. Each step embeds its own verification activities.

As implied by figure 4.2, we divided the integration strategy into two steps according to the activities to be held. We briefly describe in the next paragraphs the content of these steps; a graphic representation is also given in figure 6.2 page 164.

SW/HW allocation The first step consists in allocating the software platform onto the hardware platform. The variables in this step represent the identification of the cores on which each partition is allocated to, along with the tasks WCETs. The output of this step is a valid allocation. To ensure the validity of the selected allocation, this step embraces the interference-aware response time analysis proposed in this thesis as a means for safe WCET upper-bounding. The response time analysis also allows to verify that there exists at least one schedule in which no deadline will be missed at runtime for the selected allocation.

Schedule generation Once an allocation is selected, one then has to configure the static schedule to be enforced at runtime. The input of this step is the selected allocation. The variables in this step are the activation offsets and the WCETs of the tasks of the SW platform, the two variables being combined in order to build the scheduling table to be enforced at runtime. The output of this step is an optimized static schedule that has been verified to be correct regarding timing requirements of the corresponding system. As in the one-to-all strategy, the correctness is ensured by embedding the interference-aware timing analysis proposed in this thesis, as a means for safe interference and WCET upper-bounding.

4.2.4 Comparison of the Two Strategies

Table 4.1 sums up the differences between the two strategies proposed as solutions for IMA software integration on multicore COTS. In the one-to-all strategy, the software/hardware allocation is done by allocating tasks to cores for each partition, whereas in the one-to-one strategy, the task-to-core allocation is inherited from the partition-to-core allocation: partitions are allocated to the cores, and all tasks of the same partition are allocated to the same core. In the rest of this thesis, whenever we talk about a task's core allocation in the case of the one-to-one integration strategy, we are referring to the core to which its partition has been allocated to.

Indeed, as illustrated in figure 4.2 in the one-to-one integration strategy, the allocation is performed as if each core corresponds to a separate computer on which to allocate software; each core then has its own set of periods, and the smallest repeated pattern at runtime may not correspond to the same period value for each core. In the one-to-all integration strategy, there is only one hyperperiod – called MAjor time Frame (or MAF) in IMA systems – for the entire multicore platform (cf. figure 4.1), and the schedule for an entire multicore platform is built similarly to the way a schedule is built for a single-core processor. On the contrary, in the one-to-one integration strategy, there is one MAF per core and the schedule of each core of a multicore platform is built as a schedule for a separate single-core processor.

	Static WCET Bounds	Legacy Reuse	Robust Parti- -tioning	Incre- -mental Certifi- -cation	Achieved Design Optimi- -zation	Highest Compatible DAL
One-to-All integration strategy	yes	yes	yes	yes	poor	A
One-to-One Integration Strategy	yes	yes	no	no	maximal	C

Table 4.1: Comparison of the two Proposed Integration Strategies

In the rest of this thesis, we regularly talk about MAF as a general concept when no distinction between the strategies is made, despite the fact that the MAF corresponds to a different definition in each strategy.

The various activities to be done during the integration are handled differently in both strategies. In the one-to-all integration strategy, each activity is assumed to be handled by the same role as in the current IMA integration process (module integrator, application supplier, system designer...) whereas the one-to-one integration strategy considers all activities are handled by the same person; this would imply one of the current profiles to handle activities that are usually out of their scope (for instance, having a module integrator performing a task-level timing analysis). Such a separation of concerns implemented in the one-to-all integration strategy forces some elements of the software model – along with the allocation and the scheduling activities – to be defined accordingly. For instance regarding the schedulability verification: in the one-to-one strategy, the role performing the verification can do so with the knowledge of the entire software platform, and therefore, rely on the knowledge of the value of the MAF. It is not the case in the one-to-all strategy, where the module integrator must produce a partition table and verify it, before each supplier can work on their own time windows allocated to the partitions they designed respectively and verify that their partitions needs are all respected. This implies the schedulability verification to be split into several steps, as will be explained in details in chapter 6.

As mentioned in the assumptions, we assume an AMP RTOS to be exploited when applying the contributions of this thesis. However, AMP RTOS forbid the allocation of a partition to more than one core, contrary to what is done in the one-to-all integration strategy. As a consequence, a workaround to be able to still implement the strategy using an AMP RTOS has been found.

In practice, in order to implement the one-to-all integration strategy using legacy IMA software with an AMP RTOS, partitions that can be scheduled simultaneously on different cores at runtime are partitions belonging to the same application only. As such when reusing legacy software, if an application is defined only as one partition, this means that only one core will be active on the entire multicore when that application is scheduled at runtime. An alternative to benefit as much as possible from software parallelization capabilities of multicore architectures is to modify single-partition applications in order to turn them into multi-partition applications. In the latter case, the corresponding suppliers would therefore have to redefine their application so that it consists in several tasks that can be executed in parallel at runtime.

This thesis contributions assist suppliers in such situations. For each partition to be split into several partitions, the proposed allocation problem formulation searches for the optimal allocation for the tasks to the cores while considering them to be scheduled simultaneously at runtime. The resulting task-to-core allocation then defines the new multi-partition definition of the application. Extended details about how partitions are split into several partitions such that their respective tasks can be scheduled on different cores in parallel as expected in the one-to-all strategy will be given in chapter 6.

The separation of concerns in the one-to-all integration strategy serves the enforcement of legacy certification requirements that are central to the IMA concept. More precisely, the separation of concerns allows the one-to-all integration strategy to enforce robust partitioning and be eligible to incremental certification acceptance. Indeed, WCETs are upper-bounded for each application independently from each other; one modification on one application has no impact on any other application which therefore do not have to be re-verified. These properties are specific to the one-to-all integration strategy, as they are not verified in the one-to-one integration strategy.

However, the one-to-all integration strategy may lead to designs that are less optimized than in the one-to-one integration strategy, where every activity is handled altogether and guide the selection of the final configuration. In addition, this strategy is easier to implement than the one-to-all integration strategy in the sense that it is more straightforward. It also may lead to more optimized system designs in terms of SWaP reduction achievement.

Finally, the fact that the one-to-all integration strategy manages to get rid on inter-application interference while following an incremental design and verification process makes it eligible for use with up to DAL A applications. The one-to-one integration strategy can be exploited for

DAL A applications only in the case of one multi-partition DAL A application being allocated alone on the entire multicore module. In all other cases, the one-to-one strategy can be used for up to DAL C applications only, or non-IMA software.

4.3 Software Architecture Model

Several models are defined in order to represent the same element of a system, depending on the activity to be performed. For instance, tasks are not represented the same way whether the allocation or the schedule generation problems are considered; they are not represented exactly the same way for both strategies either. This chapter presents all modeling elements exploited to represent the software platform at some point of this thesis; extensive details on which elements are used in which strategy/activity will be given in chapters dedicated to each integration strategy.

Let AS represent the set of application suppliers involved in the development of avionics functions considered for integration on the same module, and $as_i \in AS$ denote the identifier of the i^{th} application supplier. Let then $N_P^{as_i}$ (resp. $N_T^{as_i}$) denote the total number of partitions (resp. tasks of all partitions) the i^{th} supplier designed or is in charge of designing, and N_P (resp. N_T) the total number of partitions (resp. tasks) in the software platform considered.

Finally, for simplicity of the models representation, we assume one application to be assigned per supplier, and partitions π_i and tasks τ_j of the software platform to have unique identifiers.

Core Allocation

The elements of the model representing the software allocation to the cores depend on which integration strategy is addressed. Let N_C denote the total number of cores available on the considered multicore.

One-to-All Integration Strategy As illustrated in figure 4.6, in the one-to-all integration strategy, the core allocation is done at task level. We represent such a task-to-core allocation using a boolean matrix na defined as follows:

$$na_{ij} = \begin{cases} 1 & \text{if } \tau_j \text{ is allocated to core } i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

The size of na depends on the supplier exploiting our approach; it is $N_C \times N_T^{as_i}$ for the i^{th} supplier.

Some affinities between tasks and cores might be defined, as well as exclusion constraints. We model such requirements by defining two boolean matrices *taskCoreAff* and *taskCoreExcl* respectively as follows:

$$taskCoreAff_{ij} = \begin{cases} 1 & \text{if } \tau_j \text{ must be allocated to core } i, \\ 0 & \text{if no such requirement has been expressed.} \end{cases} \tag{4.2}$$

$$taskCoreExcl_{ij} = \begin{cases} 1 & \text{if } \tau_j \text{ must not be allocated to core } i, \\ 0 & \text{if no such requirement has been expressed.} \end{cases} \tag{4.3}$$

As for na, the size of *taskCoreAff* and *taskCoreExcl* is $N_C \times N_T^{as_i}$ for the i^{th} supplier.

One-to-One Integration Strategy As illustrated in figure 4.5, in the one-to-one-integration strategy, the core allocation is done at partition level. We represent the partition-to-core allocation using a boolean matrix a defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } \pi_j \text{ is allocated to core } i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The size of a is $N_C \times N_P$, where N_P is the total number of partitions designed by the corresponding application supplier.

Some affinities between partitions and cores might be defined, as well as exclusions requirements. We model such requirements by defining two boolean matrices *coreAff* and *coreExcl* respectively, as follows:

$$coreAff_{ij} = \begin{cases} 1 & \text{if } \pi_j \text{ must be allocated to core } i, \\ 0 & \text{if no such requirement has been expressed.} \end{cases} \tag{4.5}$$

$$coreExcl_{ij} = \begin{cases} 1 & \text{if } \pi_j \text{ must not be allocated to core } i, \\ 0 & \text{if no such requirement has been expressed.} \end{cases} \tag{4.6}$$

coreAff and *coreExcl* are of size $N_C \times N_P$.

Task Model

As mentioned earlier, depending on whether the allocation or the scheduling problem is considered, the parameters used for tasks representation are slightly different. The task model used for the allocation process is represented in figure 4.3, while the task model used for the schedule generation problem is represented in figure 4.4. We present here all elements of the task models.

Basic Definition

The set of tasks designed by a supplier $as_a \in AS$ is denoted $\mathcal{T}^{as_a}$, the union of all $\mathcal{T}^{as_a}$ sets corresponding to the set containing all tasks of the software platform, denoted $\mathcal{T}$. Tasks are denoted τ_i. We model a task τ_i as a vector $(C_i, T_i, D_i, H_i, prio_i, pid_i, tRam_i)$, where $C_i = (C_i^1 \dots C_i^{N_C})$ and $H_i = (H_i^1 \dots H_i^{N_C})$.

The elements of C_i give the execution duration in isolation of τ_i depending on which core it is running: C_i^k is the worst-case duration of execution of the entry point of τ_i on core k when considering that no other core is active during its execution. Similarly, H_i gives an upper-bound on the maximum number of memory access requests that τ_i can issue during its execution in isolation depending on which core it is assigned to. T_i is the period of τ_i and D_i is its deadline.

The parameter $prio_i$ is the priority level of τ_i, $prio_i < prio_j$ meaning that τ_i has a higher priority level than τ_j. We denote $hp(\tau_i)$ the set of tasks of higher priority than τ_i.

$$\forall \tau_i \in \mathcal{T}, hp(\tau_i) = \{\tau_j \in \mathcal{T} \mid prio_j < prio_i\} \tag{4.7}$$

The parameter $tRam_i$ is the memory footprint of τ_i, and pid_i is the index of the partition τ_i belongs to, and it is an input provided by each application supplier. We assume each D_i to be smaller than or equal to T_i, as each execution of τ_i must be finished before the next periodic activation. In IMA systems, D_i is usually equal to T_i.

Since "single-core duration in isolation" and "multicore duration in isolation" are equivalent, the C_i^k parameters can be deduced from single-core WCET analysis. Any analysis can be used; however for safety-critical systems as in the aerospace industry, execution times must be upper-bounded safely, in the sense that the timing analyses performed on a system remain valid even in the worst-case situation. For this reason, we exploit static code analysis techniques to extract the C_i^k parameters, thanks to tools like aiT Analyzer [2] for instance.

Similarly to C_i^k, a bound on each H_i^k can be extracted after static code analysis, for instance by using the capabilities of aiT Analyzer again.

Task Model for the Allocation Problem

Figure 4.3 illustrates the task model used when addressing the allocation problem. We denote the WCRT of τ_i as R_i. As illustrated in figure 4.3, R_i is broken down into (cf. figure):

- w_i, the time elapsed between the beginning and the end of execution of τ_i,
- J_i, the jitter that might appear upon the first release of τ_i.

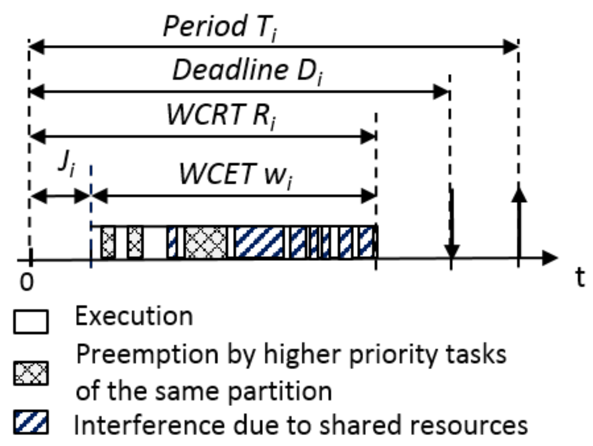

Period T_i
Deadline D_i
WCRT R_i
J_i
WCET w_i
0
t
Execution
Preemption by higher priority tasks of the same partition
Interference due to shared resources

Figure 4.3: Task Model for the Allocation Problem

By definition, as illustrated in figure 4.3, w_i includes all waiting delays suffered by τ_i due to preemption or resource sharing. Moreover, w_i being computed using a maximum WCRT upper-bound, we assume w_i can be considered to be a maximum upper-bound on the WCET of τ_i. As such, w_i will be referred as the WCET upper-bound of τ_i in the rest of this thesis.

Task Model for the Schedule Generation Problem

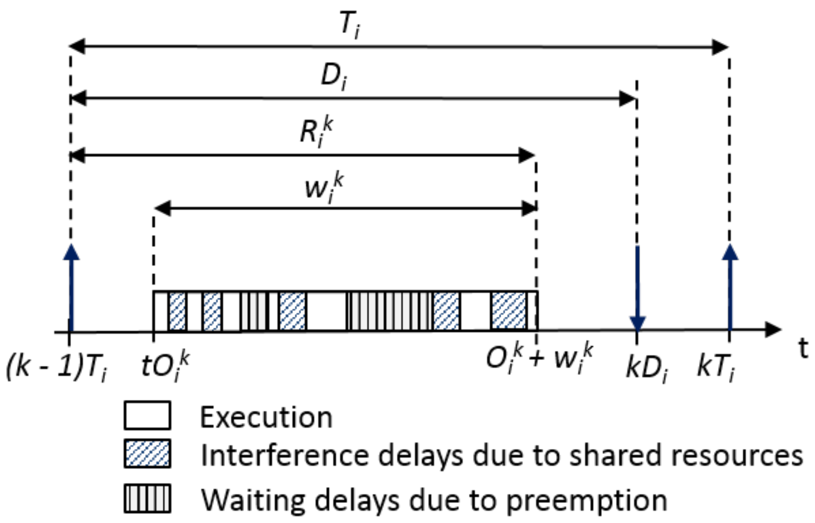


Figure 4.4: Task Model for the Scheduling Problem

Figure 4.4 illustrates the task model used when addressing the scheduling problem. For each task τ_i, we define an array $(tO_i^1 \dots tO_i^{nbActiv_i})$ of activation offsets of all instances of τ_i, where $nbActiv_i$ represents the total number of instances of τ_i in one MAF to be considered when building a schedule at task-level. The parameter $nbActiv_i$ is computed differently depending on the integration strategy considered, and must include the total number of instances of τ_i in one MAF so that all situations of inter-task interference potentially occurring at runtime are covered when computing WCET bounds. Details on how $nbActiv_i$ is computed in each strategy will be given later in this chapter.

Each activation date tO_i^k corresponds to the k^{th} activation of τ_i, and therefore respects the following property, in both integration strategies:

$$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i], \quad (k-1) \times T_i \leqslant tO_i^k \leqslant k \times T_i \tag{4.8}$$

For preemptive tasks, tO_i^k corresponds to the jitter upon first activation of the k^{th} instance of τ_i, scheduled only once in the MAF schedule; more details about such an equivalence will be provided later in this chapter.

An analogous reasoning is used to build an array of WCETs $(w_i^1 \,..\, w_i^{nbActiv_i})$, where w_i^k corresponds to an upper-bound of the WCET of the k^{th} instance of τ_i, as will be explained in details later as well.

Partition Model

As for the task model, the partition model varies depending on the integration strategy and the activity considered. For understanding purposes while introducing each parameter involved in the partitions models, the reader may want to refer to the examples of schedules computed as a result of each of the one-to-all and one-to-one integration strategies respectively, illustrated in figures 4.6 (page 104) and 4.5 (page 104).

Basic Definition

The set of partitions designed by the a^{th} supplier as_a is denoted $\mathcal{P}^{as_a}$, the union of all $\mathcal{P}^{as_a}$ sets corresponding to the set containing all partitions of the software platform, denoted $\mathcal{P}$. Partitions are denoted π_i. We define P_i as the period of π_i, i.e. the time interval between two successive time windows of π_i. The task-to-partition allocation is an input provided by each application supplier and is represented by the following boolean matrix $PART$:

$$PART_{ji} = \begin{cases} 1 & \text{if } \tau_i \text{ belongs to partition } \pi_j, \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

The size of $PART$ is $N_P \times N_T$ in the one-to-one integration strategy, $N_P^{as_a} \times N_T^{as_a}$ for each supplier as_a in the one-to-all integration strategy.

The parameter pid_i of each task τ_i giving the index of the partition it belongs to is defined as follows:

$$\forall \pi_i \in \mathcal{P},\;\; pid_i = \sum_{j=1}^{N_P} j \times PART_{ji} \tag{4.10}$$

To ease explanations in the rest of this thesis, we use the notation $\tau_i \in \pi_j$ to express the fact that a task τ_i belongs to partition π_j.

This thesis work may be used both for preemptive and non-preemptive task sets. As such, we define a boolean variable $isPreemptive_i$ configuring whether or not the task set inside partition π_i is preemptive, in order to be able to account for both cases in the contributions proposed in this thesis; $isPreemptive_i$ is defined as follows:

$$\forall \pi_i \in \mathcal{P},\; isPreemptive_i = \begin{cases} 1 & \text{if tasks inside } \pi_i \text{ are preemptive}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

Partition Cycle

It is important to note that all tasks are periodic, and all tasks of the same partition may not have the same periods. As such, not only is the schedule of activations of partitions periodic over time – the MAF being the smallest pattern repeated infinitely – but inside each partition is there a pattern of activation of its tasks that is repeated over time as well.

As mentioned earlier, such cycle is referred to as ***partition cycle*** throughout this thesis, and can have a different length depending on the partition, as illustrated in figure 4.7. The length of a partition cycle depends on the periods of activation of its tasks and corresponds to the smallest repeated pattern over time. All periods being harmonic, the duration of the smallest pattern of tasks activations inside one given partition π_i corresponds to the maximum period of its tasks $\max\limits_{pid_j=i}(T_j)$.

Partition Model for the Allocation Problem

We proceed define a CPU time budget E_i^k per partition π_i for each frame k of their respective partition cycle, as illustrated in figures 4.5 and 4.6 for both strategies. E_i^k corresponds to a maximum bound on the worst-case CPU time budget that must be reserved for π_i's tasks in the k^{th} frame of one MAF. Although the general definition of the E_i^k is valid in both strategies, the number of frames k to be considered depends on the strategy targeted.

In the one-to-one integration strategy, the number of frames to be considered in the allocation problem is $nFrames$ for all partitions, where $nFrames$ is computed using equation (4.28) page 108 as will be explained later in this chapter.

In the one-to-all integration strategy, the number of frames to be considered is $nFrames_i$ for each partition π_i; each $nFrames_i$ is computed using equation (4.23) page 107 as will be explained later in this chapter.

If the first CPU window of a given partition occurs earlier than the release of all its tasks, then no task is scheduled at the beginning of the window. This is not ideal in a schedule since this leads to slack times in each time window where the corresponding core is idle. Such a situation also jeopardizes the enforcement of task-level timing requirements.

In fact, partition windows are dimensioned so that the WCET of all its tasks supposed to be running in that window fit in it, and the expressed budgets do not take into consideration slack times or tasks jitters upon first activation. Windows usually include a safety margin; however, if tasks jitters have not been taken into account when dimensioning the partitions windows or when configuring the schedule, then if the slack time occurring at runtime is bigger than the safety margin introduced in the window, then a task may become unable to complete its execution before the end of the partition window or overrun its deadline.

As a consequence, it is very important to translate task-level jitters upon first activation information into partition-level jitters upon first time window information to make sure that, during the feasibility and schedulability analyses, the partition windows respect their tasks jitters and dependence constraints. To do so, we define a jitter for the first time window of each partition π_i respectively, denoted pJ_i. This jitter corresponds to the first window only, and is exploited during the allocation search and the related feasibility analysis.

On the other hand, when configuring a schedule at partition-level, one must make sure that partition windows are relevant for the corresponding task level. To be able to perform such a verification and define relevant partition windows, we define a jitter for each partition window, denoted as $latest_i^k$ for the k^{th} window of π_i. These jitters are exploited during the schedule generation phase and in the corresponding schedulability analysis. The $latest_i^k$ parameters enable to make the link between the task-level feasibility analysis and the partition-level feasibility analysis. This will be explained in greater details later in chapter 5.

Additional Parameters Specific to the One-to-All Integration Strategy In the one-to-all integration strategy, partitions memory footprints respectively are allocated to the main memory depending on their tasks core allocation. The memory context of tasks allocated to a given core will be stored in some main memory area addressable by that core. As will be presented in section 4.4, the memory storage area has been divided into sub-areas linked to each memory controller. Such subdivision usually corresponds to non unified memory architectures where each memory controller allows access only to a portion of the main memory. It can also be used in other, unified architectures but provided that the hardware platform offers safe memory partitioning capabilities, as has been identified in the last revision of the ARINC 653 standard [1] as IMA-friendly features for future multicore-based IMA systems. As such, our work is applicable to any multicore which architecture can be assimilated to the schema given in figure 4.9 page 112.

When integrating several partitions onto a multicore module, the module integrator needs to know the memory footprint per core of each partition in order to verify the realism of the memory allocation regarding the actually available memory capacity. To be able to perform such a verification, we define for each partition π_i a vector $(pRam_i^1 \ldots pRam_i^{N_{MC}})$ where $pRam_i^k$ denotes the size of the part of π_i's memory context supposed to be stored in some main memory area that is addressable by the k^{th} memory controller. By construction, $pRam_i^k$ corresponds to the sum of the memory contexts of the tasks τ_j of π_i that have been allocated to the k^{th} memory controller, $tRam_j$. Each $pRam_i^k$ can be seen as a memory budget per core and is computed according to the following relation:

$$\forall k \in [1; N_{MC}], \forall \pi_i \in \mathcal{P}, pRam_i^k = \sum_{j=1}^{N_T} PART_{ij} \times t2mc_{kj} \times tRam_j \tag{4.12}$$

where the boolean $t2mc_{kj}$ is equal to one if τ_j uses the memory controller j and zero if it is not the case, as will be explained later when defining matrix $t2mc$ in equation (4.36) page 114.

Partition Model for the Schedule Generation Problem

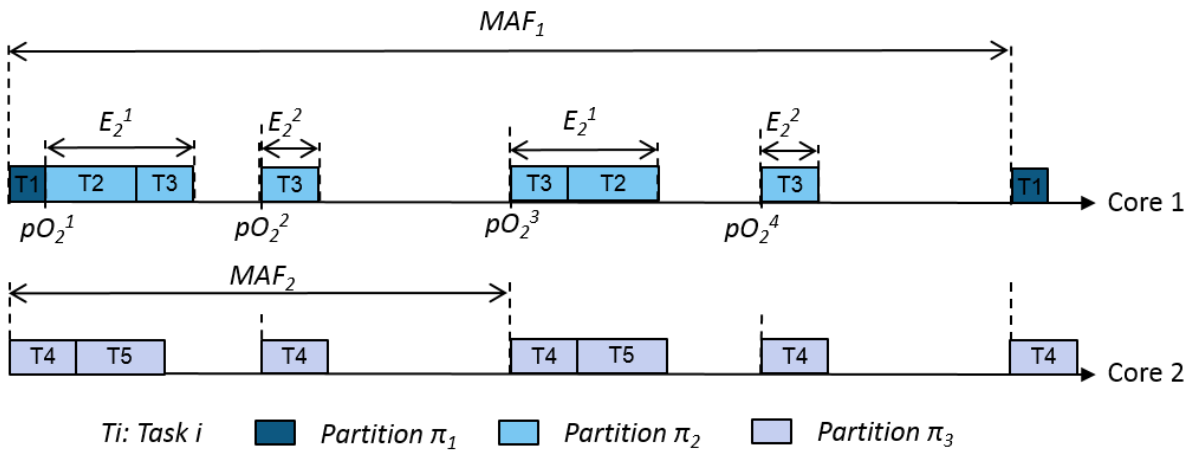


Figure 4.5: Example of MAF Schedule Resulting from the One-to-One Integration Strategy

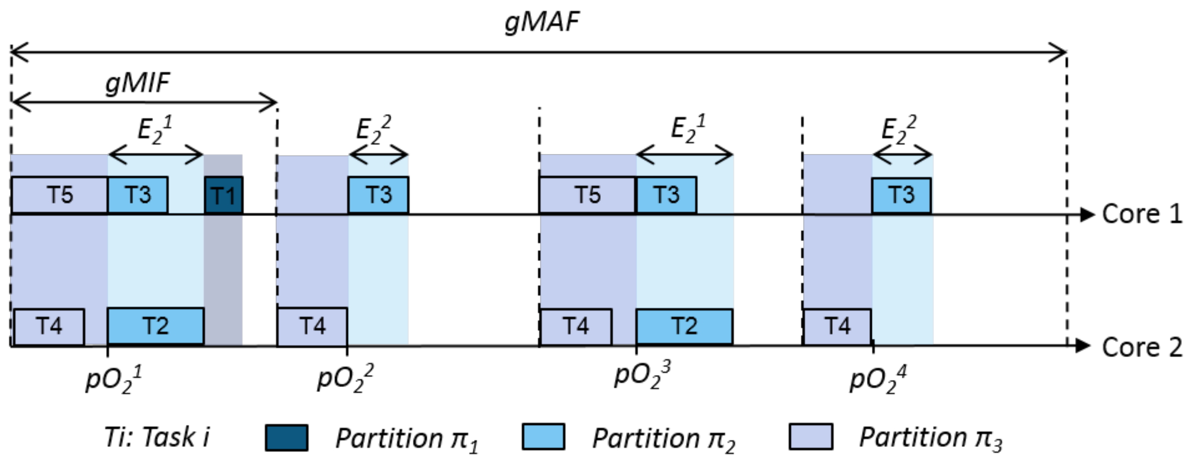


Figure 4.6: Example of MAF Schedule Resulting from the One-to-All Integration Strategy

To be able to compute a partition-level schedule, CPU time windows are reserved for each partition π_i according to their respective periodicity. A window – or partition window, or

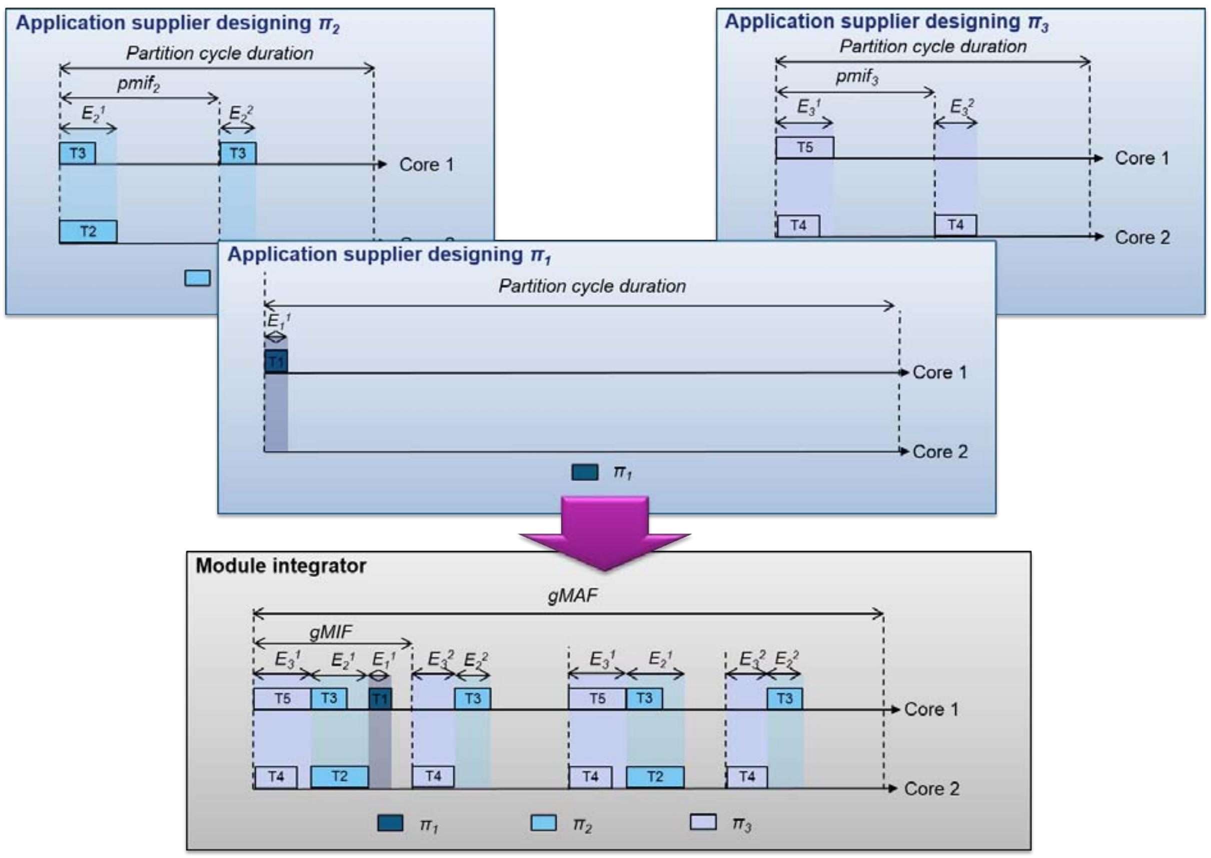


Figure 4.7: System Schedule Generation using each Partition CPU Time Budgets (One-to-All Integration Strategy)

partition time window – consists in a start date and a duration – or CPU time budget. The number of windows defined in the schedule for each partition depends on the period of the partition, but also on the duration of the MAF. The total number of windows for a given partition π_i in the MAF schedule is denoted $nbWindows_i$.

The MAF is divided into frames of equal duration, denoted MIF. The number of frames in the MAF is denoted $nFrames$. For a given integer $k \in [1; nFrames]$, the k^{th} window of π_i in the MAF schedule is defined by the start date pO_i^k, and the duration $E_i^{k'}$, (with $k' \in [1; nFrames_i]$) corresponding to one of the CPU time budgets of π_i in its partition schedule: the next paragraphs explain how to retrieve which CPU time budget $E_i^{k'}$ corresponds to the k^{th} window of π_i.

As partitions periods may be smaller than the MAF, their respective partition cycles may be repeated more than once per MAF. The number of repetitions depends on the partition period. As such, all partitions may not have the same partition cycle duration and thus may not be repeated the same number of times per MAF.

During the feasibility and schedulability analysis, the verification of timing-related requirements must be performed at least on a duration of the smallest pattern of activations of partitions at runtime. Such a duration may be longer than each partition cycle, and as such, the vector E_i^1, ... $E_i^{nFrames_i}$) does not cover the entire length of the analysis time interval.

An example is illustrated in the context of the one-to-all integration strategy in figure 4.7. The example, with three partitions, shows the difference between the number of windows in a given partition cycle and the total number of windows allocated to it in one MAF schedule. As illustrated, the same time budget E_i^k for a partition π_i may occur more than once per MAF. For instance, the partition cycle of π_2 is reproduced twice in the MAF. Indeed, each partition's period is either equal to the MAF or an even divider of the MAF by construction, and therefore each partition cycle may occur more than once in one MAF. As a consequence, for each frame k in π_i's cycle $[1; nFrames_i]$, the window corresponding to the budget E_i^k may also appear more than once, as illustrated in figure 4.7. In the case of π_2 in figure 4.7, the partition cycle of π_2 contains two frames, i.e. $nFrames_2$ is equal to two. The partition cycle of π_2 is reproduced twice in the MAF, leading to each time budget E_2^k occurring twice in the MAF schedule: the first and third windows of π_2 both have a length equal to E_2^1, the second and fourth windows have a length equal to E_2^2.

As such, depending on π_i's periodicity and the value of the module's MAF, the correct number of start dates must be derived in order to cover all time windows to be reserved for π_i in one MAF. To do so and therefore take into account the possible repetition of partitions budgets E_i^k per MAF, we define all π_i activation dates in one schedule in a vector $(pO_i^1, \ldots pO_i^{nFrames})$ where pO_i^k corresponds to the start date of the k^{th} window of π_i in the schedule, and $nFrames$ is computed using equation (4.24) page 107 in the one-to-all integration strategy, equation (4.28) page 108 in the one-to-one integration strategy. By construction and as illustrated in figure 4.7, the CPU time budget associated to pO_i^k is $E_i^{k'}$ where k' corresponds to k modulo $nFrames_i$ if $(k-1) \times gMIF$ is an even multiple of the partition period, and zero otherwise.

$$E_i^{k'} = \begin{cases} E_i^{k\%nFrames_i} & \text{if}(k-1) \times gMIF \equiv 0 \mod (P_i) \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

Analogously, it is possible to identify in which partition window a task instance will be scheduled in the MAF schedule. Let τ_j be a task belonging to partition π_i, and τ_j^k be an instance of τ_j with $k \in [1; nbActiv_j]$. Then τ_j^k is scheduled in the $\tilde{k}^{th}$ window of π_i if and only if the following relation is verified:

$$(\tilde{k} - 1) \times P_i = (k-1) \times T_j \tag{4.14}$$

If this relation is true, then k corresponds to the index of the instance of τ_j scheduled in the $\tilde{k}^{th}$ frame, and according to equation (4.13), $\tilde{k}\%nFrames_i$ corresponds to the index of the CPU time budget of the partition cycle of π_i to be used as the duration of the $\tilde{k}^{th}$ window of π_i.

MAF, MIF

The hyperperiod of a module is called Major time Frame (MAF) in IMA systems. It can be divided evenly in one or several time frames of equal size called Minor time Frames, or MIFs. The MAF, MIF and number of frames per MAF of a given module depends on the integration strategy considered and on each partition cycle.

One-to-All Integration Strategy As illustrated in figure 4.6 in the one-to-all integration strategy, there is only one MAF (resp. MIF) value for all cores. We denote *gMAF* and *gMIF* the MAF and MIF of the resulting schedule respectively. The MAF corresponds to the smallest pattern of activations repeated over time, and its duration is equal to the least common multiple of the partitions cycle duration:

$$gMAF = lcm_{\pi_i \in \mathcal{P}} (nFrames_i \times P_i) \tag{4.15}$$

where $nFrames_i \times P_i$ corresponds to the period of the partition cycle of π_i (cf figure 4.7): $nFrames_i \times P_i$ gives the duration of the cycle of π_i by multiplying the period of occurence of π_i's time budgets P_i by the number of frames inside π_i's cycle $nFrames_i$. For instance in the example illustrated in figure 4.7, $nFrames_2$ is equal to two for partition π_2, since its partition cycle consists in two frames corresponding to two different budgets E_2^1 and E_2^2 respectively.

Analogously to the MAF, the MIF corresponds to the greatest common divider of the partitions periods:

$$gMIF = gcd_{\pi_i \in \mathcal{P}} (nFrames_i \times P_i) \tag{4.16}$$

In the case of harmonic periods, the MAF and MIF respectively correspond to the maximum and minimum partition periods and can thus also be computed as follows:

$$gMAF = \max_{\pi_i \in \mathcal{P}} (nFrames_i \times P_i) \tag{4.17}$$

$$gMIF = \min_{\pi_i \in \mathcal{P}} (nFrames_i \times P_i) \tag{4.18}$$

One-to-One Integration Strategy As illustrated in figure 4.5 in the one-to-one integration strategy, each core may have a different MAF and MIF. Let MAF_p (resp. MIF_p) denote the MAF (resp. the MIF) of core p. By definition, MAF_p and MIF_p can be computed as follows for all core p:

$$\forall p \in [1; N_C], \quad MAF_p = lcm_{\pi_i \in \mathcal{P}} (a_{pi} \times nFrames_i \times P_i) \tag{4.19}$$

$$\forall p \in [1; N_C], \quad MIF_p = gcd_{\pi_i \in \mathcal{P}} (a_{pi} \times nFrames_i \times P_i) \tag{4.20}$$

Similarly to the one-to-all integration strategy, in case of harmonic periods, the MAF and MIF of each core can be computed as follows:

$$\forall p \in [1; N_C], \quad MAF_p = \max_{\pi_i \in \mathcal{P}} (a_{pi} \times nFrames_i \times P_i) \tag{4.21}$$

$$\forall p \in [1; N_C], \quad MIF_p = \min_{\pi_i \in \mathcal{P}} (a_{pi} \times nFrames_i \times P_i) \tag{4.22}$$

Number of Frames and Task Instances

This subsection describes how the parameters $nbActiv_i$ for each task τ_i, $nFrames_i$ for each partition π_i and $nbFrames$ are computed. We provide here brief explanations on how these parameters have a different impact in each strategy before providing the equations exploited for computing the corresponding parameters respectively. Extended details about how each parameter is used will be provided in chapter 6 when detailing each integration strategy proposed.

One-to-All Integration Strategy In the one-to-all integration strategy, the number of frames to be considered in one MAF depends on which activity (allocation or schedule generation) is performed and by which role (supplier or integrator). An illustration is given in figure 4.7.

For intra-partition analysis purposes, one partition cycle is analyzed. The number of frames $nbFrames_i$ per partition π_i identifies the number of windows E_i^k in a partition cycle.

For each partition π_i, the parameter $nFrames_i$ is computed as follows:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, nFrames_i = \frac{\max\limits_{pid_j = i} (T_j)}{\min\limits_{pid_j = i} (T_j)} \tag{4.23}$$

Periods being harmonic, $nFrames_i$ is always an integer for all partitions π_i.

On the other hand, for partition level schedule generation, the total number of frames $nFrames$ into which the MAF schedule can be divided is exploited, in order for the module integrator to be able to derive how many times each budget E_i^k is going to appear in the overall schedule. The total number of frames in one MAF schedule is computed as follows:

$$nFrames = \frac{gMAF}{gMIF} \tag{4.24}$$

Similarly to $nFrames_i$, $nFrames$ is always an integer.

Finally, the number of activations per task $nbActiv_i$ should cover all activations of τ_i in the MAF schedule, including in situations where a partition cycle is repeated more than once per MAF. It is therefore computed as follows:

$$\forall \tau_i \in \mathcal{T}, nbActiv_i = \frac{\frac{nbWindows_{pid_i}}{nFrames_{pid_i}} \times nFrames_{pid_i} \times P_{pid_i}}{T_i} \tag{4.25}$$

where $\frac{nbWindows_{pid_i}}{nFrames_{pid_i}}$ gives the number of repetitions of the partition cycle of π_{pid_i} in one MAF, and $nFrames_{pid_i} \times P_{pid_i}$ is the length of its partition cycle as explained earlier. Equation (4.25) can be simplified as follows:

$$\forall \tau_i \in \mathcal{T}, nbActiv_i = \frac{nbWindows_{pid_i} \times P_{pid_i}}{T_i} \tag{4.26}$$

Indeed, the MAF schedule consists in an even number of repetitions of partitions cycles. As such, by construction, the number of windows specified by the module integrator for a partition π_m is an even multiple of the number of frames in the partition cycle of π_m. As a result, $nbWindows_{pid_i} \times P_{pid_i}$ is equal to the total duration of all repetitions of π_{pid_i} partition cycle in one MAF. Dividing such a value by T_i for a given task τ_i provides the total number if instances of τ_i scheduled during that time interval, i.e. the total number of activations of τ_i in one MAF.

One-to-One Integration Strategy In the one-to-one strategy, there is one MAF per core, which are not necessarily equal. The time interval to be considered in the timing analysis must take this fact into account in order to be able to consider all potentially interfering task instances (cf. figure 4.4). More precisely, the analysis must be performed for all activations in the time interval $[0; \underset{p \in [1;N_C]}{\text{gcd}}(MAF_p)]$. For instance, let MAF_p and MAF_q be the MAFs on cores p and q respectively, with MAF_p bigger than MAF_q. Then to build a schedule for core p while accurately taking into account inter-task interference due to tasks on core q, more than one MAF_q must be represented in the timing analysis; as such, in the one-to-one integration strategy, the number of activation dates tO_i^k generated for each task τ_i is computed according to the least common multiple of the MAFs of the same module:

$$\forall \tau_i, \;\; nbActiv_i = \frac{lcm_{p \in [1;N_C]}(MAF_p)}{T_i} \tag{4.27}$$

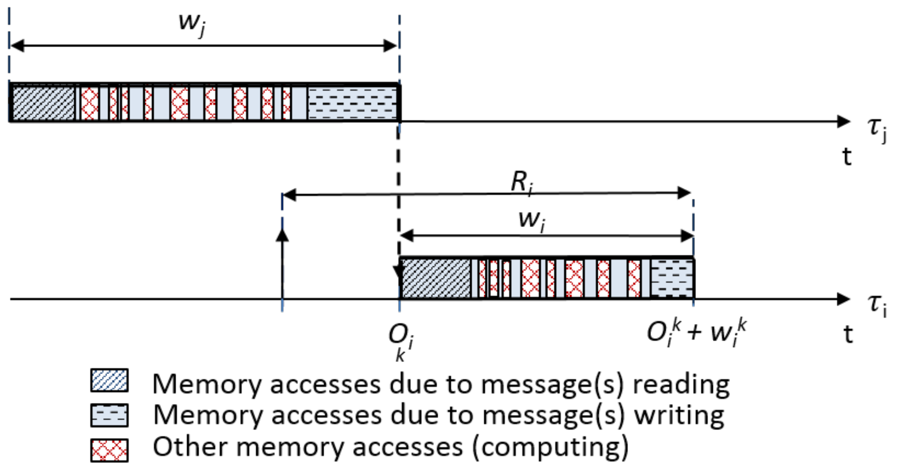
where MAF_p is computed using equation (4.19).

Regarding the number of frames to be considered when building a schedule, the same reasoning can be made. The number of frames must cover the entire analysis time interval, and divide the corresponding interval into frames of equal size, corresponding to the greatest common divider of partitions periods. The total number of frames to be considered is computed as follows:

$$nFrames = \frac{\underset{p \in [1;N_C]}{\text{gcd}}(MAF_p)}{\underset{p \in [1;N_C]}{\min}(MIF_p)} \tag{4.28}$$

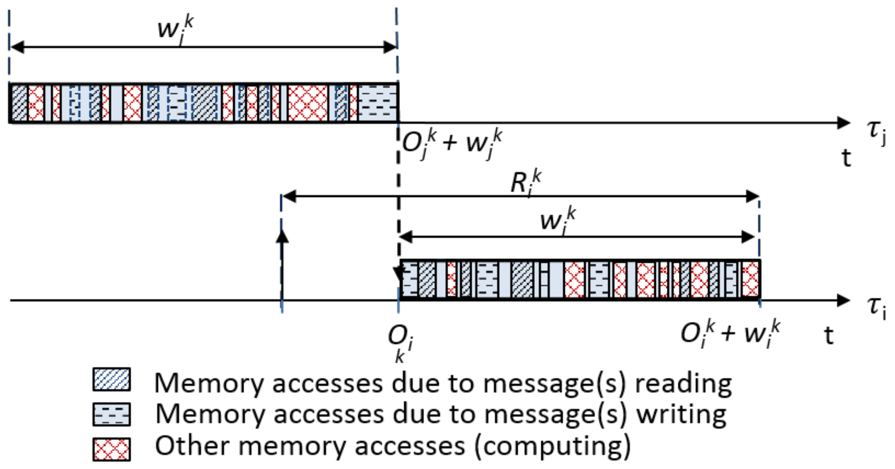
Communication and Dependence Model

As stated in the assumptions, the term "communications" refers to either shared data or message-based communications. Dependence relations consist in simple precedence relations without explicit data exchange. Communications and Dependences can be defined at two levels: task level and partition level. As such, they are represented separately according to these two levels as will be explained in the rest of this subsection.



w_j
τ_j
t
R_i
w_i
τ_i
O_k^i
$O_i^k + w_i^k$
t
Memory accesses due to message(s) reading
Memory accesses due to message(s) writing
Other memory accesses (computing)

(a)



w_j^k
τ_j
$O_j^k + w_j^k$
t
R_i^k
w_i^k
τ_i
O_k^i
$O_i^k + w_i^k$
t
Memory accesses due to message(s) reading
Memory accesses due to message(s) writing
Other memory accesses (computing)

(b)

Figure 4.8: Representing Messages as Memory Accesses in Tasks Response Times: (4.8a) when using the classic holistic model versus (4.8b) when using the model proposed in this thesis

Communications A message is seen as a maximum number of access requests to the main memory per execution, corresponding to writing (resp. reading) the message in memory by the task producing (resp. consuming) the message. The number of main memory accesses to consider is computed as part of the maximum number of main memory accesses H_i each task τ_i can perform in the worst-case situation, as presented earlier in the task model.

Figure 4.8b presents the model proposed in this thesis for communications-related latencies consideration Figure 4.8a showing the classic message model exploited in the literature. As displayed in the latter model, the latencies corresponding to a task reading and writing messages are respectively added at the beginning and the end of execution of the corresponding task. This corresponds to the assumption commonly adopted in the literature that all messages are read at the beginning of execution, and written at the end of execution. However, contrary to us, the classic model does not represent the actual message reading/writing in memory when evaluating the additional latency corresponding to reading/writing each message.

In our model, any message manipulation (reading and writing) amounts to reading/writing in main memory: to do so, messages are seen as a maximum number of memory reading/writing requests, corresponding to actually reading/writing the message in memory. Given such a representation, separating message reading/writing from the tasks execution as done in the classic holistic model is not accurate as soon as an attempt to bound the worst-case inter-task interference due to sharing the main memory is made. Indeed, such a separation is artificial, and in reality, a task may as well read and/or write a message in the middle of its execution, as illustrated in figure 4.8b. The corresponding memory access requests are then issued in the middle of the task execution, and therefore should be treated no differently than any other memory request generated by the task during its execution when evaluating how much memory interference that task can suffer.

In the model we propose, the maximum number of memory accesses corresponding to reading/writing all messages by each task τ_i respectively in the software platform are computed as part of the maximum main memory accesses τ_i can issue per execution, H_i. As such, H_i being exploited to compute the worst-case interference delay τ_i can suffer due to sharing access to the main memory, without making any difference whether the considered accesses are due to message reading/writing or pure computing, the resulting interference bound already contains a bound of the worst-case message reading/writing latency of τ_i per execution. The computed bounds are therefore produced independently of the origin of each memory access, exactly as illustrated in figure 4.8b.

As a consequence, the model we propose in this thesis for safely upper-bounding message-related latencies is more accurate than the classic one, and it successfully gets rid the inaccuracy of the classic holistic model when it comes to interference consideration.

We represent communications at task-level by a boolean matrix *msg* defined as follows:

$$msg_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and } \tau_i \text{ sends messages to } \tau_j, \\ 0 & \text{otherwise} \end{cases} \tag{4.29}$$

In the one-to-one integration strategy, *msg* covers all tasks of the software platform and is of size $N_T \times N_T$. In the one-to-all integration strategy, one matrix *msg* is defined per application supplier as_a: each corresponding matrix covers tasks of that same supplier's applications only, and its size therefore is $N_T^{as_a} \times N_T^{as_a}$. In addition, msg_{ii} is always null, as a task is assumed not to send (resp. receive) any message to (resp. from) itself.

Inter-application communications are represented by a boolean matrix *ipc*, defined as follows:

$$ipc_{ij} \quad = \quad \begin{cases} 1 & \text{if } i \neq j \text{ and some task belonging to } \pi_i \text{ produces data consumed} \\ & \text{by some task belonging to } \pi_j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.30}$$

The size of *ipc* is $N_P \times N_P$ in both integration strategies.

Finally, it is important to note that *msg* and *ipc* are unidirectional so that the producer and consumer of each message can easily be identified respectively.

Precedence Relations Tasks may have dependences without involving explicit data exchange. We model such precedence relations at task-level through the following boolean matrix *prec*:

$$prec_{ij} = \begin{cases} 1 & \text{if } j \neq i \text{ and } \tau_i \text{ is a predecessor of } \tau_j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.31}$$

As for *msg*, *prec* is of size $N_T \times N_T$ in the one-to-one integration strategy and $N_T^{as_a} \times N_T^{as_a}$ for each supplier $as_a \in AS$ in the one-to-all integration strategy.

According to this definition, $prec_{ii}$ is always null, as a task cannot be a predecessor and a successor of itself.

Our model supports the expression of precedence relations between tasks of different partitions. The corresponding partitions can correspond to the sameapplication or not. However, as mentioned in the assumptions, inter-application dependences are prohibited in IMA system. Nevertheless, no protection against an integrator exploiting our contributions and defining inter-applications dependences is implemented. The reason is to allow an integrator to guide the automated schedule generation process by setting some partial partitions windows orderings, for simplicity of management of production and consumption of messages occurring in one MAF. For the sake of simplicity of the explanations, such optional ordering preference choices are referred to as precedence in the rest of this thesis even though they do not actually correspond to precedence requirements of the system to be designed.

To represent inter-partition precedence relations, we define a following matrix *pPrec* as follows:

$$pPrec_{ij} = \begin{cases} 1 & \text{if } i \neq j \text{ and the windows of } \pi_j \text{ must occur after teh windows of } \pi_i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.32}$$

The size of *pPrec* is $N_P \times N_P$ in both integration strategies.

Identification of Interfering Instances in a given Schedule

To be able to detect when two instances of tasks on different cores are scheduled in parallel, we define a four-dimension matrix denoted *overlapping*, specifying whether the execution time interval of τ_i^k overlaps (even partially) with the one of τ_j^m.

Let $I_k(i) = \left[tO_i^k; tO_i^k + w_i^k\right]$ denote the time interval in which τ_i^k is executed. Then for each couple of task instances τ_i^k and τ_j^m, $overlapping_{i,j,k,m}$ is non-null if τ_i and τ_j are not on the same core and their respective intervals $I_k(i)$ and $I_m(j)$ overlap in time:

$$\begin{aligned} &\forall \tau_i, \tau_j \in \mathcal{T}, \forall (k,m) \in [1; nbActiv_i] \times [1; nbActiv_j], \\ &overlapping_{ijkm} = \\ &\begin{cases} 1 & \text{if } (I_k(i) \cap I_m(j) \neq \varnothing) \\ & \wedge \left(\sum_{r=1}^{N_P} PART_{ri} \times \sum_{p=1}^{N_C} a_{pr} \times \sum_{n=1}^{N_P} PART_{nj} \times a_{pn} = 0 \right), \\ 0 & \text{otherwise.} \end{cases} \end{aligned} \tag{4.33}$$

As will be explained in chapter 6, *overlapping* is used in order to guide the schedule generation process towards solutions implying as less overlapping executions as possible.

4.4 Hardware Architecture Model

As illustrated in figure 4.9, we model a multicore platform with N_C cores, each core p having a clock frequency of Clk_p respectively, as well as its own private cache memory, linked to the main memory via an interconnect. If all cores have the same clock frequency, the notation Clk can be used for simplicity.

For each core p, we define C_{SW_p} as an upper bound of the cores context switch overhead: C_{SW_p} represents a maximum bound of the runtime overhead experienced when: (i) saving the context of a task that just finished its execution on a given core and (ii) loading the context of the task that will be executed right afterwards. Similarly to the cores clock frequencies, if all cores are identical or have the same context switch overhead upper-bound, the notation C_{SW} can be used for simplicity.

L1 Cache

As mentioned before, we assume an architecture with only one cache level private to each core p, having a total size $cacheSize_p$ in kb, consisting in $nbCacheLines_p$ lines and an access latency of $cacheLat_p$ ns.

The waiting delay suffered by a task τ_i when waiting for a data request corresponding to a cache hit is accounted for in C_i^p, as it is the case when using aiT Analyzer [2] for example. All cache misses are assumed to lead to main memory accesses, and are thus accounted for in each H_i^k by definition. The corresponding waiting delay suffered by τ_i when issuing a request then consists in (i) the delay for the request to go through the interconnect, (ii) the delay to retrieve

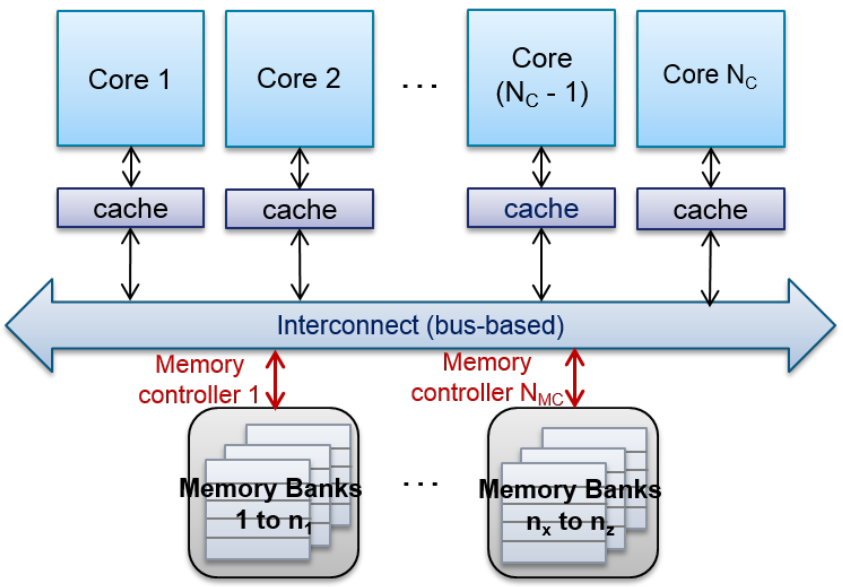


Figure 4.9: Multicore Hardware Architecture Model

the requested data in the main memory, but also (iii) the additional time spent by τ_i waiting for its requests to be elected and serviced by the memory controller. The first delay is referred to in this thesis as interconnect interference delay, and the latter two are merged into one delay referred to as main memory interference delay. These interference delays are not accounted for in C_i^p; they are computed using the timing analysis presented in chapter 5, and included in the corresponding WCET bound w_i, as will be explained in chapter 5.

Interconnect

The design of interconnects linking cores to the platform peripheral devices is often undisclosed by processors manufacturers because of IP protection. In order to cope with such an issue, we represent the interconnect using a simplified crossbar model in order to still be able to derive an upper-bound on interconnect interferences. In such a model, one request to the main memory can suffer interconnect interference caused only by requests that target the same memory controller.

In reality, interconnects between cores and the main memory often implement heuristics for optimizing requests arbitration at runtime in order to guarantee short average execution times. The resulting model results in shorter delays than with an actual crossbar as modeled in this thesis. As such, the model used in this thesis may be seen as a relevant model that is not limiting the usage of our work to any bus-based multicore COTS.

As a result, the interconnect enabling each core to access the main memory is represented as a crossbar implementing a first come first serve arbitration policy. Only requests to the same device are thus assumed to interfere with each other. As such, all requests directed to different memory controllers do not interfere, contrary to requests directed to the same memory controller. Indeed, for a given task τ_i allocated to a given core p, other tasks on other cores q contribute to τ_i's interconnect interference only if the task running on core q wants to access the same memory controller.

The latency corresponding to one main memory request traversing the interconnect is a hardware constant denoted l_{bus}, and can usually be found in the datasheet of the exploited multicore platform.

Main Memory

We consider a hardware platform with a main memory of $DRAMSIZE$ kb capacity. Let N_{MC} be the number of memory controllers available on the hardware platform. As illustrated in figure 4.9, each memory controller enables access to distinct memory areas. We define one integer

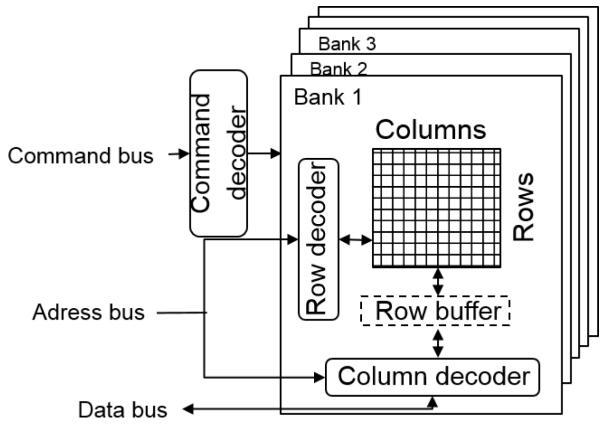


Figure 4.10: Model of Memory Banks

$mcSize_i$ per memory controller i to specify the size of memory area addressed by the memory controller i.

Finally, as shown in figure 4.10, the memory is divided into banks, and each bank is divided into columns and rows. When a request is treated by the memory controller, the bank to be accessed is identified first, then the row. When a task issues a memory request, the waiting delay suffered before the request is satisfied depends on the order in which the pending requests are treated by the memory controller.

Memory Path Allocation

In this thesis, we often use the expression ***memory path*** to refer to the usage of a specific memory controller of the hardware platform. When no confusion is possible, memory paths also refers to the choice of a path from a core to the main memory: as cores and main memory are linked via the interconnect and memory controllers, the choice of a memory path allocation is equivalent to the choice of a memory controller allocation.

If the considered multicore has only one memory controller, then the processor is said to have only one memory path. This corresponds to a high level simplification of the actual hardware implementation of the memory controller, by considering only one request can be serviced by a given memory controller at any given time.

Similarly, we mention cores ***sharing some memory path(s)*** to refer to two cores that have been allocated to the same memory controller(s).

As mentioned in the ”Assumptions” subsection, we assume no memory sharing between partitions except for inter-partition communications. However, it is important to note that not implementing any message passing does not mean being free of memory-related interference. Indeed, cores still share the path to the main memory, for multicore COTS seldom implement one path per core. Knowing which core uses which path to the memory depends on the memory allocation, but also on the hardware architecture design. As illustrated in figure 4.9, the number of paths to the main memory depends on the number of memory controllers.

We introduce a boolean matrix $c2mc$ to represent the mapping of the cores to the memory controllers available on the hardware platform.

$$c2mc_{ij} = \begin{cases} 1 & \text{if core } j \text{ is mapped to the memory controller } i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.34}$$

Matrix $c2mc$ represents the physically possible paths to each memory area, i.e. the hardware wiring of the core-to-main memory paths. It is defined in both strategies and has a size of

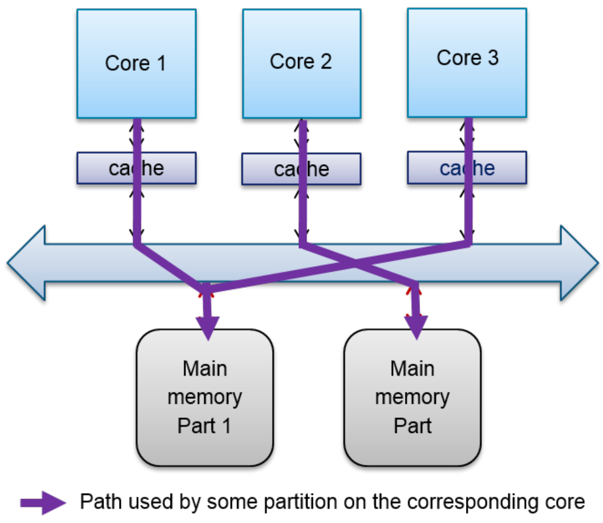


Figure 4.11: Example of Memory Path Sharing Scenario

$N_C \times N_{MC}$. To represent the actual software allocation to the memory paths, we define an additional boolean matrix depending on the integration strategy considered.

One-to-one integration strategy For the one-to-one strategy, the memory paths allocation is performed at partition-level. If a memory controller is allocated to one partition, it means the tasks of the partition will use the corresponding path. A task cannot use a path that corresponds to a memory controller that has not been allocated to its partition.

The partition-to-core allocation a and the core-to-path hardware implementation $c2mc$ determine which path can be used by each partition at runtime: indeed, a partition cannot be allocated to a path that is not wired to its core. As such, $c2mc$ gives all possibilities of memory paths available to the partitions depending on their respective allocations to the cores of the multicore platform.

On the other hand, a core may have access to two different memory paths, while its partitions may be using only one of these two paths. In such cases, the paths that are indeed used by each partition must be identified in order to improve the accuracy of the timing analysis and tighten the produced WCET bounds. To do so, we introduce the following boolean matrix $p2mc$ to model the allocation of the partitions to the memory paths:

$$p2mc_{ij} = \begin{cases} 1 & \text{if } \pi_i \text{ is mapped to the memory controller } j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.35}$$

The size of $p2mc$ is $N_{MC} \times N_P$.

One-to-all integration strategy In the case of the one-to-all integration strategy, the memory path allocation is done at task level. Following the same line of reasoning than for the one-to-one integration strategy, we define a boolean matrix $t2mc$ which gives the task-to-memory-path allocation:

$$t2mc_{ij} = \begin{cases} 1 & \text{if } \tau_i \text{ is mapped to the memory controller } j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.36}$$

The size of $t2mc$ is $N_{MC} \times N_T^{as_a}$ for each supplier $as_a \in AS$. Analogously to the partition-to-memory-path allocation, the same kind of restrictions apply to $t2mc$ regarding the hardware paths implemented on the platform: a task cannot be allocated to a path that is not wired to its core.

Shared Paths Identification

As briefly mentioned before, the identification of shared memory paths enables to tighten the computed WCRT bounds. It is important to note that we refer to "shared memory path" as

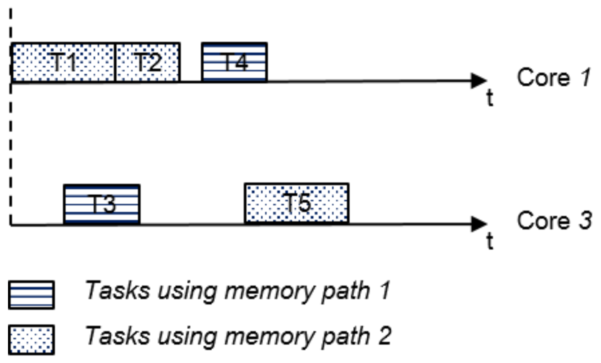


Figure 4.12: Core-Level Path Sharing and Runtime Interference

the sharing of the access to some memory controller, and not to the interconnect, since the interconnect is evidently shared by all cores whatever the memory and core allocation.

According to the interference computational model presented in chapter 5, shared paths must be identified at core level. If two patitions (resp. tasks) which share the same path to the main memory have been allocated to two different cores, then these two partitions are potentially interfering at runtime when issuing requests to access the main memory. The corresponding two cores are then also considered to share some path to the main memory.

It is important to note that such information is computed using partition- (resp. task-) to core allocations, and not from matrix *c2mc*: we are indeed interested in paths effectively used by cores at runtime, as given by definition by matrices *p2mc* and *t2mc* in the one-to-one and one-to-all integration strategies respectively, whereas *c2mc* states all physically possible paths to the memory for each core.

An example is given in figure 4.11 with three cores and two memory paths, assuming all cores share access to all memory controllers of the hardware platform. All terms of the corresponding matrix *c2mc* would then be equal to one to signify that all cores share access to all paths. The purple lines represent the memory path allocation to the partitions or the tasks depending on the implemented integration strategy, and therefore represents information contained in *p2mc* and *t2mc*. In this example, core 2 will suffer no memory interference at runtime, despite the fact that all cores are wired to both memory paths available to the memory area accessed by core 2.

Another example is given in figure 4.12, where tasks on different cores 1 and 3 share access to the same memory path but are never scheduled simultaneously, leading to zero inter-core interference during these tasks instances execution despite the fact that core 1 and 3 do share access to the same paths. As such, the information to take into account during analysis is the one given by matrices *p2mc* and *t2mc* in each strategy respectively, and not the information contained in *c2mc*, which would be likely to introduce unnecessary pessimism to the tasks WCET bounds.

One-to-one integration strategy To identify memory paths shared by cores in the context of the one-to-one integration strategy, we define the following boolean matrix $isSharingMC$:

$$isSharingMC_{pq} = \begin{cases} 1 & \text{if } p \neq q \text{ and cores } p \text{ and } q \text{ share some path to the main memory,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.37}$$

$IsSharingMC$ is of size $N_C \times N_C$ and can be deduced from matrices *p2mc* and a as follows:

$$isSharingMC_{pq} = \\ \begin{cases} 1 & \text{if } (i \neq j) \wedge \left(\sum_{i=1}^{N_P} \sum_{j=1}^{N_P} a_{pi} \times a_{qj} \times \left(\sum_{k=1}^{N_{MC}} p2mc_{ki} \times p2mc_{kj} \right) \neq 0 \right) \\ 0 & \text{otherwise.} \end{cases} \tag{4.38}$$

Indeed, if there exists at least one couple of partitions (π_i, π_j) allocated to cores p and q and which have been allocated to the same memory path k, then cores p and q host partitions that may interfere at runtime when simultaneously accessing the same main memory controller at runtime, and $isSharingMC_{pq}$ is therefore equal to one. If no such couple of partitions exists, partitions on cores p and q will not interfere with each other at runtime when trying to access the main memory, and $isSharingMC_{pq}$ is equal to zero.

One-to-all integration strategy In the one-to-all integration strategy, only one partition is running at a time. An important information is then to know not only whether two cores share some path to the memory, but also identify whether tasks of the same partition belonging to one of these two cores share some path to the memory. Indeed if the tasks of a same partition placed on different cores p and q do not share some memory path, then they will not cause any interference delay to each other, even if cores p and q use the same path to the memory for two different partitions. If tasks of a same partition but allocated to different cores do not share some memory paths, then they will never cause memory interference to one another, even if the cores they are allocated to share some memory paths.

As such in the one-to-all integration strategy, one must define one matrix *per partition* in order to identify whether simultaneous accesses to the same memory paths can occur at runtime in the windows of the corresponding partitions respectively. We define the following matrix $isSharingMC$:

$$\forall \pi_m \in \mathcal{P},\ isSharingMC_{mpq} = \begin{cases} 1 & \text{if } i \neq j \text{ and some tasks of } \pi_m \text{ respectively located on cores} \\ & p \text{ and } q \text{ share some path to the main memory,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.39}$$

The size of $isSharingMC$ is $N_P^{as_i} \times N_C \times N_C$ for each supplier as_i and can be deduced from matrices $PART$, $t2mc$ and na as follows:

$$\forall \pi_m \in \mathcal{P},\ isSharingMC_{mpq} = \begin{cases} 1 & \text{if } p \neq q \wedge \left(\sum_{i=1}^{N_T} PART_{mi} . \sum_{j=1}^{N_T} PART_{mj} . (na_{pi} \times na_{qj}) . \left(\sum_{k=1}^{N_{MC}} t2mc_{ki} \times t2mc_{kj} \right) \neq 0 \right) \\ 0 & \text{otherwise.} \end{cases} \tag{4.40}$$

Equation (4.40) is defined using an analog reasoning to the one described for the definition of equation (4.38). As mentioned before, the difference lays in the fact that cores and memory paths are allocated to tasks inside partitions rather than partitions themselves, and as illustrated in figure 4.1, tasks of the same partition are scheduled on all cores at any given time. As such, inter-core interference may occur between tasks of the same partition only, which must reflect on the definition of each $isSharingMC_{pqm}$ element.

4.5 Constraint Programming

In this thesis, we address the design space exploration search performed when looking for a software/hardware allocation, and when generating a schedule configuring the runtime behavior of the system to be designed. We propose to use constraint programming to express the allocation and scheduling problems involved in the proposed integration strategies.

Each CP defined in this thesis has one of the following two goals: generate an allocation, or generate a schedule. The corresponding set of constraints to be expressed in each CP depends on the considered strategy. However, in both strategies, the same requirements must be enforced, and therefore similar types of constraints must be expressed in each CP.

The goal of this subsection is to briefly present the main types of constraints that will be expressed when defining a CP for the allocation and the scheduling problems respectively. Extended details about the actual implementation of these constraints in the expressed CPs will be given later in chapter 6. The content of this section applies to both integration strategies proposed in this thesis.

4.5.1 Allocation Constraints

The allocation process consists in allocating the software platform – composed of partitions and tasks – onto the hardware platform – processing cores, main memory and memory paths. In the process of doing so, several conditions must be respected. In particular, the following categories of constraints are represented in all CP handling allocation concerns in both strategies proposed in this thesis:

- **Proper allocation constraints:** this type of constraints is focused on basic allocation constraints, such as ensuring all partitions and tasks are allocated exactly once to a core and at least to one memory controller. This type of constraints also includes consideration of allocation affinities and exclusion requirements, and the enforcement of the fact that all cores and memory paths must be exploited. Finally, constraints verifying the coherence of the evaluated allocations during the solving process are also covered (e.g. no overloaded core, memory controller allocation coherent with the available memory space addressable by each controller respectively, etc.)
- **Constraints performing a feasibility analysis:** this type of constraints ensures that the selected allocation corresponds to a feasible system, where it will be possible to find at least one schedule respecting all deadlines later during the schedule configuration phase. This includes the computation of tasks WCRTs and jitters, partitions CPU time budgets, etc.
- **Allocation Optimization Constraints** An example of allocation optimization constraint in the one-to-one integration strategy may be a constraint stating that if two partitions are involved in some communications, then they should be allocated to the same core and memory controller. This is considered as an optimization constraint since it is not expressing a requirement, and it enables to reduce the amount of potential multicore interference occurring at runtime. Indeed, it ensures the two partitions cannot try and access the shared space storing the exchanged data concurrently at runtime, but rather one after the other.

  Another example of optimization constraint, in the one-to-all integration strategy only, is a constraint limiting the number of shared memory controllers allocated to the tasks of the same partition. Indeed, by construction of the one-to-all schedule (see figure 4.6), tasks inside a same partition are susceptible to interfere at runtime, whereas tasks of different partitions can share memory controller without causing interference to each other.

CP solvers usually allow an objective function to be defined. The main goal of such a function is to order all valid solutions of the CP according to an optimization criteria. For instance in the case of an allocation CP, the minimization of the CPU workload is often expressed as the objective function. However, this does not correspond to the way CPU workload has been handled during the design cycle of currently existing avionics systems: a threshold value is usually set as a maximum value of CPU workload, in case additional software would be added to the module in later design versions of the system.

This threshold value guides the design process since it corresponds to a requirement of the system that must be enforced. However, previously designed systems relied on single-core platforms; systems targeted in this thesis are multicore-based and may therefore suffer from bigger workload sensitivity than monocore-based systems.

In fact, a threshold CPU workload value could be enforced as a hard requirement to the multicore allocation problem using the following constraint: ”the maximum CPU utilization ratio fixed by system designers must be enforced on all cores”. However system designers usually choose the threshold according to experience and good knowledge of the exploited hardware and software platforms.

In single-core environments, these thresholds are carefully studied and set using knowledge and know-how based on experience (previous systems, previous hardware and software behaviors) There exist no such knowledge and know-how in the multicore case to rely on when fixing a CPU utilization threshold. As a consequence, for the same software platform and the workload threshold defined for it in a monocore-based context, designers have no certainty that a solution in a multicore environment exists.

To easily cope with such uncertainty and save time and effort spent during the allocation phase, we suggest to define an objective function aiming at minimizing as much as possible each CPU workload, instead of defining a maximum threshold as a constraint when the existence of solutions respecting such a threshold is not known, and therefore taking the risk of such constraint rejecting all solutions that could have been considered valid, leading to a CP that fails to find a solution to the allocation problem.

Other than CPU workload minimization as an objective function for the allocation CP, minimizing multicore interference, or partitions CPU time budgets, are two suitable objective functions for an allocation CP as well.

4.5.2 Scheduling Constraints

The schedule generation problem consists in finding activation dates and computing time slots to be reserved for each partition (and tasks if the partition is non-preemptive at task-level). In the process of doing so, several conditions must be respected. In particular, the following categories of constraints are represented in all CP handling schedule generation concerns in both strategies proposed in this thesis:

- **Proper schedule generation constraints:** this type of constraints is focused on basic schedule generation constraints, such as ensuring the order of activation dates for a given task matches the execution instance number of the task. Another important constraint is to ensure that every activation date is an even multiple of the CPU clock, which defines the instants when tasks are selected for execution by the OS and therefore able to start executing.

  The overall coherence of the generated schedule belongs to this type of constraints as well: for instance, one must verify that all inter-task dependences defined in the software platform are respected. The absence of overlapping executions on the same core must also be verified.

- **Schedule verification constraints:** this type of constraints embraces a schedulability analysis in order to ensure all deadlines of the systems are enforced in the generated schedule. This includes the computation of tasks WCETs, the verification that all deadlines are enforced, but also that the corresponding partition windows are enforced.

- **Schedule optimization constraints:** An example of schedule optimization constraint may be a constraint stating that if a task is not scheduled concurrently to any other task on the other cores, then that task does not suffer from multicore interference at runtime. This is considered as an optimization constraints since multicore interference are sought to be minimized as much as possible in general.

As for allocation problems, it is possible to define an objective function to guide the CP search towards optimized schedules according to a custom criteria. An objective commonly chosen in the literature is the minimization of the total makespan. However, in IMA systems, it is more important to reduce as much as possible the size of partitions windows, but also multicore interference, than try and reduce the total makespan.

Another objective function that can be defined is to reduce as much as possible concurrent executions on the cores, as it reduces runtime competition of tasks trying to access the same shared resource.

4.5.3 Why Constraint Programming

CP was chosen as the approach for applying the integration strategies proposed in this thesis because it matched the best the requirements and context of this thesis. Indeed, using CP enables to focus on the problem formalization into matchematical variables and constraints, without having to study which solving algorithm must be used depending on the types of equations representing the problem. CP solvers that currently exist in the literature – no matter whether licensed or free for use – often present capabilities of selecting the right solving algorithm adapted to the type of equations of the expressed problem. In comparison, heuristics or user-defined algorithms for solving the same problem may require specific knowledge of the algorithms in order to use them properly.

Another justification lays in the portability of the usage of CP techniques. It is easy to change the problem formulation by adding, deleting or modifying its equations. If a heuristic was used instead, any change to the problem formulation may lead to changes to be made to the algorithm. For instance when using a greedy algorithm such as best-fit to allocate tasks onto a multicore platform, adding allocation criteria such as energy consumption limitation may require to code a new loop inside the algorithm for this characteristic to be taken into account. When using a CP formulation, it would be enough to just add an equation expressing the energy consumption limitation into the CP problem formulation, without having to modify the CP solving process and/or algorithm. As such, modifications to a CP problem does not require modifying the CP solving process and/or algorithm. For similar reasons also, it may be easier to move to a different CP solver than to switch heuristics for a given problem to be solved.

Finally, it is easier to optimize the solving process when using a CP formulation of a problem that has been formalized into a mathematical representation. When the solving process is taking too long, one has the possibility to change the way of expressing the problem, by changing the mathematical equations, adding new equations to help reduce the search space, etc. This can hardly be done when using less flexible solving approaches such as heuristics. Heuristics are usually selected in order to escape the intractability of some mathematical problems, but it is hard to optimize or help the algorithm behind the heuristic when it is stuck. At the beginning of this thesis, early work has been implemented using a heuristic named Midaco [147]. Midaco consists in a machine learning algorithm for solving nonlinear mixed integer problems. While this heuristic is able to explore large design spaces and proposes features for search optimization, one has to learn about the algorithm in order to truly benefit from these features. Eventually expressing the most complicated equations, the allocation problem that had been represented became intractable for Midaco.

Finally, it is important to note that having selected CP as a representation of the steps of the integration strategies proposed in this thesis does not limit the usage of this work. To reuse the contributions of this thesis, one would have to follow one of the two approaches for SW/HW integration described in chapter 6, which describes all the requirements that must be verified in order to guarantee a correct runtime behavior of the system.

4.6 Discussions

This section reviews some elements presented in this chapter. More specifically, this section discusses some modeling choices that have been made when representing the system model, and provides insight on the main motivation behind each choice.

One-to-All Integration Strategy: MAF cycle versus Partition Cycle. One may have noticed that, although the MAF may be longer than each partition cycle, it suffices to use the CPU time budgets per frame E_i^k of each partition cycle when building a partition schedule for the entire MAF, i.e. use a vector of size $nFrames_i$ to create time windows for π_i for a MAF having $nFrames$ frames.

The reason is that, in the one-to-all integration strategy, all interference are intra-partition. As such, inter-task interference – and therefore, the corresponding partition CPU time budgets – will not change depending on other partitions time windows in the schedule. The timing analysis performed during the allocation phase has produced bounds on E_i^k that are valid for each occurrence of E_i^k in the MAF of length $gMAF$. As such, the module integrator can work with the E_i^k budgets, as illustrated in figure 4.7.

It may be interesting to note that each application supplier may then find out that these budgets could have been tightened depending on the defined windows position in the MAF, but the computed configurations and analyses still remain safe as is. As the work in this thesis constitutes the first basis of legacy software transfer to multicore platforms with minimization of rework and enforcement of key certification aspects, partitions CPU time budgets value optimization can be considered as interesting future work for further increase design optimization.

Path Representation, Improvement of the Models Accuracy and the Tightening of Interference Bounds. It is important to note that a software platform which does not implement any message passing between partitions allocated ot different cores is not free of memory-related interference, as cores may use the same path to the main memory at runtime for computing purposes. Knowing which core uses which path to the memory depends on the path allocation, but also on the chip design. In the one-to-one integration strategy, the partition-to-core allocation a and the core-to-path hardware implementation $c2mc$ determine which path may be used by each partition at runtime and therefore have an impact on the possible values of the elements of $p2mc$. Such information can be valuable from the timing analysis point of view since it would enable to distinguish which tasks cannot interfere at runtime because they are allocated to different memory paths, as illustrated in the example given in figure 4.11. The same can be said in the one-to-all integration strategy, with matrices na, $c2mc$ and $t2mc$ instead of a, $c2mc$ and $p2mc$ respectively.

The knowledge of which partitions (resp. tasks) uses which main memory path is useful to compute tight WCRT bounds: at runtime, tasks which do not share any path to the main memory with a task τ_i can be ruled out from the computation of R_i and w_i. As such, although identifying which partition (resp. task) is using which memory path can be used to identify which partitions (resp. tasks of the same partition) share some path to the main memory, and then reject non-interfering requests from their WCRT and WCET computation.

It is important to note that it is necessary to represent both the hardware path wiring $c2mc$ and the software path allocation $p2mc$ (resp. $t2mc$). Matrix $c2mc$ gives all possibilities of memory paths available to the partitions (resp. tasks) depending on their core allocation. On one hand, a core may have access to two different memory paths, while its partitions (resp. tasks) may be using only one of these two paths. In such cases, the paths that are indeed used by each partition (resp. task) must be identified. On the other hand, some COTS may offer only one path per core, and thus $p2mc$ (resp. $t2mc$) can directly be derived from $c2mc$ and the core allocation a (resp. na). However, even in such cases $p2mc$ still holds valuable information in our model, since it enables to distinguish potentially interfering partitions from partitions that will never interfere because they do not use the same path to the memory. In addition, if only one memory path is possible per core, then partitions (resp. tasks) involved in message communications are forced to be allocated to the same core; in that case, $c2mc$ would have influenced the allocation a (resp. na), so it would still have held valuable information.

Finally, it is also important to note that memory paths are not represented in the approach in [84], which has been used as a basis for the interference analysis presented in chapter 5. As a result, interference upper-bounds computed by using the work in this thesis are likely to be more tight that the corresponding upper-bounds produced as a result of the application of the approach in [84]. As such, the work we present in this thesis improves the accuracy of the hardware model exploited for deriving WCET upper bounds compared to the approach proposed in [84].

Shared Memory. When a message is produced, both the sending and the receiving partitions (resp. tasks) need to be able to access the memory region where the message is stored after reception. The memory allocation an memory path allocation must take into account such constraints, as will be done in this thesis and explained in chapter 6.

For a given shared memory area, only the tasks of two partitions can access it: the partition containing the sender task and the partition containing the receiver task of the message for which that shared memory has been specified. This means the shared area can potentially be accessed simultaneously by two tasks: one for writing in memory, and another for reading in memory. As such, sharing memory in these specific conditions generates some interference between the two involved partitions. This translates at runtime into additional waiting delay the corresponding message producer and consumer tasks can experience due to writing/reading that message, if they simultaneously try to write/read the message in memory. Such a scenario is considered when computing the WCET of each task in order to consider to worst possible situation of interference due to shared resources. The corresponding interference delay suffered by each task sending and/or receiving at least one message is accounted for in the inter-task memory interference delays we compute thanks to our timing analysis, as will be explained in chapter 5.

A task sending a message has a ”write only” authorization on the shared memory space used for storing the corresponding message. A task receiving a message has a ”read only” authorization on the shared memory space used for storing the corresponding message. Such authorizations can be enforced at runtime using MMUs and health monitoring recovery actions in case of detected violations; it is important to note that MMUs are currently implemented in most COTS multicores. Such mechanisms also ensure that there cannot exist any problem of data overwriting by two different tasks at runtime.

According to the IMA standard, message-based communications are formalized in two different ways. Messages are either queued or sampled. Queueing is used when a message is destroyed after being read by the receiving task(s). Sampling is exploited in situations where a message is refreshed or rewritten in memory, and each receiving task works with the copy available at a given time or an old version of the message, depending on the schedule and on the specifications of the corresponding application. Situations of unplanned and undesired message overwriting by the task sending the corresponding message are excluded by good software development practices.

Either system designers decide it is not a problem, or they want to avoid such situations, in which case the shared memory space to be used for the corresponding message exchange is to be large enough to store more than one copy of the message, in order to allow for multiple copies to be stored in memory at the same time. In the later case, additional mechanisms for proper avoidance of data overwriting is assumed to be handled accordingly by software.

4.7 Summary

In this chapter, we briefly introduced the two integration strategies proposed as a contribution of this thesis. Each integration strategy includes a software/hardware allocation activity, and a schedule generation activity where a static schedule is built for partitions in one MAF. These two activities include timing analyses to verify beforehand that the configuration choices that were made actually correspond to valid designs so that all timing requirements of the system are guaranteed to always be respected at runtime.

Allocation and scheduling require different modelings of tasks and partitions; as such, tasks and partitions have two different representations respectively. This chapter presented all parameters and models used in this thesis, and mentioned whether they were used for the allocation and/or the schedule generation problems.

This chapter also explained how we represent the allocation and scheduling activities as CPs in order to automate these activities, but also gain time in the design cycle of a system, and ease the work of system designers and integrators. Although they could be expressed in one formulation, the allocation and schedule generation CPs are to be solved separately: the reasons lay both in the certification requirements and the difficulty of the CP solving process. Indeed, current certification requirements imply that the allocation and the schedule generation are not performed by the same persons. They also imply that allocation and schedule generation are not handled as one big activity of a system design but rather in an iterative process consisting in discussions, budgets assessment and verifications by different actors. Moreover, the allocation and scheduling problems are both NP-hard on their own, which means the solving time grows exponentially with the complexity of the expressed allocation or scheduling CP. This means that every allocation or scheduling CP expressed in this thesis may become too difficult to be solved even separately. As such, our first attempts at defining a combined CP performing both he allocation and the schedule generation in one CP led to very long solving times, contrary to when we expressed them as separate CPs.

Finally, this chapter expressed the list of constraints that must be verified either in the allocation or the schedule generation problems. Each list includes optional constraints, for optimization purposes. The chapter then concluded with a rationale subsection about the choice of definition of each CP formulation.

Chapter 5

Multicore Timing Analyses

This chapter presents the metrics proposed for performing timing analysis on multicore-based IMA systems. As mentioned before, in current IMA systems, the timing analysis is performed at task level first, and then derived at partition level. In fact, multicore interference are analyzed at task level in order to derive a safe WCET upper-bound for each task, before computing the corresponding partitions CPU time budget needs per partition cycle.

The first section of this chapter presents the computational model proposed in order to compute each task WCRT and WCET in the context of IMA software and multicore architectures. We then present in details our approach for upper-bounding multicore interference delays suffered by tasks at runtime due to sharing access to the main memory. The chapter then describes the proposed timing-related verification to be performed respectively during the allocation selection and the schedule generation.

It is important to note that this chapter is focused on how each timing analysis metric is built, without positioning them with regards to the integration strategies proposed in this thesis: this will rather be done in extended details in chapter 6.

Eventually, it is important to note that additional verification such as memory allocation or affinity/exclusion constraints are not explained in this chapter, which will rather be done in chapter 6. Although some definitions of feasibility and/or schedulability analysis in the literature may include considerations for other resources than time, the current chapter focuses on verifications that are strictly timing-related.

Finally, unless explicitly stated, all models and equations presented in this chapter are exploited in both integration strategies. The way they are used in each of them will be explained in chapter 6.

5.1 Tasks WCRTs and WCETs Computation

This section describes the computational model proposed in this thesis for the safe WCRT and WCET upper-bounds computation. Two timing analyses are proposed for WCET upper-bounding: one for the verification of feasibility of a given allocation configuration, and one for the verification of the enforcement of timing requirements in a given schedule configuration. Subsection 5.1.1 relates to the allocation problem, while subsection 5.1.2 relates to the schedule generation problem.

5.1.1 Tasks WCRTs and Allocation

The response time analysis consists in computing tasks WCRT upper-bounds and performing a feasibility analysis for a given system. To do so, the system of equations (2.3) is exploited in the case of a preemptive environment, equation (2.4) in the case of a non-preemptive environment.

These systems of equations suit single-core architectures, but neither multicore nor IMA architectures. We present here an adaptation suited for IMA and multicore architectures.

This thesis covers both preemptive and non-preemptive task sets. This has an impact on the response time equations to be used during the analysis. To ease understanding of the reader, we explain our derivations on the response time equations of preemptive tasks, before presenting the final equations implemented that are applicable to both preemptive and non preemptive task.

WCRT and IMA

The classic response time equations do not represent an equivalent of the IMA partition level. In IMA architectures, the partition level has an impact on tasks WCRT since each partition is allocated separate CPU time intervals in a schedule that condition which tasks are allowed to run during these intervals. In fact, a task τ_i can only be preempted or blocked by tasks of the same partition,the second equation of the system (2.7) can be updated as follows:

$$\forall \tau_i \in \mathcal{T}, w_i = C_i^{'} + \sum_{\substack{j=1 \\ pid_j = pid_i \wedge \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil C_j^{'} \tag{5.1}$$

where $C_i^{'}$ is the duration of execution of τ_i in isolation depending on the core it is executing on, and as defined in chapter 4 and $hp(\tau_i)$ is the set of tasks having a higher priority than τ_i as defined by equation (4.7).

Equations declined from equation (2.3) for task response time computation are often solved iteratively, R_i being a fixed point solution. Another way to solve equation (2.3) is to find the smallest integer value which satisfies equation (2.3). In our case, we implement the second resolution technique, by using constraint programming to define R_i as the minimum solution of equation (2.3).

Finally, we include consideration of context switch overheads using C_{SW} as follows:

$$\forall \tau_i \in \mathcal{T}, w_i = \left(C_i^{'} + C_{SW}\right) + \sum_{\substack{j=1 \\ pid_j = pid_i \wedge \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil \left(C_j^{'} + C_{SW}\right) \tag{5.2}$$

where C_{SW} represents the overhead corresponding to context switching from the task that just finished its execution to the next task to be scheduled, τ_i.

WCRT and multicore

The classic response time analysis considers that the entire task set is to be scheduled on the same core of a single-core hardware platform. An easy transposition to multicore environments that can be found in the literature is to consider each core as a separate computing resource. The system can be divided into as many sub-systems as there are cores, one subsystem corresponding to a core and the partitions (resp. tasks) that are allocated onto that core. For each subsystem, the system of equations (2.7) is then expressed while considering only tasks belonging to it, and $C_i^{'}$ corresponds to the task duration in isolation depending on the core τ_i (resp. its partition) is allocated to, but also depending on the integration strategy.

In the one-to-one integration strategy, the core allocation is done using the partition-to-core allocation matrix a. As such for each task τ_i, $C_i^{'}$ can be computed thanks to the following relation:

$$\forall \tau_i \in \mathcal{T}, C_i^{'} = \sum_{j=1}^{N_P} PART_{ji} \times \left(\sum_{p=1}^{N_C} a_{pj} \times C_i^p \right) \tag{5.3}$$

In the one-to-all integration strategy, the core allocation is done using the task-to-core allocation matrix na. As such for each task τ_i, $C_i^{'}$ can be computed thanks to the following relation:

$$\forall \tau_i \in \mathcal{T}, C_i^{'} = \sum_{p=1}^{N_C} na_{pi} \times C_i^p \tag{5.4}$$

Ignoring additional delays appearing in practice in multicore environments limits the accuracy of any feasibility analysis, as the computed WCRT and WCET bounds then do not actually correspond to safe bounds. In this thesis, we propose an approach to safely bound such additional interference delays. According to our assumptions and configuration choices (see chapter 4), multicore interference is represented by concurrent accesses to the interconnect leading to the main memory through its memory controllers.

The interconnect and main memory being used simultaneously by all active cores at runtime, additional delays do appear, and should therefore be safely taken into account in the computation of the WCRTs. While runtime overheads such as C_{SW} are hardware dependent and often modeled as additional constants added to a task execution duration in isolation as done in equation (5.2), interference delays are usually not constant over time and depend on the runtime activity of the other cores of the processor.

In the literature, interference due to shared resources is usually accounted for in the response time bounds by adding a sum of interference delays caused by accessing each shared resource separately. For instance, in the preemptive case, the updated system of equations (5.2) is the following one:

$$\forall \tau_i \in \mathcal{T}, \begin{cases} R_i = J_i + w_i \\ w_i = \left(C_i^{'} + C_{SW}\right) + \sum\limits_{\substack{j=1 \\ pid_j = pid_i \wedge \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil \left(C_j^{'} + C_{SW}\right) + \sum\limits_{k=1}^{\#Resources} I_{i,k}(w_i) \end{cases} \tag{5.5}$$

where $\#Resources$ represents the number of shared hardware resources of the multicore considered, and $I_k(w_i)$ is a maximum upper-bound of the waiting delay caused by the k^{th} resource. As stated earlier, this thesis focuses on interference due to sharing some memory space or simply sharing access to the main memory. As such, two upper-bounds of interference delays are produced for each task τ_i:

- The interference due to sharing (the path to) the main memory $d_{RAM}(w_i, H_i^p)$ depending on the task τ_i considered and the core p on which τ_i is executing,
- The interference due to sharing access to the interconnect linking the cores to the main memory $d_{INT}(w_i, H_i^p)$ depending on the same parameters.

The corresponding system of equation computing an upper-bound of tasks WCRTs in multicore environments is the following:

$$\forall \tau_i \in \mathcal{T}, \begin{cases} R_i = & J_i + w_i \\ w_i = & \left(C_i^{'} + C_{SW}\right) + \sum\limits_{\substack{j=1 \\ pid_j = pid_i \wedge \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil \left(C_j^{'} + C_{SW}\right) \\ & + d_{RAM}(w_i, H_i^p) + d_{INT}(w_i, H_i^p) \end{cases} \tag{5.6}$$

where p is the index of the core on which τ_i is executed. Detailed explanations on the implemented computational models to bound additional interference delays are given later in subsection 5.2 of this chapter.

Final Tasks WCRTs and WCETs

In this thesis, both preemptive and non-preemptive partitions are taken into account, which impacts the equations exploited to compute the WCRTs. By exploiting $isPreemptive_m$ booleans determining whether a partition π_m is preemptive at task level or not, we propose the following system of equations to cover both preemptive and non-preemptive cases when computing WCRTs:

$$\forall \pi_m \in \mathcal{P}, \forall \tau_i \in \mathcal{T} \mid pid_i = m,$$
$$\begin{cases} R_i = & J_i + w_i \\ w_i = & \left(C_i' + C_{SW}\right) + \sum_{\substack{j=1 \\ pid_j = pid_i \\ \tau_j \in hp(\tau_i)}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil \left(C_j' + C_{SW}\right) \\ & +(1 - isPreemptive_m) \times \left(\max_{\substack{\tau_j \in \mathcal{T} \setminus \{\tau_i\} \\ pid_j = pid_i \\ \tau_j \notin hp(\tau_i)}} \left(C_j' + C_{SW}\right) \right) \\ & + d_{RAM}(w_i, H_i^p) + d_{INT}(w_i, H_i^p) \end{cases} \tag{5.7}$$

where p is the index of the core on which τ_i is executed, C_i' is computed according to equation (5.3) in the one-to-one integration strategy, (5.4) in the one-to-all integration strategy; $d_{RAM}(w_i, H_i^p)$ and $d_{INT}(w_i, H_i^p)$ respectively are upper-bounds on the worst-case interference delay τ_i can suffer due to sharing access to the main memory and interconnect and are constructed as will be explained in subsection 5.2.

At this stage, the computed bounds on tasks WCRT R_i and WCET w_i suit IMA architectures, multicore platforms, and include safe bounds on inter-core interference.

5.1.2 Task Instances WCETs and Schedule Generation

Subsection 5.1.1 is about deriving WCRT upper-bounds for the allocation feasibility analysis. In this thesis, we also propose an approach to perform timing analysis during the schedule generation phase of a system design.

In IMA systems, the task-level scheduling problem is constrained by the partition level: the latter define valid time intervals within the boundaries of which their respective tasks are allowed to be executed on the corresponding processing resource.

The analysis proposed in this thesis for the scheduling problem covers situations where tasks of a partition can be either all preemptive or all non-preemptive. The task-level WCET and schedulability analysis are performed differently depending on whether tasks are preemptive or not.

If they are not preemptive, the schedule generation problem covers the computation of a static task-level schedule to be enforced at runtime. In this case, the schedulability analysis covers the verification of the enforcement of all timing requirements at task level in the computed task schedule, but also the partition-level schedule enforcement by the task-level schedules inside each partition respectively.

If the tasks of a partition are preemptive, the schedule generation problem rather focuses on performing a feasibility analysis for a given partition-level schedule, i.e. verifying that there would always exist some feasible task-level schedule for a given partition-level schedule imposed at task level.

The following paragraphs present how tasks WCETs are computed in the partition-level schedule generation phases. The approach depends on whether the considered tasks are preemptive or not.

Non-Preemptive Tasks

From the tasks point of view, the main difference between the allocation problem and the schedule generation problem is that the allocation verification focuses on the maximum WCRT each task could have at runtime in the worst-case situation, while the schedule verification verifies that, for a given schedule, each task instance is able to meet its deadline and dependence constraints. As such, in the allocation verification in the non-preemptive case, R_i is computed as if τ_i was scheduled after all higher priority tasks. The resulting WCET w_i then contains the corresponding delay τ_i spends waiting to start its execution, as expressed in the second equation of the system (5.7).

When performing the schedulability analysis, a schedule already exists, therefore it makes no sense anymore to include a delay corresponding to tasks other than τ_i being executed in the computation of its WCET w_i. The following equation can then be used instead of the second equation of the system (5.7):

$$\forall \tau_i \in \mathcal{T}, w_i = \left(C_i^{'} + C_{SW}\right) + d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p) \tag{5.8}$$

where p is the index of the core on which τ_i is executed.

Equation (5.8) computes only one WCET to be used per task as a unique WCET upper-bound for all instances of τ_i in one MAF. In contrast, computing one WCET upper-bound per instance may enable to tighten the WCET bounds for some instances, for example if they are not scheduled in parallel with any other task on the entire multicore platform.

As will be explained in chapter 6, the proposed CPs for schedule generation are able to tighten each WCET depending on the task instance considered. To benefit from such bound tightening abilities, the task model for the scheduling problem – as illustrated in figure 4.4 and presented in chapter 4 – proposed in this thesis defines one WCET upper-bound per task instance, each task therefore having a vector of WCET upper-bounds $(w_i^1, .. w_i^{nbActiv_i})$. Equation (5.8) can therefore be updated to compute the WCET of each task instance τ_i^k, denoted w_i^k, as follows:

$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i]$,

$$\begin{cases} tO_i^k \in [pO_{pid_i}^{\tilde{k}}; pO_{pid_i}^{\tilde{k}} + E_i^{\tilde{k}}] \text{ where } \tilde{k} \in [1; nFrames_{pid_i}] \mid (k-1)T_i = (\tilde{k}-1)P_{pid_i}, \\ w_i^k = \left(C_i^{'} + C_{SW}\right) + d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p) \end{cases} \tag{5.9}$$

where p is the index of the core on which τ_i is executed and $\tilde{k} \in [1; nFrames_{pid_i}]$ is the index of the frame of the partition cycle of τ_i corresponding to the execution of the k^{th} instance of τ_i.

Preemptive Tasks

If tasks are preemptive, a static schedule for tasks cannot be enforced at runtime since it is not possible to determine in advance when a task is going to be preempted at runtime. However, one can perform a feasibility analysis by considering each task instance occurring in a MAF as a separate task, executing only once per MAF, and which jitter upon first activation is equal to or greater than the start of the partition window in which the instance is scheduled.

For instance, let τ_i^k be the task representing the k^{th} instance of τ_i. Then τ_i^k can be considered as a separate task having the following parameters:

- τ_i^k is scheduled only once per MAF and has a period value of a MAF.
- The jitter upon first activation of τ_i^k is greater than or equal to $pO_{pid_i}^{\tilde{k}}$, where $\tilde{k} \in [1; nFrames_{pid_i}]$ is the index of the window in which τ_i^k is scheduled, computed using the relation (5.43). For simplicity, the parameter tO_i^k, corresponding to the task instance

activation date in a non-preemptive setup, is considered to represent the jitter upon first activation of τ_i^k in the case when τ_i is defined as preemptive.

- The priority of τ_i^k is equal to the priority of τ_i.
- The deadline of τ_i^k is equal to $k \times D_i$.
- The WCRT of τ_i^k is R_i^k.
- The WCET of τ_i^k is w_i^k.

In such a model, the WCRT computed for each instance τ_i^k must include a term taking into account the worst-case possible blocking delay suffered by this instance due to preemption by higher priority instances τ_j^m scheduled in the same frame as τ_i^k:

$$\sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i) \wedge \\ pid_j = pid_i}}^{N_T} \sum_{\substack{m=1 \\ (m-1)T_j = (k-1)T_i}}^{nbActiv_j} \left\lceil \frac{w_i^k + J_j^m}{T_j} \right\rceil \left(C_j' + C_{SW} \right) \tag{5.10}$$

The resulting analysis then consists in checking whether each instance WCRT respectively remains smaller than the corresponding instance deadline.

To sum up, if a partition tasks are preemptive, the computed schedule does not actually correspond to a schedule, and neither will it be enforced at runtime on the designed system. Each task instance activation date tO_i^k computed by a schedule generation CP then actually corresponds to the jitter upon activation of the k^{th} instance of τ_i, and w_i^k corresponds to the WCET of the k^{th} instance of τ_i as if it were a task executed only once per MAF and which jitter upon first activation is tO_i^k.

Such equivalence enables us to reuse the response time analysis equations for preemptive task sets that has been presented in the system (5.7) for the computation of WCRT and WCET upper-bounds. The resulting timing analysis evaluates partition schedules for partitions with preemptive tasks, i.e. checks whether a feasible task-level schedule could be found within the boundaries of a given partition-level schedule. The resulting equation to compute each task instance WCET is the following:

$$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i],$$

$$\begin{cases} tO_i^k \in [pO_{pid_i}^{\tilde{k}}; pO_{pid_i}^{\tilde{k}} + E_i^{\tilde{k}}] \text{ where } \tilde{k} \in [1; nFrames_{pid_i}] \mid (k-1)T_i = (\tilde{k}-1)P_{pid_i}, \\ w_i^k = \left(C_i' + C_{SW} \right) + \sum\limits_{\substack{j=1 \\ \tau_j \in hp(\tau_i) \wedge \\ pid_j = pid_i}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil \left(C_j' + C_{SW} \right) + d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p) \end{cases} \tag{5.11}$$

where p is the index of the core on which τ_i is executed and $\tilde{k}$ is the index of the frame of the partition cycle of τ_i corresponding to the execution of the k^{th} instance of τ_i.

Final Task Instances WCETs in the Schedule Generation Problem

The system (5.9) is exploited to compute WCETs upper-bounds for non-preemptive task instances, while the system (5.11) is exploited for preemptive tasks. The two systems can be combined into one unique system covering both situations thanks to the parameters $isPreemptive_{pid_i}$ enabling to know whether a task τ_i is preemptive or not. The resulting system of equations is the following:

$$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i],$$

$$\begin{cases} tO_i^k \in & [pO_{pid_i}^{\tilde{k}}; pO_{pid_i}^{\tilde{k}} + E_i^{\tilde{k}}] \text{ where } \tilde{k} \in [1; nFrames_{pid_i}] \mid (k-1)T_i = (\tilde{k}-1)P_{pid_i}, \\ w_i^k = & \Big(C_i' + C_{SW}\Big) + d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p) \\ & + isPreemptive_{pid_i} \times \left(\sum_{\substack{j=1 \\ \tau_j \in hp(\tau_i) \wedge \\ pid_j = pid_i}}^{N_T} \sum_{\substack{m=1 \\ (m-1)T_j = (k-1)T_i}}^{nbActiv_j} \left\lceil \frac{w_i^k + J_j^m}{T_j} \right\rceil \Big(C_j' + C_{SW}\Big) \right) \end{cases} \tag{5.12}$$

where p is the index of the core on which τ_i is executed and $\tilde{k} \in [1; nFrames_{pid_i}]$ is the index of the frame of the partition cycle of τ_i corresponding to the execution of the k^{th} instance of τ_i.

where p is the index of the core on which τ_i is executed, C_i' is computed according to equation (5.3) in the one-to-one integration strategy, (5.4) in the one-to-all integration strategy; $d_{RAM}(w_i, H_i^p)$ and $d_{INT}(w_i, H_i^p)$ respectively are upper-bounds on the worst-case interference delay τ_i can suffer due to sharing access to the main memory and interconnect and are constructed as will be explained in subsection 5.2.

At this stage, the computed bounds on tasks WCRT R_i and WCET w_i suit IMA architectures, multicore platforms, and include safe bounds on inter-core interference.

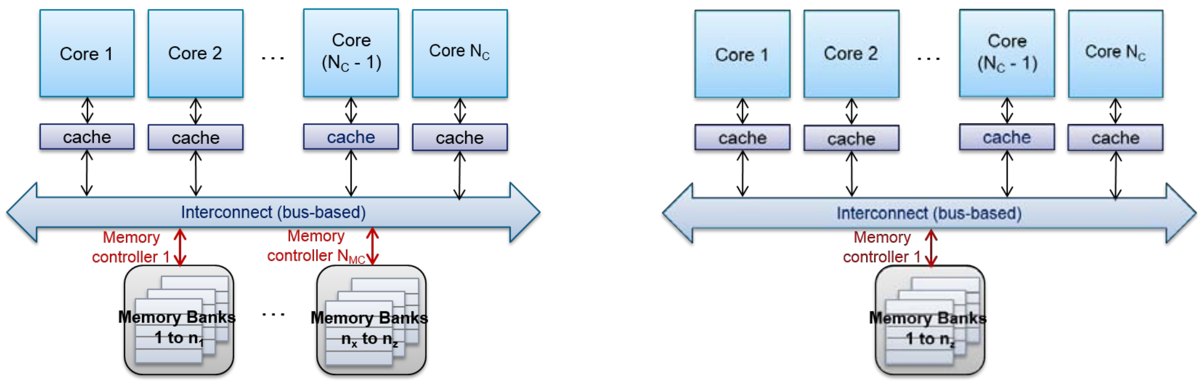
5.2 Multicore Interference Computation

In this section, we present the model we propose to compute safe upper-bounds of inter-core interference. We use a bottom-up approach in order to first bound interference at core level, then derive the corresponding interference delays at task level. The corresponding implications at partition level will be extracted from the task level when computing each partition CPU time budget per frame E_i^k, as will be explained later.

As mentioned in chapter 4, caches are private to each core, which implies the absence of inter-core cache interference. The multicore interference addressed in this thesis targets interference due to sharing the path from cores to the interconnect, and interference due to sharing the path from the interconnect to the main memory space through its memory controllers. To ease explanations, the former is referred to as ***interconnect interference***, and the latter as ***memory interference***. In order to compute interconnect and memory interference upper-bounds for each task, we define one function per interference type, denoted in previous equations as d_{RAM} and d_{INT} respectively. They are based on the hardware platform model and take as input the tasks characteristics in order to produce a safe upper-bound of the corresponding worst-case interference delays. In the current section, we present how we built functions d_{RAM} and d_{INT}. These two functions are currently under evaluation, which necessitates a software case study leading to a feasible schedule in which parallel executions of interfering tasks are occurring, in order to be able to compare the measured maximum interference delays suffered by tasks and the corresponding interference upper-bounds computed by functions d_{RAM} and d_{INT}.

Main Memory Interference Function, d_{RAM}

When accessing the main memory, a task may suffer additional waiting delays at runtime due to sharing the multicore resources with other tasks executed concurrently on other cores. The corresponding delay depends on the task WCET w_i and the maximum number of accesses to



(a) Multicore Architecture Covered by the Work in this Thesis

(b) Multicore Architectures Covered by the Approach in [84]

Figure 5.1: Multicore Architectures

the main memory it can generate during one execution. The main memory access latency is not constant over time and depends on other cores activities: indeed, tasks which execution time slot overlaps in time at least partially with τ_i's but on a different core may simultaneously emit requests to the main memory during the execution of τ_i.

We present here the mathematical model we propose for computing static upper-bounds of memory interference delays. The contributions described in this subsection have been presented in [118]; a first version, without path modeling and less tightened bounds, was first presented in [116].

Definitions and foreword

Whenever a task on a core wants to access a peripheral device of the hardware platform, it is translated into the corresponding core sending a request through the bus interconnecting cores to the peripheral devices. The bus is in charge of redirecting the request to the corresponding device. As such, when a task wnats to fetch some data from the main memory, the corresponding core emits a request through the bus, and the bus redirects the request to the main memory. Then entry point to the main memory is the memory controller. When more than one memory controller is available, the bus redirects the request to the memory controller handling the memory area targetet by the request.

In this subsection, we explain how d_{RAM} is computed. We used the model in [84] as a basis to our memory model. The approach in [84] relies on the knowledge of tasks to cores allocation, and the maximum number of requests to the main memory per task, to derive an upper-bound on the worst-case inter-core interference due to sharing access to the main memory. While very interesting, the work in [84] could not be reused as is in the context of this thesis. Indeed, as illustrated in figure 5.1b, the approach in [84] is focused on interference in multicore architectures having only one memory controller. As a consequence, the approach in [84] considers all memory requests of all cores to be redirected to the same memory controller, even though the real multicore processor has more than one memory controller, like the one illustrated in figure 5.1a for instance. This adds pessimism to the interference bounds computed for architectures where more than one memory controller is actually present. Indeed, two cores emitting a request to two different memory controllers do not create memory interference to each other. However, when using the work in [84], all simultaneous requests will be considered as interfering, even though in realty it is not the case because some requests are not redirected towards the same memory controller. Such false interference add unnecessary pessimism to the computed worst-case interference delay upper-bound. In contrast, the work in this thesis considers that more than one memory controller may exist. As such, only the requests redirected towards the same memory controller are considered as interfering.

Another characteristic made the approach in [84] not directly applicable to the context of this thesis: in fact, it is not directly applicable to IMA environments. Indeed, according to the IMA architecture, two tasks executing on the same core and belonging to two different partitions will never be able to preempt each other. These two tasks will then never be able to cause memory interference to each other as well. This situation is not covered in the approach presented in [84], where any task of higher priority than a given task is deemed perfectly able to preempt it at runtime. As such, the corresponding, worst-case interference delay due to the preempting task accessing the main memory during the preemption will be accounted for in the computed WRCT upper-bounds, including for preempting tasks actually not able to perform any preemption because it belongs to a different partition.

To sum up, while the work presented in [84] is very interesting, it relies on a system model that is inaccurate in the context of this thesis. It does not include IMA architecture or multi-memory path considerations, which therefore leads to WCRT upper-bounds that are overly pessimistic. To avoid introducing such unnecessary pessimism, we propose to modify some of the model's equations in order to extend the approach to IMA systems and multicore architectures featuring more than one memory controller.

The next subsection presents in details the approach for computing d_{RAM}, based on modyfing the equations first presented in [84]. The approach in [84] proposed two computation methods for d_{RAM} and chooses the minimum value of the two produced bounds for each task. In this thesis, we proceed similarly by deriving each method and then combining them to produce inter-task memory interference bounds. The maximum bounds produced thanks to the two methods will be referred to as $d_{RAM_{method_1}}$ and $d_{RAM_{method_2}}$ respectively.

Method 1: Request Driven Approach

The first approach computes an upper-bound on the maximum delay a request on one core can suffer. To do so, two types of interference are considered: inter- and intra-bank interference.

Inter-bank interference For a request *req* issued by core p, the worst case situation happens when: (i) every other core issued a request just before *req*; (ii) none of these requests targets the same memory bank as *req*; (iii) the management of each of these requests takes the longest latency possible, l_{max} – which happens when neither the row to access, nor the type of request are the same as for the previously issued request. The computation of l_{max} has first been expressed in [84]. Although to the approach in [84] is not applicable to the systems considered in this thesis, the definition of l_{max} given in [84] remains valid in our context of study since it is independent of the software architecture and solely hardware-dependent: its definition relies only on standard DRAM parameters that can be found in the memory's datasheet (see [84] or appendix 9.1 for a detailed description). As such, it is not a contribution of this thesis. For a detailed explanation on the computation of l_{max}, interested readers are invited either to read [84] or have a look at appendix 9.1.

In the one-to-one integration strategy, the inter-bank interference delay for a request issued by core p is computed as follows:

$$\forall p \in [1; N_C], RD_p^{inter} = \sum_{\substack{\forall q, q \neq p \wedge \\ shared_{pq} = \varnothing}} l_{max} \tag{5.13}$$

where $shared_{pq}$ is, as defined in [84], the set of memory banks shared by cores p and q. In this thesis, the $shared()$ parameters are represented through a boolean matrix *shared* instead of sets. This does not limit the usage of *shared* since all equations depending on *shared* in [84] rely on detecting whether the sets are empty or not, without focusing on their actual content. As a conclusion, the sets $shared_{pq}$ exploited in [84] have been substituted by the corresponding matrix element $shared_{pq}$ in this thesis. Finally, the formal definition of *shared* depends on the integration strategy considered.

In the one-to-one integration strategy, the definition of *shared* is the following one:

$$shared_{pq} = \begin{cases} 1 & \text{if } p \neq q \text{ and cores } p \text{ and } q \text{ share access to some memory area,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.14}$$

The values of the elements of *shared* are deduced from matrices a and ipc. In particular, $shared_{pq}$ is empty only if no partition on core q is sharing any memory area with any partition on core p. This translates into the terms a_{pi} and a_{qj} being equal to one for all partition π_i belonging to p and all partition π_j belonging to q, and the term $a_{pi} \times a_{qj} \times (ipc_{ij} + ipc_{ji})$ being equal to zero for all such couple of partitions. As a consequence, the emptiness of $shared_{pq}$ can be assessed in our model as follows:

$$\begin{aligned} &\forall (p,q) \in [1; N_C]^2 \mid p \neq q, \\ &shared_{pq} \begin{cases} = \varnothing & \text{if } \left(\sum\limits_{i=1}^{N_p} \sum\limits_{j=1}^{N_p} a_{pi} \times a_{qj} \times (ipc_{ij} + ipc_{ji}) = 0 \right), \\ \neq \varnothing & \text{otherwise.} \end{cases} \end{aligned} \tag{5.15}$$

In the one-to-all integration strategy, by construction of the task-to-core allocation, the matrix *shared* to be defined must enable to identify the couples of cores sharing some memory area, but *for each partition*. As such, in equation (5.13) each $shared_{pq}$ set is replaced by $shared_{pqj}$, where the new variable j represents the partition to which τ_i belongs to. Analogously to the one-to-one integration strategy, $shared_{pqj}$ can be deduced from matrices na and msg. In particular, $shared_{pqj}$ is empty only if no task of π_j allocated to core q is sharing some memory area with any other task of π_j located on core p. This can be expressed as follows:

$$\begin{aligned} &\forall \pi_j \in \mathcal{P}, \forall (p,q) \in [1; N_C]^2 \mid p \neq q, \\ &shared_{pqj} \begin{cases} = \varnothing & \text{if } \quad \sum\limits_{m=1}^{N_T} \sum\limits_{n=1}^{N_T} PART_{jm} \times na_{pm} \times PART_{jn} \times na_{qn} \\ & \qquad \times (msg_{mn} + msg_{nm}) = 0, \\ \neq \varnothing & \text{otherwise.} \end{cases} \end{aligned} \tag{5.16}$$

If equation (5.16) is injected in equation (5.13), one may notice that there now exists one inter-bank interference delay RD_p^{inter} per partition, and equation (5.13) is therefore inaccurate. As such, in the one-to-all integration strategy, we denote $RD_{p,j}^{inter}$ as the inter-bank interference delay suffered by core p when issuing one request to the main memory in the context of partition π_j. Equation (5.13) can then be replaced by the following equation:

$$\forall \pi_j \in \mathcal{P}, \forall p \in [1; N_C], \; RD_{p,j}^{inter} = \sum_{\substack{\forall q, q \neq p \\ shared_{pqj} = \varnothing}} l_{max} \tag{5.17}$$

Equations (5.13) and (5.17) take into account cores q which share some memory area with core p, according to the hardware model implemented in [84]. However, the memory model in [84] contains only one memory path, contrary to our hardware model which is able to represent – and take into account when computing interference delays – all memory paths available on a multicore platform. As such, in the example illustrated in figure 4.12, the approach in [84] would consider all tasks to be interfering with each other, and therefore compute an overly pessimistic WCET upper-bound due to the production of a significant memory interference upper-bound, which should in fact be equal to zero. This is valid for both the one-to-all and the one-to-one integration strategies.

As such, equations (5.13) and (5.17) must be updated since the *shared*() sets do not differentiate cores that are sharing the same memory paths from cores that do not. It is also important to understand that such a differentiation is not to be performed using the physical core-to-memory paths *c2mc*, but rather using the result of the partition-to-core and -memory allocation, which leads to the identification of the paths that are actually used by the cores at runtime. Indeed, a core may be able physically to use all paths to the memory while it is using only one of them at runtime.

If two cores embed partitions sharing some memory space, then these two cores must share the same path to the main memory in order for both partitions to be able to access the corresponding shared space. But if two cores do not share any memory space, then two situations are possible: either the two cores share some path to the memory, or they do not. If they do not share some memory path, then they will never interfere at runtime since they do not use the same paths, and as a consequence, such cores should not be taken into consideration when computing the memory interference of each other's tasks.

This is not done in the approach of Kim et al. [84]. In contrast, to do so in this thesis, we update equations (5.13) and (5.17) to take into account only cores sharing the same paths to the main memory by exploiting the boolean matrix *isSharingMC*. This matrix is built using the partitions (resp. tasks) allocation to identify the paths actually used by each core and then identify which couples of cores share the same memory paths at runtime. For the one-to-one integration strategy, equation (5.13) is updated as follows:

$$\forall \pi_j \in \mathcal{P}, \forall p \in [1; N_C], RD_{p,j}^{inter} = \sum_{\substack{\forall q, q \neq p \\ shared_{pq} = \varnothing}} isSharingMC_{pq} \times l_{max} \tag{5.18}$$

whereas for the one-to-all integration strategy, equation (5.17) is updated as follows:

$$\forall \pi_j \in \mathcal{P}, \forall p \in [1; N_C], RD_{p,j}^{inter} = \sum_{\substack{\forall q, q \neq p \\ shared_{pqj} = \varnothing}} isSharingMC_{jpq} \times l_{max} \tag{5.19}$$

Eventually, it is important to note that such usage of *isSharingMC* to exclude cores that do not share the same memory path from the memory interference computation of another core is required in the rest of this thesis every time the *shared*() sets intervene in some equations.

Intra-bank interference When requests to the same bank as *req* are issued, the longest interference delay suffered by a memory access request *req* to be serviced happens when: (i) every other core q that shares access to the same bank as core p emitted a request that is serviced by the memory controller before *req*; (ii) all these requests target a different row; (iii) a memory refresh – or requests reordering – is happening. If L is the delay to open a row before accessing a column, then the worst-case delay per such request is $L + RD_q^{inter}$. As for l_{max}, the definition of L expressed by Kim et al. in [84] remains valid in our context of study since it is independent of the software architecture model and solely hardware-dependent: its definition relies only on standard DRAM parameters that can be found in the memory's datasheet (see [84] or appendix 9.1 for a detailed description). As such, it is not a contribution of this thesis. For a detailed explanation on the computation of L, interested readers are invited either to read [84] or have a look at appendix 9.1.

In the one-to-one integration strategy, the intra-bank interference delay suffered by *req* issued by a core p is thus:

$$\forall p \in [1; N_C], RD_p^{intra} = reorder(p) + \sum_{\substack{\forall q, q \neq p \\ shared_{pq} \neq \varnothing}} (L + RD_q^{inter}) \tag{5.20}$$

Reorder(p) computes an upper-bound of the maximum delay suffered by a memory request *req* due to the reordering effect. For the same reasons as for l_{max} and L, the computation of *reorder*(p) has first been expressed in [84] and the given definition remains valid in our context of study. It is therefore not a contribution of this thesis; as such, for a detailed explanation of $reorder(p)$'s definition, interested readers are invited either to read [84] or have a look at appendix 9.1.

The definition of RD_p^{intra} must take into account the hardware model refinement proposed as a contribution of this thesis. Indeed, the definition proposed in [84] must be modified in order to take into account cores q that share some memory path with core p. In the one-to-one integration strategy, the corresponding equation is the following:

$$\begin{aligned}&\forall p \in [1; N_C],\\ &RD_p^{intra} = reorder(p) + \sum_{\substack{\forall q, q \neq p\\ shared_{pq} \neq \varnothing}} isSharingMC_{pq} \times (L + RD_q^{inter})\end{aligned} \tag{5.21}$$

For the one-to-all integration strategy, the corresponding equation is the following:

$$\begin{aligned}&\forall \pi_j \in \mathcal{P}, \forall p \in [1; N_C],\\ &RD_{p,j}^{intra} = reorder(p) + \sum_{\substack{\forall q, q \neq p\\ shared_{pqj} \neq \varnothing}} isSharingMC_{pq} \times (L + RD_q^{inter})\end{aligned} \tag{5.22}$$

Finally, the total maximum interference delay RD_p that a request originating from core p can experience is computed as follows in the one-to-one integration strategy:

$$\forall p \in [1; N_C], \quad RD_p = RD_p^{inter} + RD_p^{intra} \tag{5.23}$$

In the one-to-all integration strategy, the maximum delay suffered by a request emitted by core p actually depends on the partition executed during the emission of the request:

$$\forall \pi_j, \forall \pi_j \in \mathcal{P}, \forall p \in [1; N_C], \quad RD_{p,j} = RD_{p,j}^{inter} + RD_{p,j}^{intra} \tag{5.24}$$

Each task τ_i of core p having H_i^p requests to issue, the total maximum interference delay directly caused by issuing these requests is bounded by the number of requests H_i^p multiplied by the maximum delay of one request: RD_p in the one-to-one integration strategy, RD_{p,pid_i} in the one-to-all integration strategy. In addition, the cost of memory requests of tasks with higher priorities than τ_i has also to be accounted for: either the corresponding partition is preemptive at task-level, in which case the requests of all tasks with higher priorities than τ_i extend the delay during which τ_i is preempted and waits to resume its execution; or the task-level is non-preemptive, in which case the additional delay corresponds to the requests emitted by higher priority tasks blocking τ_i's execution.

In the end, in the one-to-one integration strategy, a maximum bound on the memory interference delay a task $\tau_i \in \mathcal{T}$ allocated to core $p \in [1; N_C]$ can experience is computed as follows:

$$\begin{aligned}&dram_{method_1}(w_i, H_i^p) = H_i^p \times RD_p\\ &+ \sum_{m=1}^{N_P} PART_{mi} \times \left(isPreemptive_m \times \left(\sum_{\substack{j=1\\ \tau_j \in hp(\tau_i) \wedge\\ pid_j = pid_i}}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil H_j^p \times RD_p \right)\right)\\ &+ \sum_{m=1}^{N_P} PART_{mi} \times \left((1 - isPreemptive_m) \times \sum_{\substack{j=1\\ \tau_j \in hp(\tau_i) \wedge\\ pid_j = pid_i}}^{N_T} H_j^p \times RD_p \right)\end{aligned} \tag{5.25}$$

where p is the index of the core on which τ_i is scheduled. In the one-to-all integration strategy, the corresponding equation is the following:

$$d_{RAM_{method_1}}(w_i, H_i^p) = \sum_{m=1}^{N_P} PART_{mi} \times (H_i^p \times RD_{p,m})$$
$$+ \sum_{m=1}^{N_P} PART_{mi} \times \left(isPreemptive_m \times \left(\sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ pid_j = pid_i}} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil H_j^p \times RD_{p,m} \right) \right)$$
$$+ \sum_{m=1}^{N_P} PART_{mi} \times \left((1 - isPreemptive_m) \times \sum_{\substack{\forall j, \tau_j \in hp(\tau_i) \\ pid_j = pid_i}} H_j^p \times RD_{p,m} \right) \tag{5.26}$$

where p is the index of the core on which τ_i is scheduled.

Method 2: Job Driven Approach

The second method of computation of $dram_i()$ focuses on how many interfering memory requests per core are generated during the execution of τ_i; the time interval to consider is thus w_i. It may be interesting to note that the authors of [84] used R_i in their approach, since their model did not represent jitters upon activation, contrary to the work in this thesis.

The maximum number of requests $A_p(w_i)$ generated by a core p during a time interval of length w_i is computed as follows:

$$\forall \tau_i \in \mathcal{T}, \forall p \in [1; N_C], A_p(w_i, H_i) = \sum_{j=1}^{N_T} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil H_j^p \tag{5.27}$$

Equation (5.27) is valid for both integration strategies. It is important to note that p is not necessarily the index of the core on which τ_i is located yet.

Inter-bank interference The inter-bank interference a task τ_i located on a core p can suffer during its execution can be computed by multiplying the maximum number of generated requests by the maximum delay possible to serve a request to the same bank as req. For the one-to-one integration strategy for example, the corresponding interference delay is computed as follows:

$$\forall \tau_i \in \mathcal{T}, JD_p^{inter}(w_i, H_i^p) = \sum_{\substack{q=1 \\ q \neq p \wedge \\ shared_{pq} = \emptyset}}^{N_C} A_q(w_i, H_i^p) \times l_{max} \tag{5.28}$$

where, this time, p is the index of the core where τ_i is located.

Thanks to our models, equation (5.28) can be further refined to include knowledge on the memory path sharing by the cores, and thus reduce the pessimism of the final WCRT bound. In fact, the activity on core q only interfere during τ_i's execution if there exists a partition (in the one-to-one integration strategy) or task of the same partition (in the one-to-all integration strategy) allocated to core q which shares some memory path with τ_i. Otherwise, core q must be ignored when computing τ_i's memory interference delay. Therefore equation (5.28) is modified as follows in the one-to-one integration strategy:

$$\forall \tau_i \in \mathcal{T}, JD_p^{inter}(w_i, H_i^p) = \sum_{\substack{q=1 \\ q \neq p \wedge \\ shared_{pq} = \emptyset}}^{N_C} isSharingMC_{pq} \times A_q(w_i, H_i^p) \times l_{max} \tag{5.29}$$

where p is the index of the core onto which τ_i is allocated.

In an analogous way for the one-to-all integration strategy, intra-bank interference suffered by a request *req* can be computed using the following equation:

$$\forall \tau_i \in \mathcal{T},\quad JD_p^{inter}(w_i, H_i^p) = \sum_{\substack{q=1 \\ q \neq p \wedge \\ shared_{pqi} = \varnothing}}^{N_C} isSharingMC_{pqpid_i} \times A_q(w_i, H_i^p) \times l_{max} \tag{5.30}$$

where p is the index of the core on which τ_i is scheduled.

Intra-bank Interference The maximum intra-bank interference $\tau_i \in \mathcal{T}$ can suffer during its execution when allocated to core $p \in [1; N_C]$ can be computed as follows in the one-to-one integration strategy:

$$JD_p^{intra}(w_i, H_i^p) = \sum_{\substack{q=1 \\ q \neq p \wedge \\ shared_{pq} \neq \varnothing}}^{N_C} A_q(w_i, H_i^p) \times L + JD_q^{inter}(w_i, H_i^p) \tag{5.31}$$

In the one-to-one strategy, similarly to JD_p^{inter}'s refinement, JD_p^{intra} can be further tightened by considering only cores containing partitions sharing some memory path with τ_i:

$$JD_p^{intra}(w_i, H_i^p) = \sum_{\substack{q=1 \\ q \neq p \wedge \\ shared_{pq} \neq \varnothing}}^{N_C} isSharingMC_{pq} \times A_q(w_i, H_i^p) \times L + JD_q^{inter}(w_i, H_i^p) \tag{5.32}$$

where p is the index of the core on which τ_i is scheduled.

In a similar way, in the one-to-all integration strategy, the intra-bank interference of a request generated by a task τ_i belonging to a partition π_j is derived as follows:

$$\forall \pi_j \in \mathcal{P} \mid pid_i = j,\quad JD_{p,j}^{intra}(w_i, H_i^p) = \sum_{\substack{q=1 \\ q \neq p \wedge \\ shared_{pqj} \neq \varnothing}}^{N_C} isSharingMC_{jpq} \times A_q(w_i, H_i^p) \times L + JD_{q,j}^{inter}(w_i, H_i^p) \tag{5.33}$$

where p is the index of the core on which τ_i is scheduled.

Finally, if τ_i is located on core p, then a maximum bound of τ_i's memory interference is given by equation (5.34) in the one-to-one integration strategy.

$$\forall \tau_i \in \mathcal{T},\ d_{RAM_{method_2}}(w_i, H_i^p) = JD_p^{inter}(w_i, H_i^p) + JD_p^{intra}(w_i, H_i^p) \tag{5.34}$$

where p is the index of the core on which τ_i is executed. In the one-to-all integration strategy however, the following equation is used instead:

$$\forall \tau_i \in \mathcal{T},\quad d_{RAM_{method_2}}(w_i, H_i^p) = \sum_{m=1}^{N_P} PART_{mi} \times \left(JD_{p,m}^{inter}(w_i, H_i^p) + JD_{p,m}^{intra}(w_i, H_i^p)\right) \tag{5.35}$$

where p is the index of the core on which τ_i is scheduled.

Final Memory Access Delay Function

Methods 1 and 2 being about the computation of maximum bounds, $dram_i$ is set to the less pessimistic of the two:

$$\forall \tau_i \in \mathcal{T}, d_{RAM}(w_i, H_i^p) = min(d_{RAM_{method_1}}(w_i, H_i^p), d_{RAM_{method_2}}(w_i, H_i^p)) \tag{5.36}$$

where p is the index of the core on which τ_i is executed. During the WCRT analysis, p is computed using either matrix a in the one-to-one integration strategy, or na in the one-to-all integration strategy.

In the one-to-all integration strategy, equation (5.36) is replaced by the following one:

$$\begin{aligned} &\forall \tau_i \in \mathcal{T}, d_{RAM}(w_i, H_i^p) = \\ &\sum_{p=1}^{N_C} na_{pi} \times \Big(min(d_{RAM_{method_1}}(w_i, H_i^p), d_{RAM_{method_2}}(w_i, H_i^p))\Big) \end{aligned} \tag{5.37}$$

where p is the index of the core on which τ_i is executed.

In the one-to-one integration strategy, equation (5.36) is updated as follows:

$$\begin{aligned} &\forall \tau_i \in \mathcal{T}, d_{RAM}(w_i, H_i^p) = \\ &\sum_{m=1}^{N_P} PART_{mi} \times \sum_{p=1}^{N_C} a_{pm} \times \Big(min(d_{RAM_{method_1}}(w_i, H_i^p), d_{RAM_{method_2}}(w_i, H_i^p))\Big) \end{aligned} \tag{5.38}$$

where p is the index of the core on which τ_i is executed.

Eventually, the so defined $d_{RAM}()$ function can be exploited with tasks respective WCETs w_i during the allocation feasibility analysis, or with task instances respective WCETs w_i^k during the schedulability analysis.

Interconnect Interference Function, d_{INT}

The worst-case interference a task τ_i can suffer due to accessing the interconnect linking the cores to the main memory is upper-bounded via the usage of $d_{INT}()$.

Crucial information to be able to accurately represent the interconnect through a function d_{INT} is often undisclosed by processors manufacturers because of IP protection. In order to cope with such an issue, we represent the interconnect using a simplified crossbar model in order to still be able to derive an upper-bound on interconnect interferences. In such a model, one request to the main memory can suffer interconnect interference caused only by requests that target the same memory controller. As such, for a given task τ_i allocated to a given core p, other tasks on other cores q contribute to τ_i's interconnect interference only if the task running on core q requests access to the same memory controller.

Such situation can be identified clearly thanks to matrix *overlapping* during the schedule generation problem, but not during the allocation. As such, the computation of d_{INT} differs slightly depending on which activity – allocation or schedule generation – is considered. It also depends on the integration strategy implemented.

Allocation. In the allocation problem, an upper-bound on tasks interconnect interference is computed using the number of cores that share access to the same memory path. For a given task, the worst-case situation is consider to be the one where all other cores sharing access to the same memory controller to have issued a request to that controller, and the corresponding request to be serviced before any request emitted by that task to the memory controller.

In the one-to-one integration strategy, the corresponding delay is computed as follows:

$$\forall \tau_i \in \mathcal{T}, d_{INT}(w_i) = \sum_{p=1}^{N_C} a_{p,pid_i} \times \left(l_{bus} \times \sum_{\substack{q=1 \\ q \neq p}}^{N_C} isSharingMC_{pq} \right) \tag{5.39}$$

In the one-to-all integration strategy, the equation is similar but based on na:

$$\forall \tau_i \in \mathcal{T}, d_{INT}(w_i) = \sum_{p=1}^{N_C} na_{pi} \times \left(l_{bus} \times \sum_{\substack{q=1 \\ q \neq p}}^{N_C} isSharingMC_{p,q,pid_i} \right) \tag{5.40}$$

Scheduling. In the schedule generation problem, the interconnect interference is computed for a given schedule, where task instances execution intervals are known. As such, it is possible to take such knowledge into account in order to isolate cores where potential interference can actually occur during the execution of a given task instance τ_i^k when computing its interference delay due to sharing the interconnect.

In our model, it is possible to do so using matrix *overlapping* defined in equation (4.33) (see chapter 4). The corresponding interconnect interference delay for the schedulability analysis corresponds to a refinement of the interference delay computed during the allocation phase: thanks to matrix *overlapping*, only cores sharing access to the same memory paths which schedule at least one task instance, also allocated to the corresponding controller, in parallel of τ_i.

In the one-to-one integration strategy, the interconnect interference delay for the schedulability analysis is computed as follows for each task τ_i:

$$\begin{aligned} &\forall \tau_i \in \mathcal{T}, d_{INT}(w_i) = \\ &\sum_{p=1}^{N_C} a_{p,pid_i} \times l_{bus} \times \sum_{\substack{q=1 \\ q \neq p}}^{N_C} \left(\sum_{\substack{j=1 \\ j \neq i}}^{N_T} a_{qj} \times \sum_{l=1}^{nbActiv_j} overlapping_{i,j,k,l} \times isSharingMC_{pq} \right) \end{aligned} \tag{5.41}$$

In the one-to-all integration strategy, the corresponding equation is the following:

$$\begin{aligned} &\forall \tau_i \in \mathcal{T}, d_{INT}(w_i) = \sum_{p=1}^{N_C} na_{pi} \times l_{bus} \\ &\times \sum_{\substack{q=1 \\ q \neq p}}^{N_C} \left(\sum_{\substack{j=1 \\ j \neq i \\ pid_j = pid_i}}^{N_T} na_{qj} \times \sum_{l=1}^{nbActiv_j} overlapping_{i,j,k,l} \times isSharingMC_{p,q,pid_i} \right) \end{aligned} \tag{5.42}$$

Eventually, the so defined $d_{INT}()$ functions in the one-to-one and one-to-all integration strategies can be exploited with tasks respective WCETs w_i during the allocation feasibility analysis, or with task instances respective WCETs w_i^k during the schedulability analysis.

5.3 Partitions CPU Time Budgets Computation

This section presents how IMA partitions CPU time budgets are computed in this thesis. As mentioned in chapter 4, each partition cycle can be divided into $nFrames_i$ frames, the k^{th} frame requiring a CPU time budget of E_i^k for its tasks execution.

As such, for each partition π_i and frame $k \in [1; nFrames_i]$, the value of E_i^k must correspond to enough CPU time for all tasks of π_i scheduled in the k^{th} frame to start and complete their respective executions within the boundaries of the k^{th} window of π_i. E_i^k being the duration of that window, it must take into account the worst-case multicore interference delays that the tasks of π_i scheduled in the k^{th} window may suffer.

On the other hand, a task τ_j belonging to π_i is scheduled in the k^{th} frame if the following relation is verified:

$$(k-1) \equiv 0 \mod \left(\frac{nFrames_i}{nbActiv_j} \right) \tag{5.43}$$

If equation (5.43) is verified, then the execution of a task τ_j belonging to π_i during the k^{th} frame must be taken into account in the budget E_i^k. To do so, the amount of time corresponding to τ_j holding the CPU must be evaluated: it corresponds to the actual execution of τ_j plus the interference delays suffered by τ_j, since it does not release the CPU while waiting for its memory requests to be serviced. For instance for the allocation problem, assuming τ_j is located on core $p \in [1; N_C]$, this corresponds in total to $C_j^p + d_{RAM}(w_j, H_j^p) + d_{INT}(w_j, H_j^p)$.

An alternate way of expressing the relation between the index $\tilde{k}$ of the frame in which a task instance τ_i^k is scheduled can be summed up in the following equation:

$$(\tilde{k} - 1) \times P_{pid_i} = (k - 1) \times T_i \tag{5.44}$$

As implied by figures 4.5 and 4.6, although the general definition of the partitions CPU time budgets per frame is the same for both strategies, their computation depends on which of the two strategies is considered.

One-to-one integration strategy Each partition CPU time budget per frame k of the corresponding partition cycle is computed as follows in the one-to-one integration strategy:

$$\begin{aligned} &\forall \pi_i \in \mathcal{P}, \forall k \in [1; nFrames_i], \\ &E_i^k = \sum_{p=1}^{N_C} a_{pi} \times \left(\max_{\substack{j=1 \\ pid_j = i \wedge \\ (k-1)\times P_i = (k-1)\times T_j}} \left(C_j^p + d_{RAM}(w_j, H_j^p) + d_{INT}(w_j, H_j^p) \right) \right) \end{aligned} \tag{5.45}$$

where the first sum enables to retrieve the index p of the core on which π_i is located, and the condition $(k-1) \times P_i = (k-1) \times T_j$ on the second sum enables to select only tasks of π_i that are scheduled in the k^{th} time window of their partition.

Finally, the maximum WCRT of such tasks is taken instead of summing all tasks WCRTs, because the definition of WCRTs accounts for tasks with higher priorities preempting or being scheduled prior to the corresponding task. The maximum WCRT therefore corresponds to the task of π_i with the smallest priority level, which scheduled only after all other tasks and preempted as many times as possible from its release to the end of its execution.

One-to-all integration strategy In the one-to-all integration strategy, as illustrated in figure 4.6, the computation of E_i is a bit less straightforward, as it depends on the core allocation.

Let $E_{i,p}$ denote the CPU time budget required by tasks of a partition π_i belonging to core p. Then $E_{i,p}$ is computed as done in equation (5.45) for the one-to-all integration strategy:

$$\begin{aligned} &\forall p \in [1; N_C], \forall \pi_i \in \mathcal{P}, \forall k \in [1; nFrames_i], \\ &E_{i,p}^k = \left(C_{SW_p} + \max_{\substack{j=1 \\ pid_j = i \wedge \\ (k-1)\times P_i = (k-1)\times T_j}} na_{pj} \times (R_j) \right) \end{aligned} \tag{5.46}$$

where na_{pj} selects only the tasks of π_i allocated to core p. This equation corresponds to a pessimistic but safe upper-bound of the time duration necessary for the k^{th} time window of π_i.

Tasks of a same partition being scheduled in overlapping time intervals on each core according to figure 4.6, the total CPU time budget used by π_i corresponds to the maximum value of $E_{i,p}$, for all core p of the considered multicore. As a consequence, E_i^k is computed as follows for each partition π_i and each frame k:

$$\forall \pi_i \in \mathcal{P}, \forall k \in [1; nFrames_i], E_i^k = \max_{p \in [1; N_C]} \left(E_{i,p}^k \right) \tag{5.47}$$

5.4 Allocation and Timing-Related Verification

This section presents the timing analysis performed during the allocation search. The goal of such an analysis is to ensure that a given software/hardware allocation corresponds to a schedulable system. The task and partition models for the allocation verification (see chapter 4) are used during this analysis.

The proposed timing-related verification can be divided into two-level: task-level verification, and partition-level verification. Unless explicitly stated, all equations presented in this section are valid in both the one-to-one and the one-to-all integration strategies.

Task-level Timing Verification during the Allocation

The goal of the task-level verification is to compute tasks WCRT for a given software/hardware allocation, and verify whether the computed values are smaller than the corresponding deadlines. As such, the software/hardware allocation is exploited as an input to the analysis in order to compute the respective values of the parameters R_i, w_i and J_i for each task τ_i as outputs. While R_i and w_i are computed using the system of equation (5.7), J_i must be determined according to the dependences defined in the software platform, as will be explained in the next paragraphs. Finally, another output of the task-level verification are the partitions CPU time budgets required per frame E_j^k, computed as explained in the previous section.

Dependences. Let $\tau_i \in \mathcal{T}$ be a task of the software platform. J_i depends on the precedence relations and communications τ_i is involved in. In fact, the predefined dependences of the system constraints the value of J_i, which must be taken into account when computing the WCRT of τ_i.

If τ_i does not have any predecessor nor receives any message at runtime, its jitter is null.

$$\forall \tau_i \in \mathcal{T} \mid \left(\sum_{j=1}^{N_T} msg_{ji} + prec_{ji} = 0 \right), J_i = 0 \tag{5.48}$$

On the contrary, if τ_i has some predecessor, its first activation can happen only after the end of the first execution of all its predecessors, as defined by matrix *prec*. In the one-to-one integration strategy, this corresponds to verifying the following relation:

$$J_i \geqslant \max_{k \in [1;N_T] \setminus \{i\} \mid prec_{ki} \neq 0} (J_k + w_k) \tag{5.49}$$

The analog relation is defined in the one-to-all integration strategy, by replacing N_T by $\#\mathcal{T}^{as_a}$ for each supplier $as_a \in AS$.

Similarly, the first activation of τ_i can happen only after the end of the first execution of all the tasks it is receiving messages from, which implies the following relation to be verified in addition to equation (5.49) if τ_i has some predecessors. In the one-to-one integration strategy, this corresponds to verifying the following relation:

$$J_i \geqslant \max_{k \in [1;N_T] \setminus \{i\} | msg_{ki} \neq 0} (J_k + w_k) \tag{5.50}$$

The analog relation is defined in the one-to-all integration strategy, by replacing N_T by $\#\mathcal{T}^{as_a}$ for each supplier $as_a \in AS$.

It is important to note that equations (5.48), (5.49) and (5.50) concern intra-application precedence and communications since matrices *msg* and *prec* are expressed by each application supplier and only mention the partitions they are respectively in charge of designing. The verification of inter-partition precedences and communications is part of the partition-level verification performed during the allocation. As will be explained in the next subsections, such a verification will be done using matrices *ipc* and *pPrec* instead of *msg* and *prec* respectively.

Finally, it is important to note that in the feasibility analysis, the jitter upon first activation of the tasks targets only the first activation of each task; as such, equations (5.49) and (5.50) hold even in cases of multiperiodic precedence relations or communications.

Tasks Schedulability. Once jitters upon first activation and WCRTs are computed, the feasibility of the allocation under study is assessed by evaluating whether all tasks are schedulable or not. The software/hardware allocation can be considered as a valid allocation if all tasks are able to respect their deadlines in the worst-case situation represented by R_i, i.e. if the following relation is satisfied by all tasks of the software platform:

$$\forall \tau_i \in \mathcal{T},\ R_i \leqslant D_i \tag{5.51}$$

If it is not the case, the allocation considered as input to the analysis is rejected as an invalid configuration that cannot be used for the system to be designed.

Partition-Level Timing Verification during the Allocation

Partition-level verification conducted during the allocation phase targets all dependences between different partitions or applications on the same multicore.

The partition-level timing verification during the allocation consists in verifying that, for a given allocation, it will always be possible to find a schedule where:

- All CPU time budgets of a partition π_i guarantees enough time to each of its tasks according to their respective WCRTs, while meeting their periodicity requirement P_i;
- All partitions are able to satisfy such a first requirement during one MAF;
- All partition-level dependences defined in the software platform are respected. The first two requirements are about the feasibility of the partitions, while the last requirement is about dependences enforcement.

In order to perform such a verification, the MAF must be computed using all software applications partitions periods information. On the other hand, all partitions CPU time budgets per frame must be computed using their respective tasks WCRTs – which are computed during the task-level timing verification of the allocation. As such, the allocation and the tasks WCRTs are inputs to the partition-level timing analysis, along with the partitions and tasks characteristics; the MAF and the partitions CPU time budgets per frame E_i^k are outputs of the analysis.

The MAF is computed using equation (4.19) for each core in the one-to-one integration strategy, and equation (4.15) in the one-to-all integration strategy. Each partition CPU time budget per frame E_i^k is computed using equation (5.45) in the one-to-one integration strategy, and equation (5.47) in the one-to-all integration strategy.

Dependences. At partition-level, the verification of inter-partition dependences must be done. In particular, partitions jitters must be defined so that the first window of a partition occurs later than the first window of all its predecessors in the MAF schedule .

Let $\pi_i \in \mathcal{P}$ be a partition of the software platform. If π_i has some predecessors, its jitter must verify the following relation:

$$pJ_i \geqslant \max_{\substack{k \in [1;N_P]\setminus\{i\} \\ pPrec_{ki} \neq 0}} \left(pJ_k + E_k^1\right) \tag{5.52}$$

Indeed, pJ_i defining the jitter of the first window of π_i, it must be defined such that the first window of π_i occurs only after the first window of all its predecessors π_k, represented by their jitters pJ_k and their first budgets E_k^1 respectively.

The analog relation can be derived if π_i is involved in inter-partition communications:

$$pJ_i \geqslant \max_{k \in [1;N_P] \ \{i\} | ipc_{ki} \neq 0} \left(pJ_k + E_k^1\right) \tag{5.53}$$

Finally, pJ_i respects both equations (5.52) and (5.53). This is true for both integration strategies proposed in this thesis. These equations can be merged into the final definition of pJ_i as follows:

$$pJ_i \geqslant \max_{k \in [1;N_P] \ \{i\} | ((pPrec_{ki} \neq 0) \vee (ipc_{ki} \neq 0))} \left(pJ_k + E_k^1\right) \tag{5.54}$$

Task-Level Information Transfer to the Partition-Level Partitions jitter upon first activation pJ_i enable to take into account, at partition level, the existence of jitters for each task first activation inside its partition respectively. However, pJ_i covers only the first window of each partition. Aside from the first window, one must also take into account the fact that a task will not have exclusive access to the computing resource it is running on, but is rather limited to its partition CPU time windows. Partition windows being monolithic, one must make sure it will always be possible to define window start dates that are compatible with all tasks execution – occurring in that partition window – starting and ending in the same window.

This verification can be done using the parameter $latest_i^k$: $latest_i^k$ is the latest start date of the k^{th} window of π_i for which all tasks of π_i are guaranteed to start and end their execution in the k^{th} window. To do so, $latest_i^k$ is computed using task-level information as follows:

$$\begin{aligned} &\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall k \in [1; nFrames_i], \\ &latest_i^k = \min(k \times P_i - E_i^k, \min_{\substack{\tau_j \in \mathcal{T}^{as_a} \\ pid_j = i \\ (k-1) \times P_i = (k-1) \times T_j}} (k \times T_j - \sum_{p=1}^{N_C} na_{pj} \times w_j)) \end{aligned} \tag{5.55}$$

Once $latest_i^k$ is computed for each partition and window, one must make sure that each $latest_i^k$ parameter value matches its domain definition, i.e. it value actually corresponds to the k^{th} activation of π_i. This verification can be done by verifying that the following equation is true for all partitions:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall k \in [1; nFrames_i], \quad (k-1) \times P_i \leqslant latest_i^k \leqslant k \times P_i \tag{5.56}$$

Partitions Schedulability. Once each E_i^k has been computed according to the WCRTs of the tasks of π_i, one must verify that each budget E_i^k is compatible with the periodicity of π_i, i.e. that all frame budgets represent a smaller time interval than the corresponding partition period P_i. For the first window of π_i, this can be expressed as follows:

$$\max\left(latest_i^1,\ pJ_i\right) + E_i^k \leqslant P_i \tag{5.57}$$

The jitter pJ_i covers only the first window of π_i. In order to extend the verification to all windows of π_i during the allocation feasibility analysis, one must also check whether $latest_i^k$ actually corresponds to a valid start date for the k^{th} window of π_i for each partition. This is done by checking whether the following relation is true:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall k \in [1; nFrames_i], \ \ latest_i^k + E_i^k \leqslant k \times P_i \tag{5.58}$$

It is important to note that equations (5.57) and (5.58) correspond to a verification that partition π_i has been designed in a realistic manner for multicore environments. Indeed, if one of these two relations is not respected, it means the sum of the durations of all tasks of π_i to be scheduled in the k^{th} window of π_i is bigger than the window duration. The cause of such an ill-design may be the underestimation of inter-task interference in multicore environments, which are in fact impossible to predict in advance.

The thesis contributions cope with this problem by safely upper-bounding inter-task interference through functions $d_{RAM}()$ and d_{INT} presented earlier in this chapter. Thanks to these two functions, tasks WCRTs and WCETs do include interference considerations.

Since partitions CPU time budgets are computed based on their respective tasks WCETs and WCRTs, the partition-level schedule is safe in the sense that partitions time windows have been dimensioned to provide enough CPU time to all their tasks at runtime even in the worst-case scenario of interference. As a conclusion, the WCRT computational approach, along with the timing analysis proposed in this thesis, enable to get rid of the risk of defining ill-designed partition schedules relying on underestimated time budgets.

Finally, the last requirement to be verified is the fact that all partitions budgets must fit in one MAF, which therefore depends on which strategy is considered.

In the one-to-all integration strategy, one must make sure the entire MAF schedule must be covered, and more specifically, take into account the possibility to have multiple repetitions or a given partition cycle inside one MAF, as illustrated in figure 4.7 page 105. To do so, the verification whether all CPU time budgets fit in one MAF can be performed by verifying that the following equation holds:

$$\sum_{m=1}^{N_P} \left(\frac{gMAF}{nFrames_m \times P_i} \times \left(\sum_{k=1}^{nFrames_m} E_m^k \right) \right) \leqslant gMAF \tag{5.59}$$

where $\frac{gMAF}{nFrames_m \times P_i}$ is the number of repetitions of π_m cycle inside one MAF, i.e. the number of repetitions of each budget E_i^k per MAF in the partition schedule, and $\left(\sum_{k=1}^{nFrames_m} E_m^k \right)$ is the total CPU time budget required for π_m per partition cycle of π_m.

In the one-to-one integration strategy, there is one MAF per core. As such, an analog verification must be done for each core:

$$\forall p \in [1; N_C], \sum_{m=1}^{N_P} a_{pm} \times \left(\frac{MAF_p}{nFrames_m \times P_i} \times \left(\sum_{k=1}^{nFrames_m} E_m^k \right) \right) \leqslant MAF_p \tag{5.60}$$

5.5 Scheduling and Timing-Related Verification

In this section, we present the timing-related verifications that must be performed in the scheduling problem. The schedule verification phase can be divided into task-level verification and partition-level verification. The task and partition models used are the ones for the schedule generation as defined in chapter 4. Finally, unless explicitly stated, all equations presented in this section are valid in both the one-to-one and the one-to-all integration strategies.

Task-Level Timing Analysis

The task level schedulability analysis consists in verifying if a given task schedule respects all timing requirements of the system for a given partition-level schedule, since tasks can only execute inside the time windows of their respective partitions. This implies partitions time windows to have been set using activation dates pO_i^k and the corresponding time budget $E_i^{k'}$, which are all inputs to the analysis.

The parameters tO_i^k are inputs as well, with a different signification depending on whether tasks are preemptive or not: as explained in subsection 5.1.2, each tO_i^k parameter corresponds to the actual start date of the task instance τ_i^k in the case of a non-preemptive task-level environment, and the jitter upon first activation of τ_i^k in a preemptive task-level environment.

In the rest of this chapter, unless specifically stated otherwise, explanations and equations are given in the context of non-preemptive setups but they are directly transposable – and thus applicable – to preemptive setups without modifications. In particular, the tO_i^k parameters are referred to as start dates as is the case in non-preemptive setups for simplicity of the explanations.

Finally, the outputs that must be computed during the analysis are the task instances WCETs w_i^k using equation (5.9) if τ_i is non-preemptive, equation (5.11) if τ_i is preemptive.

Tasks Scheduling. Once all w_i^k have been computed, the following relation must be verified to ensure that all task instances are able to respect their deadlines in the corresponding task schedule:

$$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i], tO_i^k + w_i^k \leqslant k \times D_i \tag{5.61}$$

One must also verify that tasks execution occurs within the boundaries of their respective partitions time windows. Let π_j be a partition of the partition set $\mathcal{P}$ and τ_i a task belonging to π_j. Then all executions of τ_i must occur within the boundaries of one of the time windows allocated to π_j in the MAF schedule.

In order to verify such a requirement, one must be able to identify in which time window a task execution instance will be scheduled. It is possible to do so thanks to equation (4.14) defined page 106. The following relation can then be expressed in order to verify that all tasks executions occur within the boundaries of their respective partitions time windows:

$$\begin{aligned}&\forall \pi_j \in \mathcal{P}, \forall \tau_i \in \mathcal{T} \mid (pid_i = j) \wedge \left(\exists \tilde{k} \in [1; nbActiv_i], \mid (\tilde{k}-1)T_i = (k-1) \times P_j\right), \\ &pO_j^k \ \leqslant \ tO_i^{\tilde{k}} \ \leqslant \ pO_j^k + E_j^{k\%nFrames_j}\end{aligned} \tag{5.62}$$

Analogously, the end of execution of all task instances of the k^{th} window allocated to π_j of π_j must occur within the boundaries of the corresponding time window.

$$\begin{aligned}&\forall \pi_j \in \mathcal{P}, \forall \tau_i \in \mathcal{T} \mid (pid_i = j) \wedge \left(\exists \tilde{k} \in [1; nbActiv_i], \mid (\tilde{k}-1)T_i = (k-1) \times P_j\right), \\ &pO_j^k \ \leqslant \ tO_i^{\tilde{k}} + w_i^{\tilde{k}} \leqslant pO_j^k + E_j^{k\%nFrames_j}\end{aligned} \tag{5.63}$$

Dependences. In the feasibility analysis, dependence relations – either due to simple precedence relations or message passing – influence tasks jitter upon first activation. In the schedulability analysis, these dependences have an impact on the choice of activation dates, not only for the first but rather all instances of tasks inside one MAF.

We denote $\tau_i \rightarrow \tau_j$ as the precedence relation stating that τ_i is a predecessor of τ_j. If a dependence is defined for a couple of tasks $(\tau_i,\ \tau_j)$ which have the same period, then the dependence relation should be reflected on each couple of instances $(\tau_i^k,\ \tau_j^k)$ for all corresponding frames k of the MAF. If on the other hand, τ_i and τ_j have different periods, the relation to express is a bit less straightforward. In the next paragraphs we explain in details how to define the corresponding relation for simple precedence, before deriving the corresponding relations for message communications. Eventually, it is important to note that only intra-partition dependences are considered in the task-level schedulability analysis.

In both strategies, if τ_i and τ_j are involved in a precedence relation $\tau_i \rightarrow \tau_j$ and have the same period, then they have the same number of activations per MAF and the precedence relation must be enforced for each couple (τ_i^k, τ_j^k):

$$\forall (\tau_i, \tau_j) \in \mathcal{T}^2 \mid (T_j = T_i) \wedge (prec_{ij} \neq 0), \ \ tO_j^k \geqslant tO_i^k + w_i^k \tag{5.64}$$

A similar equation can be expressed for message-based communications between equiperiodic tasks:

$$\forall (\tau_i, \tau_j) \in \mathcal{T}^2 \mid (T_j = T_i) \wedge (msg_{ij} \neq 0), \ \ tO_j^k \geqslant tO_i^k + w_i^k \tag{5.65}$$

If the precedence relation $\tau_i \rightarrow \tau_j$ involves two tasks having different periods, then the dependence relation between the instances of τ_j and τ_i must be expressed for instances occurring in the same frame, i.e. between instances τ_i^k and τ_j^l where the indexes k and l correspond to the same frame in the MAF schedule. Two situations can occur depending on the two tasks periods.

If τ_j is slower than τ_i, i.e. if $T_j > T_i$, then τ_j has less instances scheduled in one MAF than τ_i. The precedence relation must be expressed for all instances of τ_j to make sure each instance of τ_j is scheduled after the corresponding instance of τ_i. To do so, the precedence relation is expressed as follows:

$$\begin{aligned}&\forall \tau_i, \tau_j \in \mathcal{T} \mid (prec_{ij} = 1) \wedge (T_i < T_j), \forall l \in [1; nbActiv_j],\\ &\forall k \in [1; nbActiv_i] \mid (k-1)T_i = (l-1)T_j, \;\; tO_j^l \geqslant tO_i^k + w_i^k\end{aligned} \tag{5.66}$$

The condition $(k-1)T_i = (l-1)T_j$ enables to match the task instance numbers, i.e. target couples of instances τ_i^k and τ_j^l corresponding to the same frame in the MAF schedule, by getting advantage of the fact that tasks periods are harmonic and therefore even multiples of each other.

If τ_j is faster than τ_i, i.e. if $T_j < T_i$, then τ_j has more instances scheduled in one MAF than τ_i. In this case, the precedence relation must be expressed for all instances of τ_i to make sure each instance of τ_i is scheduled before the corresponding instance of τ_j. To do so, the precedence relation is expressed as follows:

$$\begin{aligned}&\forall \tau_i, \tau_j \in \mathcal{T} \mid (prec_{ij} = 1) \wedge (T_i > T_j), \forall k \in [1; nbActiv_i],\\ &\forall l \in [1; nbActiv_j] \mid (k-1)T_i = (l-1)T_j, \;\; tO_j^l \geqslant tO_i^k + w_i^k\end{aligned} \tag{5.67}$$

Similar relations can be defined for message-based communications. If $T_j > T_i$, the following relation must be enforced:

$$\begin{aligned}&\forall \tau_i, \tau_j \in \mathcal{T} \mid (msg_{ij} = 1) \wedge (T_i < T_j), \forall l \in [1; nbActiv_j],\\ &\forall k \in [1; nbActiv_i] \mid (k-1)T_i = (l-1)T_j, \;\; tO_j^l \geqslant tO_i^k + w_i^k\end{aligned} \tag{5.68}$$

If $T_j < T_i$, the following relation must be enforced:

$$\begin{aligned}&\forall \tau_i, \tau_j \in \mathcal{T} \mid (msg_{ij} = 1) \wedge (T_i > T_j), \forall k \in [1; nbActiv_i],\\ &\forall l \in [1; nbActiv_j] \mid (k-1)T_i = (l-1)T_j, \;\; tO_j^l \geqslant tO_i^k + w_i^k\end{aligned} \tag{5.69}$$

Partition-Level Timing Analysis

The partition-level schedulability analysis is performed after a partition schedule has been built. The goal is to verify that the computed partition-level schedule respects all timing requirements of the corresponding partitions.

The inputs are a software/hardware allocation, and the partition schedule, i.e. activation dates pO_i^k and the partitions CPU time budgets per frame E_i^k. The output is the result of the verification of the satisfaction of all partitions timing requirements.

Partitions Scheduling. Once each E_i^k is computed, the partition-level schedulability analysis then consists in verifying that all time windows are in line with the corresponding partition periodicity. It is indeed the case if the following relation is always verified, for each partition π_i and each frame k of the MAF schedule, independently of the considered strategy:

$$\forall \pi_i \in \mathcal{P}, \forall k \in [1; nFrames_i], \quad pO_i^k + E_i^k \leqslant k \times P_i \tag{5.70}$$

In the one-to-all integration strategy, an additional verification must be performed. Indeed, as explained in subsection 5.4, a partition start date pO_i^k must not be bigger than the corresponding latest start date $latest_i^k$ defined according to task-level information:

$$\forall \pi_i \in \mathcal{P}, \forall k \in [1; nFrames_i], \quad pO_i^k \leqslant latest_i^k \tag{5.71}$$

Dependences. In the one-to-one integration strategy, no additional partition-level verification of dependences is necessary, since they are all expressed using the task-level matrices, without differentiating intra-partition dependences from inter-partition dependences.

In the one-to-all integration strategy however, one needs to verify all inter-partition dependences since they are not covered by the matrices representing task-level dependences. To do so, similar verification must be performed, replacing task-level matrices *prec* and *msg* by partition-level matrices *pPrec* and *ipc* respectively. The same differentiation of the situation depending on partitions periods holds as well.

In the case of equiperiodic partitions, equations (5.72) and (5.73) must be enforced.

$$\forall k \in [1; nFrames], \forall \pi_i, \pi_j \in \mathcal{P} \mid pPrec_{ij} = 1, \quad pO_j^k \geqslant pO_i^k + E_i^{k\%nFrames_i} \tag{5.72}$$

$$\forall k \in [1; nFrames], \forall \pi_i, \pi_j \in \mathcal{P} \mid ipc_{ij} = 1, \quad pO_j^k \geqslant pO_i^k + E_i^{k\%nFrames_i} \tag{5.73}$$

If π_i must be scheduled before π_j and $P_j > P_i$, equations (5.74) and (5.75) must be verified.

$$\begin{aligned} &\forall \pi_i, \pi_j \in \mathcal{P} \mid (pPrec_{ij} = 1) \wedge (P_i < P_j), \forall l \in [1; nFrames_j], \\ &\forall k \in [1; nFrames_i] \mid (k-1)P_i = (l-1)P_j, \quad pO_j^l \geqslant pO_i^k + E_i^{k\%nFrames_i} \end{aligned} \tag{5.74}$$

$$\begin{aligned} &\forall \pi_i, \pi_j \in \mathcal{P} \mid (ipc_{ij} = 1) \wedge (P_i < P_j), \forall l \in [1; nFrames_j], \\ &\forall k \in [1; nFrames_i] \mid (k-1)P_i = (l-1)P_j, \quad pO_j^l \geqslant pO_i^k + E_i^{k\%nFrames_i} \end{aligned} \tag{5.75}$$

Finally, if π_i must be scheduled before π_j and $P_j < P_i$, equations (5.76) and (5.77) must be verified.

$$\begin{aligned} &\forall \pi_i, \pi_j \in \mathcal{P} \mid (pPrec_{ij} = 1) \wedge (P_i > P_j), \forall k \in [1; nFrames_i], \\ &\forall l \in [1; nFrames_j] \mid (k-1)P_i = (l-1)P_j, \quad pO_j^l \geqslant pO_i^k + E_i^{k\%nFrames_i} \end{aligned} \tag{5.76}$$

$$\begin{aligned} &\forall \pi_i, \pi_j \in \mathcal{P} \mid (ipc_{ij} = 1) \wedge (P_i > P_j), \forall k \in [1; nFrames_i], \\ &\forall l \in [1; nFrames_j] \mid (k-1)P_i = (l-1)P_j, \quad pO_j^l \geqslant pO_i^k + E_i^{k\%nFrames_i} \end{aligned} \tag{5.77}$$

5.6 Discussions

Partitions CPU Time Budgets: Computed during the Allocation Search, Exploited for the Schedule Generation. Section 5.3 presents the equations exploited to compute each partition CPU time budget per frame E_i^k in the two integration strategies respectively. They are computed using tasks WCETs w_i exploited in the allocation problem, no equivalent definition updating the E_i^k parameters in the schedule generation problem is proposed. The reason behind such a choice depends on the considered strategy. In the one-to-one integration strategy, the E_i^k parameters are computed for verification purposes only, every decision being based on the task level in that strategy.

In the one-to-all strategy, the E_i^k parameters are exploited for verification, allocation and schedule generation purposes, which means they must already be computed before the schedule generation problem. Indeed, the E_i^k parameters are inputs of the schedule generation problem, which justifies their computation in the allocation problem, addressed before the schedule problem. One could recompute the E_i^k parameters during scheduling in order to use task instances WCETs w_i^k instead of one task WCET for all instances w_i, which would be likely to tighten the CPU budgets bounds. However, doing so appears to be useless in the one-to-all strategy except for optimization purposes in order to recompute a new schedule. Even so, all w_i^k may change once a new schedule has been defined, and partitions budgets will have to be recomputed as well.

Boolean Matrices versus Integer Vectors. As already observed in the "Discussions" section of chapter 4, it is more beneficial for each CP problem solving to define boolean matrices over vectors of integers for some parameters of our model.

An illustration of such a statement can be identified in the construction of the memory interference function d_{RAM} (see section 5.2), in particular with the usage of matrix *shared.* The model needs to identify which cores share some memory areas and which do not, depending on the core and path allocation. It is easy to do so with a boolean matrix, where, as currently defined, the corresponding $shared_{pq}$ term is either equal to one if cores p and q share some memory area, and zero if they do not.

It is therefore easy to transpose such information into the equations requiring such distinction, by adding the terms $shared_{pq}$ and $(1 - shared_{pq})$ in front of all parts of the equations that are respectively applicable only to cores sharing and not sharing memory areas. If *shared* were replaced with a vector of integers, one would have needed to express equations with conditions on the corresponding integers. However, it would have not been possible to define such equations in a CP in Cplex [6], which do not allow test conditions to rely on variables of the expressed CP; one would then have had to find some alternative, probably leading to more equations and/or more intermediary variables to be defined in the CP, therefore uselessly increasing the complexity of the corresponding CP.

Path Identification and Memory Interference. The fact that equations have been updated with $isSharingMC$ leads to the production of tighter bounds. Without such update, the bounds are perfectly valid, but contains some pessimism. Indeed, cores can share paths without sharing memory space, and they are thus likely to interfere at runtime. On the other hand, cores that do not share path to the memory or memory space will never try to access the same memory path, and as such, no memory interference as defined in this thesis: in equations (5.13) and (5.17) which are not updated with path sharing identification, such cores as still taken into account in the computation of RD_{inter}, and consequently, in the computation of the DRAM interference of tasks that will never be interfering together. This adds a positive integer to the computed WCET bound corresponding to unnecessary pessimism.

5.7 Summary

In this chapter, we presented how we constructed the timing analysis metrics necessary during software/hardware integration. To our knowledge, the proposed metrics are the first to be adapted for multicore since they are interference-aware, but also to IMA since specific care for the partition level has been taken.

The first timing analysis is a sufficient condition for feasibility of a multicore based IMA system, and is exploited when guiding the software/hardware allocation exploration search. The second timing analysis metric is a schedulability analysis to verify that all timing requirements of the systems are enforced in a given schedule. It is used once the allocation has been done, when computing activation dates to build a static schedule for one MAF of the system.

This chapter presented each analysis metric. In particular, the chapter includes an extended description of the interference computational models we implemented to bound inter-task interference. The equations corresponding to the timing verification to be performed when analysing the timing properties of a systems are also given in this chapter, but not yet positioned with regards to the industrial system design process, which will rather be done in the next chapter.

Chapter 6

IMA System Integration

This chapter presents the integration strategies proposed in this thesis.

For each strategy, we first give a general overview in chronological order of the activities constituting the proposed integration process. We then describe each step in greater details. In order to help automate the activities involved in each integration strategies, we propose to cover every step separately using constraint programming (CP). The resulting CP formulation defined for each step will be presented in a dedicated subsection describing the corresponding step.

6.1 One-to-All Integration Strategy

Strategy Overview

This section describes in detail the one-to-all integration strategy proposed in this thesis. As illustrated in figure 6.1, this strategy is divided into three steps. We describe them in chronological order and specify which role is in charge of which activities and during which step of the integration process.

The one-to-all integration strategy consists in the following steps, also summarized in figure 6.1:

- **Step AS1 - SW/HW allocation:** This step is performed by each application supplier, alone on its own applications.

  During step AS1, each application supplier must decide, for each partition separately, on which core to allocate the tasks of its partitions. Each supplier must do so while ensuring the selected allocation is safe, for instance by verifying that no task will overrun its deadline at runtime despite interference.

  To automate this allocation activity and ease the timing analysis to be performed to ensure the validity of the chosen allocation, we propose a CP formulation, named **AS11cp**. This CP will be presented in details later in the subsection detailing the content of step AS1.

  The result of step AS1 is a task-to-core allocation, for all partition designed by the corresponding application supplier.

- **Step MI1 - Global allocation verification and partition-level schedule generation:** This step is performed by the module integrator, who receives feedback from each supplier on the allocations chosen during step AS1.

  During step MI1, the module integrator is in charge of verifying the global allocation, i.e. verifying the existence of a feasible schedule with all partitions according to information on each partition provided separately by each supplier. If the verification results in the negation of existence of a valid schedule, the integrator discusses with some or all suppliers in order to negotiate some changes to the configuration choices made in step AS1. If on the

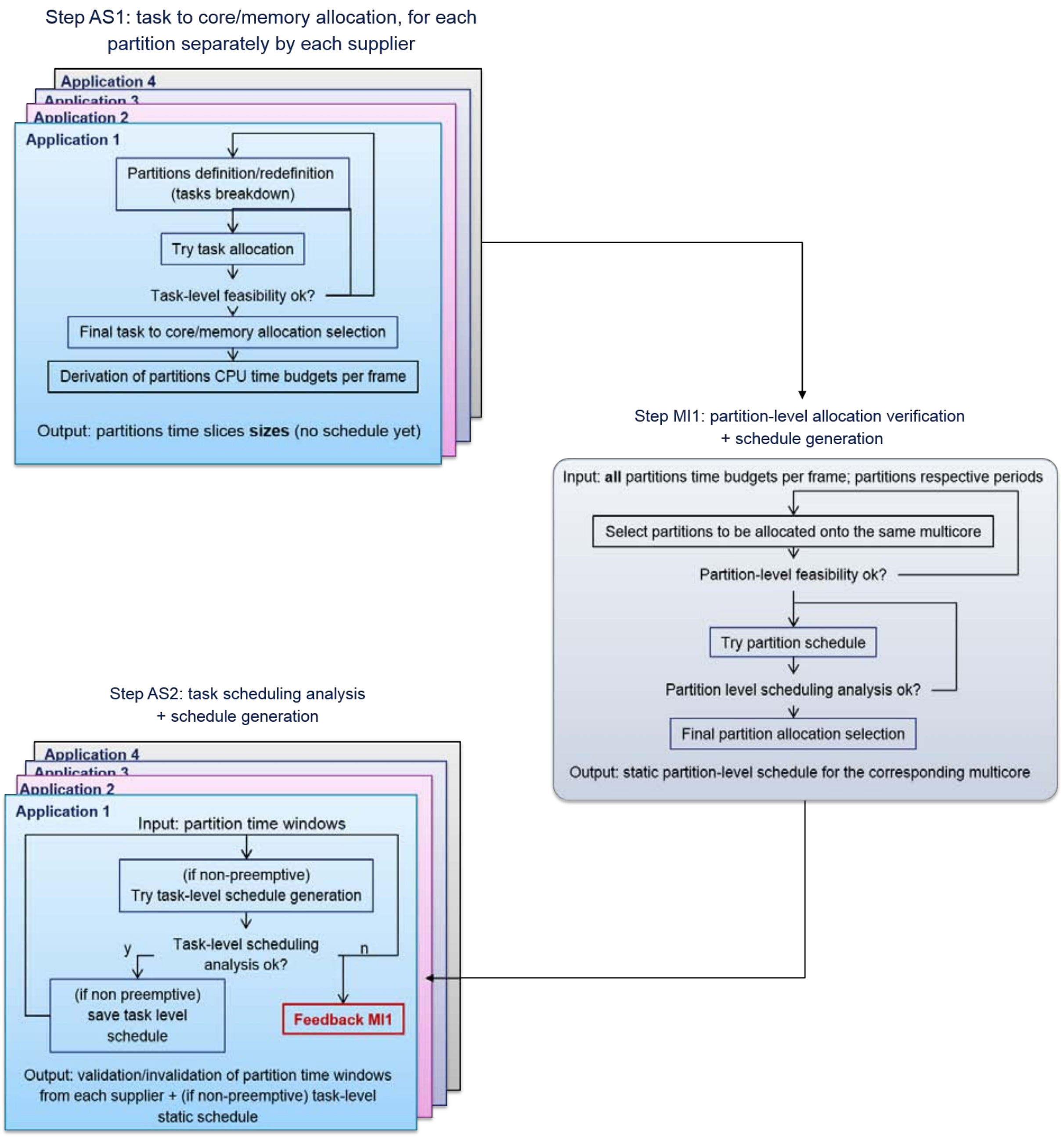

Step AS1: task to core/memory allocation, for each partition separately by each supplier
Application 4
Application 3
Application 2
Application 1
Partitions definition/redefinition (tasks breakdown)
Try task allocation
Task-level feasibility ok?
Final task to core/memory allocation selection
Derivation of partitions CPU time budgets per frame
Output: partitions time slices sizes (no schedule yet)
Step MI1: partition-level allocation verification + schedule generation
Input: all partitions time budgets per frame; partitions respective periods
Select partitions to be allocated onto the same multicore
Partition-level feasibility ok?
Try partition schedule
Partition level scheduling analysis ok?
Final partition allocation selection
Output: static partition-level schedule for the corresponding multicore
Step AS2: task scheduling analysis + schedule generation
Application 4
Application 3
Application 2
Application 1
Input: partition time windows
(if non-preemptive) Try task-level schedule generation
Task-level scheduling analysis ok?
y
n
(if non preemptive) save task level schedule
Feedback MI1
Output: validation/invalidation of partition time windows from each supplier + (if non-preemptive) task-level static schedule

Figure 6.1: Steps of the One-to-All Integration Strategy

contrary the verification shows the existence of a valid schedule, the corresponding partition allocation is considered valid, and the module integrator can proceed with building a partition schedule. In order to automate the verification and configuration choices to be done in step MI1, we propose a CP formulation, named **MI1cp**; this CP will be presented in details later in the subsection detailing the content of step MI1.

The output of step MI1 is a partition-level schedule, i.e. activation dates and time slots constituting the time windows of each partition in one MAF schedule, of duration $gMAF$ (cf. figure 4.6).

At the end of step MI1, the module integrator communicates, to each application supplier separately, the time windows of the partitions they designed respectively, so that they can verify whether their applications will always be able to respect their timing requirements at runtime given their configured, respective time windows.

- **Step AS2 - Local schedule verification:** This step is performed by each supplier separately, on their own partitions.

During step AS2, each supplier is in charge of verifying the behavior of their partitions at runtime by checking whether the tasks inside each partition will always be able to respect their timing requirements, given that they must be scheduled within the boundaries of their respective partitions time windows. If the verification fails for some partitions, the corresponding suppliers communicate with the integrator in order to negotiate some changes to the partition schedule built in step MI1 or the allocation choices made in step AS1.

In order to automate the verification and configuration choices to be done in step AS2, we propose a CP formulation, named **AS2cp**; this CP will be presented in details later in the subsection detailing the content of step AS2.

By the end of step AS2, if all partitions schedules respectively lead to the existence of valid task-level schedules, then the integration process ends successfully with a valid partition schedule for the corresponding module.

In the rest of this section, we describe in detail each of these steps, and their corresponding CP formulation. Each subsection describes one of the steps of the one-to-all integration strategy, including the content of the corresponding CP proposed.

Step AS1: SW/HW Allocation and Verification

This step is performed by each supplier on their own and separately from each other. For every partition, each application supplier must select a task-to-core allocation, verify that the corresponding timing properties satisfy the system requirements, and derive the associated partitions CPU time budgets per frame. In particular, this implies the computation of tasks WCETs and partitions CPU time budgets per frame E_i^k.

We propose to automate step AS1 by expressing them in a CP called **AS1cp**. The goal of **AS1cp** is to find an allocation such that, inside each partition, all tasks are schedulable and all timing requirements of the partition are guaranteed to always be respected at runtime.

To do so, the inputs of the defined CP are the description of the software and hardware architectures. The outputs of the CP are a software/hardware allocation – represented by na and $t2mc$ along with the partitions CPU time budgets per frame E_i^k and the tasks WCETs w_i as outputs. The constraints of the CP correspond to the allocation choices, the allocation verification and the time budgets and WCETs computation (see listing of constraints for the allocation problem in chapter 4).

To solve **AS1cp**, the search algorithm exploited by the CP solver explores the possibilities of allocation combinations, and for each combination evaluated, check whether all expressed constraints are satisfied. If such a combination exists, it is considered to be a solution of the allocation problem, and will be stored temporarily by the solver until one of such solutions is finally selected at the end of the solving process of **AS1cp**. If no such combination exists, it means it is not possible to allocate the software platform onto the hardware platform while guaranteeing the existence of a safe schedule for the corresponding system according to the timing analysis presented in chapter 5. Some modification must then be undertaken on the software and/or the hardware platform until **AS1cp** is able to find a solution. For instance, this may involve deciding to decrease or increase the number of cores to be used, or deciding to integrate less partitions onto the same module.

We present here the CP formulation of the allocation problem proposed in this thesis, referred to as **AS1cp**.

Inputs of AS1cp The inputs to **AS1cp** are, relatively to each application supplier: the entire hardware model (see subsection "Hardware Architecture in chapter 4), the partitions periods

P_i, their respective tasks vectors (C_i, T_i, D_i, H_i, $prio_i$, $tRam_i$), and matrices $PART$, msg and $prec$.

Variables of AS1cp The decision variables are na and $t2mc$ for the software/hardware allocation, and w_i and J_i for the timing analysis. The auxiliary variables are R_i, and (E_i^1, .. $E_i^{nFrames_i}$), which are computed using the w_j variables by construction, but also pJ_i, derived from the tasks jitters, and $latest_i^k$, derived form the tasks deadlines. The equations defining these parameters respectively are embedded in **AS1cp** as constraints of the allocation problem.

Outputs of AS1cp The main output of **AS1cp** is a software/hardware allocation thanks to matrices na and $t2mc$, and an upper-bound on each partition CPU time budget per frame k, E_i^k, but also the values of the $pRam_i^k$ parameters. As mentioned earlier, the produced allocation is guaranteed to be valid since **AS1cp** embeds the timing-related analysis we proposed in this thesis for the allocation problem (see chapter 5).

As explained in chapter 4, it is important to note that, in matrix na, N_T is the total number of tasks of all partitions designed by the application supplier implementing the CP. Indeed, each supplier will run its own CP, therefore na has a different size depending on which application supplier is considered. Another important remark is about the fact that N_T accounts for all tasks of all partitions designed by the same application supplier, whereas the task-to-core allocation is always done separately for each partition; as such, all constraints of the CP will be expressed with clear references to the task-to-partition definition in order to allocate tasks to cores accordingly.

Core Allocation Constraints

This subsection presents all constraints expressed in **AS1cp** that are related to the allocation of the tasks to the cores of the multicore platform.

All tasks are allocated once Each task must be allocated exactly once. This is specified in **AS1cp** by defining the following constraint:

$$\forall as_a \in AS, \forall \pi_j \in \mathcal{P}^{as_a}, \forall \tau_i \mid (pid_i = j)\,, \quad \sum_{p=1}^{N_C} na_{pi} = 1 \tag{6.1}$$

No overloaded core In the allocation to be selected in step AS1, no core should be overloaded. This can be expressed by defining a constraint forcing each core utilization ratio to be under 100%. The corresponding constraint can then be expressed as follows:

$$\forall p \in [1; N_C], \quad \sum_{i=1}^{N_T} na_{pi} \times \frac{\left(C_i^p + C_{SW_p} + d_{RAM}(w_i, H_i^p) + d_{INT}(w_i, H_i^p)\right)}{T_i} \leqslant 1 \tag{6.2}$$

Memory Path Allocation Constraints

This subsection presents all constraints expressed in **AS1cp** that are related to the memory path allocation.

Each task inside each partition is allocated to at least one memory path If a task has not been allocated to any memory controller, it will not be able to access the main memory at runtime. To prevent such unrealistic allocations, the following constraint is expressed to ensure that all tasks use at least one memory controller:

$$\forall as_a \in AS, \forall \tau_i \in \mathcal{T}^{as_a}, \sum_{k=1}^{N_{MC}} t2mc_{ki} \geqslant 1 \tag{6.3}$$

All memory paths are used by each partition For one given partition, all memory controllers must be used in order to maximize as much as possible parallel memory accesses within each partition time window. One way to do so is to implement in **AS1cp** a constraint specifying that all memory controllers must be allocated to at least one of the tasks of the partition, and for all partitions:

$$\forall as_a \in AS, \forall \pi_j \in \mathcal{P}^{as_a}, \forall k \in [1; N_{MC}], \quad \sum_{\substack{i=1 \\ pid_i = j}}^{N_T} t2mc_{ki} \geqslant 1 \tag{6.4}$$

This constraint enables to balance the load by making sure that each partition benefits from the entire memory bandwidth at runtime.

Coherent Memory Path Allocation Another verification that must be performed in relation with the memory is the coherence of the memory paths allocation with regards to the core-to-memory physical paths, i.e. verifying that the tasks have all been allocated to memory paths that are actually wired to the core they each have been allocated to. This can be done by comparing the values of matrices *t2mc* and *c2mc* as follows:

$$\forall k \in [1; N_{MC}], \forall \tau_i \in \mathcal{T}, \left(\sum_{p=1}^{N_C} na_{pi} \times c2mc_{kp} = 0 \right) \implies (t2mc_{ki} = 0) \tag{6.5}$$

Indeed, the first sum enables to retrieve the index of the core p to which τ_i has been allocated, corresponding to the only non-null term na_{pi} of the sum. Multiplying this non-nul term by $c2mc_{kp}$ for a given memory controller k enables to verify whether core p is physically wired to the controller k: the total sum is non-null if it is indeed the case, and null otherwise. And if core p is not wired to the memory controller k, then any task τ_i allocated to core p cannot be allocated to the memory controller k, i.e. $t2mc_{ki}$ can only be equal to zero.

In addition, in the one-to-all integration strategy, it is important to also verify that the memory path allocation corresponds to a realistic memory area allocation regarding the allocated path and the available memory space. This can be done in **AS1cp** by defining the following constraint:

$$\forall as_a \in AS, \forall \pi_m \in \mathcal{P}^{as_a}, \forall k \in [1; N_{MC}], \quad \sum_{\substack{i=1 \\ pid_i = m}}^{N_T} t2mc_{ki} \times tRam_i \leqslant mcSize_k \tag{6.6}$$

Finally, **AS1cp** must also contain a constraint computing the memory context of each partition π_m per memory path k, $pRam_m^k$, for later use by the module integrator in step MI1 when they will check that all partitions allocated to a same module correspond to a realistic configuration in terms of memory space allocation. The corresponding constraint would be equation (4.12) presented in chapter 4.

Coherent memory path regarding tasks contexts The memory path allocation must be coherent with the memory architecture in the hardware platform considered, i.e. the memory context corresponding to the tasks allocated to a given path k must remain smaller or equal to the size of memory space addressable from path k:

$$\forall k \in [1; N_{MC}], \quad \sum_{i=1}^{N_T} t2mc_{ki} \times tRam_i \leqslant mcSize_k \tag{6.7}$$

Coherent memory path allocation regarding Message-Based Communications If two tasks are involved in some message passing scenario, then they both must be able to access the corresponding shared memory area. As explained before, we assume the corresponding area to be stored in a memory area that is accessible by producer and consumer tasks. As such, the constraint to be expressed must specify that the producer task must be able to access the same memory area than the consumer task. This corresponds to the following constraint:

$$\forall as_a \in AS, \forall \tau_i, \tau_j \in \mathcal{T}^{as_a}, (msg_{ji} + msg_{ij}) \neq 0 \implies \left(\sum_{k=1}^{N_{MC}} t2mc_{ki} \times t2mc_{kj} \geqslant 1 \right) \tag{6.8}$$

Indeed, if two tasks τ_i and τ_j exchange messages at runtime, they must have at least one memory path in common in order to be able to access the shared memory where the corresponding message will be stored. In this equation, the term $(msg_{ji} + msg_{ij})$ enables to select couples of tasks (τ_i, τ_j) that are involved in message-based communications. For such couples, the tasks must have at least one memory path in common, i.e. there exists at least one memory path index k for which the term $t2mc_{ki} \times t2mc_{kj}$ is not null.

Schedulability Constraints

This subsection presents all constraints expressed in **AS1cp** that are related to the verification of timing-related properties of the evaluated allocations. Altogether, these constraints represent the feasibility analysis proposed in this thesis for early timing analysis and for guiding the allocation search in a safe manner by forcing the choice of allocation to correspond to an allocation that passes the feasibility test.

Partitions Time Budgets per Frame E_i^k Computation The feasibility analysis includes verifying that each partition time budget per frame can be provided within the period of the corresponding partition. This includes computing the partition time budgets per frame E_i^k first. To do so, equation (5.47) (see chapter 5) is embedded as a constraint in **AS1cp**.

Link between task-level and partition-level requirements: latest possible activation of partition window When configuring the SW/HW allocation, each supplier must verify that no frame is overloaded, i.e. that the CPU time budget required by every partition π_i every P_i time units actually fits a time interval of length P_i. If such a relation is not verified, it means that the corresponding partition periodicity is too fast for all its tasks to start and complete their executions normally.

Partitions periods are chosen with the knowledge of their tasks execution needs, but for single-core environments. Indeed, the occurrence of multicore interference suffered by tasks at runtime may be so important that a partition tasks would require a time window that actually is bigger than the partition period in order for all of them to complete their respective executions. **AS1cp** must be able to detect such situations in order to not select a SW/HW allocation leading to such a situation for any partition. Detecting such situations by exploiting **AS1cp** assists the corresponding supplier: if no valid allocation exists for a given partition because of a periodicity that is too fast, the supplier may want to increase the value of the period.

Partitions time budgets E_i^k have been computed using task-level information: indeed, as explained in the previous paragraph, they rely on tasks WCRTs, which include upper-bounds on multicore interference. In order to check whether it will be possible to find a partition-level schedule that matches tasks requirements, one must also be able to extract knowledge about tasks jitters and precedence relations which are constraining their possible activation dates. To do so, pJ_i and $latest_i^k$ have been defined for each partition π_i. They are computed using equations (5.54) and (5.55) respectively, which must therefore be expressed in **AS1cp** as constraints.

In addition, equation (5.56) is embedded in **AS1cp** as a constraint as well in order to verify that $latest_i^k$ matches its definition domain.

All partitions are schedulable As explained in details in chapter 5, equations (5.57) and (5.58) may be used in order to check whether there exists partitions time windows fitting their respective definition domain, i.e. that the k^{th} window of π_i can be scheduled within the boundaries of the k^{th} frame of the partition cycle.

All tasks are schedulable As mentioned at the beginning of the subsection, **AS1cp** embraces the task-level timing related verification proposed in chapter 5, which takes the form of a system of equation to be solved and an equation to be verified by all tasks in order to pass the test. As such: (5.7) is expressed as a constraint of **AS11cp** to compute tasks w_i and J_i. Equation (5.51) is expressed as well in order to verify feasibility of τ_i. Equations (5.48), (5.49) and (5.50) are embedded in **AS11cp** in order to compute tasks jitters upon first activation accordingly.

Finally, **AS1cp** must contain the verification that all tasks are able to complete every execution before their respective deadline. This is usually expressed using the following equation:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall \tau_j \in \mathcal{T}^{as_a} \mid (pid_j = i)\,, \quad J_j + w_j \leqslant D_j \tag{6.9}$$

However, equation (6.9) does not take into account a hierarchically constrained environment such as the partition level in IMA systems. In fact, just as a task τ_j may have some jitter upon first activation J_j that is non-null, its partition π_i may have one as well, pJ_i. The partition window defining the valid interval in which a task can be executed, partitions jitters must be taken into account when verifying the feasibility of tasks in IMA systems. As such, equation (6.9) will rather be replaced by the following equation in **AS1cp**:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall \tau_j \in \mathcal{T}^{as_a} \mid (pid_j = i)\,, \quad \max\left(pJ_i, J_j\right) + w_j \leqslant D_j \tag{6.10}$$

where $\max\left(pJ_i, J_j\right)$ enables to take into account both the task and its partition jitter.

Message passing and precedence relations As briefly mentioned in chapter 4, tasks jitter upon first activation J_i are computed according to the precedence and message relations between tasks as specified in the software architecture. If a task does not have any predecessor or does not receive any message at runtime, its jitter is null. Based on equation (5.48) expressed page 140, this relation is represented in **AS1cp** as follows:

$$\forall \tau_i \in \mathcal{T}, \left(\sum_{\substack{k=1 \\ k \neq i}}^{N_T} (msg_{ki} + prec_{ki}) = 0 \right) \implies (J_i = 0) \tag{6.11}$$

On the contrary, if a task has predecessors or receives messages, its jitter upon activation must correspond to the end of execution of all its predecessors and/or message senders. Based on equation (5.49) expressed page 140, in order to cover such situations, the following constraint must be specified in **AS1cp**:

$$\forall \tau_i \in \mathcal{T}, \left(\sum_{\substack{k=1 \\ k \neq i}}^{N_T} (msg_{ki} + prec_{ki}) \neq 0 \right) \implies \left(J_i \geqslant \max_{\substack{\tau_k \in \mathcal{T} \\ msg_{ki} + prec_{ki} \neq 0}} (J_k + w_k) \right) \tag{6.12}$$

Affinities and exclusion constraints The task-to-core allocation must be in line with the task-to-core affinities expressed by the application supplier. In fact, the following relation must be verified if two tasks that must be allocated to the same core according to *taskCoreAff*:

$$\forall as_a \in AS, \forall \tau_i \in \mathcal{T}^{as_a}, \forall p \in [1; N_C], \left(taskCoreAff_{pi} \geqslant 1\right) \implies (na_{pi} = 1) \tag{6.13}$$

Similarly, the task-to-task affinities must be respected in the task-to-core allocation:

$$\forall as_a \in AS, \forall \tau_i, \tau_j \in \mathcal{T}^{as_a}, \left(taskAff_{ij} = 1\right) \implies \left(\sum_{p=1}^{N_C} na_{pi} \times na_{pj} = 1 \right) \tag{6.14}$$

Finally, the task-to-task exclusions expressed by the application supplier must be reflected in the selected task-to-core allocation:

$$\forall as_a \in AS, \forall \tau_i, \tau_j \in \mathcal{T}^{as_a}, (taskEx_{ij} = 1) \implies \left(\sum_{p=1}^{N_C} na_{pi} \times na_{pj} = 0 \right) \tag{6.15}$$

Objective Function

More than one valid solution to the problem expressed in **AS1cp** may exist. As such, we define additional objective functions to help classify the valid solutions into ”optimized solutions” and ”less optimized solutions”, according to optimization parameters such as workload reduction for instance. The optimization criteria is defined according to preferences of the system designers.

For instance, one preferred optimization criteria for the module integrator is the reduction of E_i^k budgets, as mentioned before. The corresponding objective function would be the following:

$$\text{Minimize} \max_{\pi_i \in \mathcal{P}} \left(\sum_{k=1}^{nFrames_i} E_i^k \right) \tag{6.16}$$

Where N_P stands for the total number of partitions defined by each application supplier.

Another optimization criteria that may suit system designers targets the workload reduction per core. According to the definition that can be found in the literature, the objective function minimizing the CPU workload for each core would be the following:

$$\text{Minimize} \sum_{p=1}^{N_C} na_{pj} \times \frac{C_j^p}{T_j} \tag{6.17}$$

However, although C_j^p is usually used to define tasks workloads, the values of the C_j^p parameters are constant integers in our model. Such an objective function is therefore of no use to the workload reduction. An alternative to the C_j^p parameters to model the contribution of τ_j to the CPU workload would be to define the workload depending on the tasks interference delays, which participation to the workload can often be far from negligible at runtime. To favor solutions in which interference are reduced, we therefore define the workload according to interference by implementing the following objective function:

$$\text{Minimize} \sum_{p=1}^{N_C} na_{pj} \times \frac{C_j^p + d_{RAM}(w_i, H_i^p) + d_{INT}(w_i, H_i^p)}{T_j} \tag{6.18}$$

where the general term of the sum enforces the minimization of the workload of each core p. The total sum therefore ensures that each CPU workload is minimized as much as possible by the CP solving process, as a sum of positive terms to be minimized.

According to equation (6.18), workload reduction actually relies on multicore interference reduction. Multicore interference being the variable part of a task WCET, and by construction, of a partition CPU time budget, this also means objective function defined via equation (6.18) addresses the objective expressed by equation (6.16) as well.

However, by construction of schedules in the one-to-all integration strategy (see figure 4.6), the actual parameter constraining the usage of each core is the partitions CPU time budgets rather than the tasks WCETs. WCETs can be reduced as much as possible, but the corresponding tasks poorly scheduled, which would lead to minimized inter-task interference, and at the same time, large partitions CPU time budgets, and therefore lead to lower integration ratio and thus under-optimized designs.

On the contrary, the objective of reducing as much as possible partitions CPU time budgets actually leads to reducing as much as possible tasks WCETs. Consequently, equation (6.16) may be the most appropriate objective function to be implemented in step AS1.

Step MI1: Global Allocation Verification and Partition-Level Schedule Generation

Step MI1 is performed by the module integrator, after step AS1 has been completed by every application supplier. The output of AS1 is a task-to-core allocation and the corresponding partitions CPU time budgets per frame. These outputs are provided by each supplier to the integrator, along with the corresponding partitions periods P_i and memory budgets per path $pRam_i^k$. Once the module integrator has all partitions time and memory budgets, they can compute the value of the MAF $gMAF$ and verify that the overall allocation is valid. If it is not the case, the integrator discusses with some or all the suppliers in order to negotiate changes to the configurations set in step AS1. If it is the case, the integrator proceeds with building a partition-level schedule of a duration of one MAF.

As illustrated in figure 4.7, to compute a schedule, the module integrator must decompose one MAF into time slots for each partition, according to their time budgets E_m^k.

$$k \equiv k^{'} \mod nFrames_m \tag{6.19}$$

To automate the activities performed in step MI1, we propose a CP formulation named **MI1cp**, in order to generate partitions activation dates pO_i^k for one MAF where all timing requirements of the systems are respected.

Inputs of MI1cp As explained before, step MI1 takes as input the outputs of step AS1, performed by each supplier on all applications to be integrated on the same multicore module. Moreover, step MI1 is performed by the module integrator. As such, the main inputs to **MI1cp** are partition-level information such as $pRam_i^k$, E_i^k and $latest_i^k$ P_i parameters.

Variables of MI1cp The variables of **MI1cp** are the partitions windows start dates pO_i^k, and the total number of windows implemented per partition π_i in the MAF schedule, $nbWindows_i$.

Outputs of MI1cp The main output of **MI1cp** is a partition schedule that has been verified at partition-level. As such, the outputs of **MI1cp** are the partitions windows activation dates pO_i^k, the total number of windows per partition $nbWindows_i$, but also the MAF $gMAF$ and MIF $gMIF$ on the system.

Finally, at the end of step MI1, the module integrator communicates, to each supplier separately, the time windows of their respective partitions so that each of them can perform further verifications on their own applications at task-level in step AS2.

Preprocessing and Integrated allocation verification

preprocessing phase of MI1 In step MI1, the module integrator receives – as inputs from each supplier – the E_i^k parameters of each partition in their respective cycle frames. Then, the module integrator computes the MIF and MAF of the corresponding system according to the partitions periods. This gives the total number of frames in one MAF, $nFrames$.

As illustrated in figure 4.7 page 105, the number of frames $nFrames$ in one MAF may be different from the number of frames in each partition cycle $nFrames_i$. One consequence is that the number of CPU time budget per frame E_i^k computed for each partition π_i is computed with $k \in [1; nFrames_i]$. In step MI1, the module integrator defines partition windows for each frame $k \in [1; nFrames]$, knowing that $nFrames$ is an even multiple of $nFrames_i$ and that partitions periods P_i are harmonic. Let $\tilde{E}_i^{\tilde{k}}$ be the CPU time budget computed by the application supplier as_a responsible of the design of π_i, with $\tilde{k} \in [1; nFrames_i]$. The first step prior to defining and launching the CP search corresponding to step MI1 is to derive the CPU time budgets per frame for each partition π_i for the duration of one complete MAF; let E_i^k denote such a time budget, with $k \in [1; nFrames]$. The relation between $\tilde{E}_i^k$ and E_i^k is the following:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall \tilde{k}, k \in [1; nFrames_i] \times [1; nFrames],$$
$$E_i^k = \begin{cases} \tilde{E}_i^{(\tilde{k}-1)\%nFrames_i+1} & \text{if } ((k-1) \times gMIF) = 0 \mod (P_i) \\ 0 & \text{otherwise.} \end{cases} \tag{6.20}$$

Similarly, let $\tilde{latest}_i^{\tilde{k}}$ denote the parameters computed by the supplier π_i with $\tilde{k} \in [1; nFrames_i]$, and $latest_i^k$ the corresponding parameters extended to one complete MAF with $k \in [1; nFrames]$. The relation between $\tilde{latest}_i^k$ and $latest_i^k$ is the following:

$$\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall \tilde{k}, k \in [1; nFrames_i] \times [1; nFrames],$$
$$latest_i^k = \begin{cases} \tilde{latest}_i^{(\tilde{k}-1)\%nFrames_i+1} & \text{if } ((k-1) \times gMIF) = 0 \mod (P_i) \\ 0 & \text{otherwise.} \end{cases} \tag{6.21}$$

At the beginning of step MI1, all computations involving any of the E_i^k and $latest_i^k$ parameters are performed on one MAF, i.e. with $k \in [1; nFrames]$. As such, the computations induced by equations (6.20) and (6.21) are done prior to all activities done in step MI1 and described in this subsection.

Memory Allocation Verification Another verification that must be done prior to generating a schedule, is the coherence of the memory allocation. In fact, the module integrator has to check that the memory allocation is realistic: it is the case if, when combining all partitions of a same module, the total memory footprint of partitions on a same core is smaller than or equal to the size of memory addressed by the corresponding core. More precisely, the size of memory addressed by each core according to the selected task-to-memory allocation is not bigger than the physically addressable memory area for each core. Such a property corresponds to the following relation:

$$\forall k \in [1; N_{MC}], \sum_{i=1}^{N_P} pRam_i^k \leqslant mcSize_k \tag{6.22}$$

Where $pRam_i^k$ is computed using equation (4.12). However, the module integrator is not allowed ot perform any task-level verification. As an alternative, the module integrator must work with the partitions memory budgets per path $pRam_i^k$, computed as outputs of **AS1cp**.

The following constraint must then be expressed in **MI1cp** to verify that the memory path allocation is realistic in terms of memory layout:

$$\forall k \in [1; N_{MC}], \sum_{i=1}^{N_P} pRam_i^k \leqslant mcSize_k \tag{6.23}$$

Schedule Generation Constraints

Partition Windows Start Dates In step MI1, the module integrator generates a partition-level schedule by setting the start date pO_i^k of each partition window. It is done automatically in **MI1cp** by defining pO_i^k as variables of the problem. To do so, a constraint must be expressed in **MI1cp** in order to check that pO_i^k is defined correctly. In fact, it is important to keep in mind the fact that although there are $nFrames_i$ CPU time budgets defined for π_i, there are $nFrames$ windows and start dates to be defined for π_i in one MAF. The index k of each variable pO_i^k is defined in the interval $[1; nFrames]$. Moreover, the MIF is always smaller than or equal to P_i for each partition. As such, if $gMIF$ is smaller than P_i for a given partition π_i, then there are more pO_i^k variables than there actually are windows for π_i in one MAF. As such in **MI1cp**, there should be a constraint setting to zero all pO_i^k variables for all indexes k that do not correspond to an actual window of π_i. This is done by expressing the following constraint in **MI1cp**:

$$\forall \pi_i, \forall k \in [1; nFrames], ((k-1) \times gMIF \neq 0 \mod (P_i)) \implies (pO_i^k = 0) \tag{6.24}$$

Another constraint must also be added in order to ensure that all other pO_i^k variables actually correspond to the start date of the k^{th} activation of π_i. This is done by expressing the following equation in **MI1cp**:

$$\begin{aligned} &\forall \pi_i, \forall k \in [1; nFrames], \\ &((k-1) \times gMIF = 0 \mod (P_i)) \implies \big((k-1) \times gMIF \leqslant pO_i^k \leqslant k \times P_i\big) \end{aligned} \tag{6.25}$$

Finally, one must verify that every start date is an even multiple of the CPU clock. This is done by expressing the following constarint in **MI1cp**:

$$\forall \pi_i, \forall k \in [1; nFrames], pO_i^k \equiv 0 \mod (Clk) \tag{6.26}$$

Where Clk is the value selected as clock for every active CPU in the multicore module.

Non-overlapping Partition Windows A given core cannot execute more than one task at any given time. This can be translated at partition-level: a given core cannot reserve a time slot for two different partitions at any given time. As such, the following constraint is expressed in order to verify the absence of overlapping windows for any couple of partitions on any core of the multicore module:

$$\begin{aligned} &\forall as_a \in AS, \forall \pi_i, \pi_j \in \mathcal{P}^{as_a}, \\ &\forall (k,t) \in [1; nFrames]^2 \mid \left(\begin{array}{c} ((k-1)gMIF \equiv 0 \mod (P_i)) \\ \wedge \\ ((t-1)gMIF \equiv 0 \mod P_j) \end{array} \right), \\ &\big(\big(pO_i^k \geqslant pO_j^t + E_j^t\big) \vee \big(pO_j^t \geqslant pO_i^k + E_i^k\big)\big) \end{aligned} \tag{6.27}$$

Schedule Verification

Partition-level scheduling verification Each partition time window must respect the periodicity of the corresponding partition, i.e. the end of each window must occur before the beginning of the next periodic activation of the partition. This corresponds to equation (5.70) being expressed as a constraint in **MI1cp**. Moreover, each start date must remain smaller than the corresponding latest start date $latest_i^k$ defined using task-level information; this corresponds to equation (5.71) being defined as a constraint of **MI1cp**.

Finally, each partition window start date must be a multiple of the corresponding core clock:

$$\forall \pi_i \in \mathcal{P}, \forall k \in [1; nFrames], pO_i^k \equiv 0 \mod Clk \tag{6.28}$$

Message passing and precedence constraints The partition schedule must take into account inter-partition communications and precedence relations. In fact, the generated partition-level schedule must force the order of occurrence of partitions windows according to their communications and precedences. The dependence requirements for the schedule generation have been presented in chapter 5 subsection 5.5 page 145. As such, the following equations must be expressed as constraints of **MI1cp**:

- Equations (5.72) and (5.73) for equiperiodic partitions involved in some dependence relations.
- Equations (5.74) and (5.75) in the case of multiperiodic partitions dependence relations where the message producer is faster than the consumer.
- Equations (5.76) and (5.77) in the case of multiperiodic partitions dependence relations where the message consumer is faster than the producer.

Objective Function

More than one solution of **MI1cp** may exist, so an objective function can be used in order to select the most optimized one according to a given optimization criteria. An appreciated characteristic for a partition schedule is the existence of slack times between two successive time windows of the schedule; indeed, sticking windows right next to each other increases the risk of redesign and verification if some partition needs bigger windows after further investigation. No such situation can happen if our timing analysis approach is exploited, but it can happen when low criticality applications are implemented on multicore platforms without undergoing prior timing analysis; it is indeed not required out of lower DAL applications to be verified using WCET analysis techniques. If such applications are supposed to be integrated as an IMA architecture onto a multicore platform with DAL A applications for instance, the runtime behavior of the DAL A applications can be disturbed due to the lower DAL applications that have not been verified, even though the DAL A applications have been safely analysed using our WCET analysis.

Inserting slack could then reduce such risks to some extent, and also comforts systems designers by preserving the current habit of systematically allowing some slack times between two consecutive windows in the MAF schedule. In order to do so in the one-to-all integration strategy proposed in this thesis, the following objective function can be defined:

$$\text{Minimize} \quad - \max_{\substack{(i,j)\in[1;N_P]^2 \wedge \\ (k,p)\in[1;nFrames_i]\times[1;nFrames_j]}} \left(pO_i^k - pO_j^k\right) \tag{6.29}$$

Step AS2: Task-Level schedulability analysis

Step AS2 is performed by each application supplier separately, in order to verify that it is possible to find a feasible schedule respecting the imposed partitions time windows. One way to do so is to perform a design space exploration aimed at determining whether there exists such a schedule, and exhibit one if such schedule(s) exist.

To automate step AS2, we define a CP named **AS2cp**, to generate activation dates of the tasks inside each partition and for one complete MAF. The output is either a valid schedule, respecting all constraints; or the certainty that there exists no such task schedule if **AS2cp** has no solution.

As mentioned before, an actual schedule is built for tasks, as a proof of existence of a valid schedule for the given configuration. If the corresponding tasks are to be scheduled according to a non-preemptive scheduling policy, then the generated schedule can be used by the corresponding application supplier to make sure it will be enforced at runtime. However, if some partitions are defined with preemptive task sets, the task-level schedule generated in step **AS2** only serves as a proof of existence of a valid schedule, but will not correspond to the runtime schedule. As mentioned before, tO_i^k will then correspond to the jitter upon first activation of τ_i^k, and tasks instances WCETs w_i^k respective upper-bounds are computed as outputs of a schdulability analysis.

AS2cp generates activation dates for each task instance inside one MAF. **AS2cp** also marks the end of the one-to-all integration strategy.

Inputs of AS2cp The inputs of **AS2cp** are the partitions activation offsets pO_i^k, CPU time budgets E_i^k and total number of windows $nbWindows_i$ in the MAF schedule.

Additional inputs to step AS1 are two matrices computed as outputs of step AS1: for each application and their respective partitions, **AS1cp** takes as input the task-to-core allocation na and the task-to-memory controller allocation $t2mc$.

Variables of AS2cp The task instances activation dates tO_i^k and WCETs w_i^k are the main outputs of **AS1cp**, along with the information on whether the final system configuration is schedulable at task-level for all partitions. If it is not the case, **AS2cp** will detect that the expressed problem has no solution. If it is the case, **AS2cp** will exhibit a solution that respects all scheduling constraints expressed in the next paragraphs.

Outputs of AS2cp The output of **AS2cp** is a task-level schedule for the corresponding partitions when there exists at least one such schedule. If it is not the case, then it means the timing analyses proposed in this thesis and embedded in **AS2cp** were not able to find a task-level schedule for the given partition-level schedule built by the module integrator in step MI1 for the corresponding partitions. If a partition falls into such a situation, the module integrator must propose a new partition-level schedule to the corresponding application supplier or propose some configuration modifications. The outputs of **AS2cp** are the task instances activation dates tO_i^k and WCETs w_i^k.

Schedule Generation Constraints

Activation dates definition A task may have a different period than its partition. As such, it is not scheduled in each time window of its partition in a given MAF. However, similarly to partitions windows start dates, $nFrames$ start dates tO_i^k are defined for each task τ_i, all of which not corresponding to an actual execution of τ_i. For all index k such that tO_i^k does not correspond to the actual k^{th} execution of τ_i must be set to zero in **AS2cp**:

$$\begin{aligned}&\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall k \in [1; nFrames], \forall \tau_j \in \mathcal{T}^{as_a} \mid pid_j = i,\\ &((k-1) \times gMIF \neq 0 \mod T_j) \implies \left(tO_i^k = 0\right)\end{aligned} \tag{6.30}$$

Otherwise, all other activation dates must be in the corresponding definition domain:

$$\begin{aligned}&\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall k \in [1; nFrames], \forall \tau_j \in \mathcal{T}^{as_a} \mid pid_j = i,\\ &((k-1) \times gMIF = 0 \mod T_j) \implies \left((k-1) \times T_j \leqslant tO_i^k \leqslant k \times T_j\right)\end{aligned} \tag{6.31}$$

Finally, all start dates must be even multiples of the CPU clock.

$$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i], \quad tO_i^k \equiv 0 \mod Clk \tag{6.32}$$

Partition-level windows enforcement The relation expressed in equation (6.31) is valid for any real-time environment. However in IMA software, a task execution can occur only within the boundaries of its partition time windows. As such, equation (6.31) must be reinforced by the following equation:

$$\begin{aligned}&\forall as_a \in AS, \forall \pi_i \in \mathcal{P}^{as_a}, \forall k \in [1; nFrames], \forall \tau_j \in \mathcal{T}^{as_a} \mid pid_j = i,\\ &((k-1) \times gMIF = 0 \mod T_j)\\ &\implies \left(pO_i^k \leqslant tO_j^k\right) \wedge \left(tO_j^k + w_j^k \leqslant k \times pO_i^k + E_i^k\right)\end{aligned} \tag{6.33}$$

The left member of relation 6.33 selects tasks of a given partition that are activated in the k^{th} window of their partitions respectively; the right member then forces the task execution to start and finish within the boundaries of the partition k^{th} window.

Non-overlapping executions If the partition under evaluation implements non preemptive tasks, the output of **AS2cp** will either be the absence of solution or a static schedule ensured to be valid. To ensure such validity, one must first ensure that the produced schedule does not implement overlapping executions of different tasks on the same core. The absence of collision in the schedule is verified by expressing the following constraint:

$$\begin{aligned}&\forall as_a \in AS, \forall \pi_m \in \mathcal{P}^{as_a} \mid (isPreemptive = 0),\\ &\forall \tau_i, \tau_j \in \mathcal{T}^{as_a} \mid (pid_i = pid_j), \forall k \in [1; nFrames],\\ &\left(\sum_{p=1}^{N_C} na_{pi} \times na_{pj} \neq 0\right) \wedge ((k-1) \times gMIF = 0 \mod T_i)\\ &\wedge ((k-1) \times gMIF = 0 \mod T_j)\\ &\implies \left(\left(tO_j^k \geqslant tO_i^k + w_i^k\right) \vee \left(tO_j^k + w_j^k \leqslant tO_i^k\right)\right)\end{aligned} \tag{6.34}$$

The first relation of the left part of the constraint allows to focus on tasks of a partition π_m that are allocated to two different cores. The other two relations of the left part of the constraint enables to select tasks that are activated in the k^{th} window of π_m. Finally, the right part of the constraint force the activation dates of the two tasks to lead to two non-overlapping time interval reserved for their corresponding execution.

No interference if no parallel execution Such a constraint can help the solving process of **AS2cp** converging towards schedules where the maximum number of task instances are scheduled alone on the entire multicore. To do so, we exploit matrix $interfering$ to detect whenever two task instances have overlapping execution time intervals according to their respective activation dates and WCETs, and specify that if a task instance execution interval is not overlapping in time with any other task instance on the other cores, its interference delays are null.

$$(overlapping_{i,j,k,p} = 0) \implies \Big(d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p) = 0\Big) \tag{6.35}$$

Schedule Verification Constraints

Task-level schedulability analysis The task level timing verification for the schedule generation problem has been explained in chapter 5, and consists in verifying that all scheduled tasks respect their respective deadlines.

To verify that a task instance τ_i^k respects its deadline, the corresponding WCET w_i^k must be computed using equation (5.11), which must therefore be embedded in the CP as a constraint. An additional constraint to verify that relation (5.61) is satisfied must then be expressed in **AS2cp**.

Message passing and precedence constraints The generated activation dates for each task instance must be coherent with the message exchanges and precedence relations expressed in the system's requirements. This type of requirements has been studied in chapter 5 section 5.5 page 143. As such, the following equations must be expressed as constraints of **AS2cp**:

- Equations (5.64) and (5.65) for dependence relations due to precedence relations and message-based communications between equiperiodic tasks,
- Equations (5.66) and (5.68) for dependence relations where the preceding task is faster than the successor task,
- Equations (5.67) and (5.69) for dependence relations where the preceding task is slower than the successor task.

Partition windows enforcement at task-level Tasks activation dates must be set such that the start and end of execution of a task occurs inside one of its partition time windows. To do so, equations (5.62) and (5.63) must be expressed as constraints of **AS1cp**.

Objective function

As more than one solution may exist, it is interesting to optimize the selection process of **AS1cp** by asking for scheduling tables where each CPU workload is reduced as much as possible. The usual workload definition is modified in order to take into account inter-task interference delays rather than just their execution duration in isolation which is a constant of the current problem. The corresponding objective function is thus expressed as follows:

$$\text{Minimize} \sum_{p=1}^{N_C} \left(\sum_{i=1}^{N_T} na_{pi} \times \sum_{k=1}^{nbActiv_i} \frac{\big(d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p)\big)}{gMAF} \right) \tag{6.36}$$

As explained before, workload minimization is equivalent to interference minimization in the expressed CP, which also helps reducing the portion of time of partition windows actually used for the instances execution, as covered by an objective function asking to reduce the sizes of each partition time window in one MAF. The latter objective can also be implemented more explicitely using the following objective function, asking to minimize the time interval between the end of the last instance and the start of the first instance within the same window:

$$\text{Minimize} \quad \sum_{i=1}^{N_P} \sum_{k=1}^{Frames} \max_{\substack{\tau_j \in \mathcal{T} \wedge \\ pid_j = i}} \left(\sum_{\substack{\tilde{k}=1 \\ (\tilde{k}-1)T_j = (k-1)gMIF}}^{nbActiv_j} tO_j^{\tilde{k}} + w_j^{\tilde{k}} \right) - \sum_{i=1}^{N_P} \sum_{k=1}^{Frames} \min_{\substack{\tau_j \in \mathcal{T} \wedge \\ pid_j = i}} \left(\sum_{\substack{\tilde{k}=1 \\ (\tilde{k}-1)T_j = (k-1)gMIF}}^{nbActiv_j} tO_j^{\tilde{k}} \right) \tag{6.37}$$

Finally, minimizing partition context switches or maximizing slack times between two partition windows are two objectives that are not exploitable in **AS2cp** since they consist in optimizations of the MAF schedule, which requires partition-level information, whereas step AS2 is performed at task-level.

6.2 One-to-One Integration Strategy

Strategy Overview

As illustrated in figure 6.2, the one-to-one integration strategy consists in two steps: software allocation, and schedule generation. The validity of the selected allocation and schedule are guaranteed respectively by exploiting the feasibility and schedulability analyses presented in chapter 5. We describe here how each of these two steps are performed.

Partition-to-Core Allocation

In the first step of the one-to-one integration strategy, the module integrator configures the software/hardware allocation of the avionics software on the cores of a multicore. To do so while saving time and effort but also guaranteeing the validity of the selected allocation, we define a CP named **ALLOCcp** to handle the allocation automatically. **ALLOCcp** embraces the timing verification for the allocation defined in chapter 5.

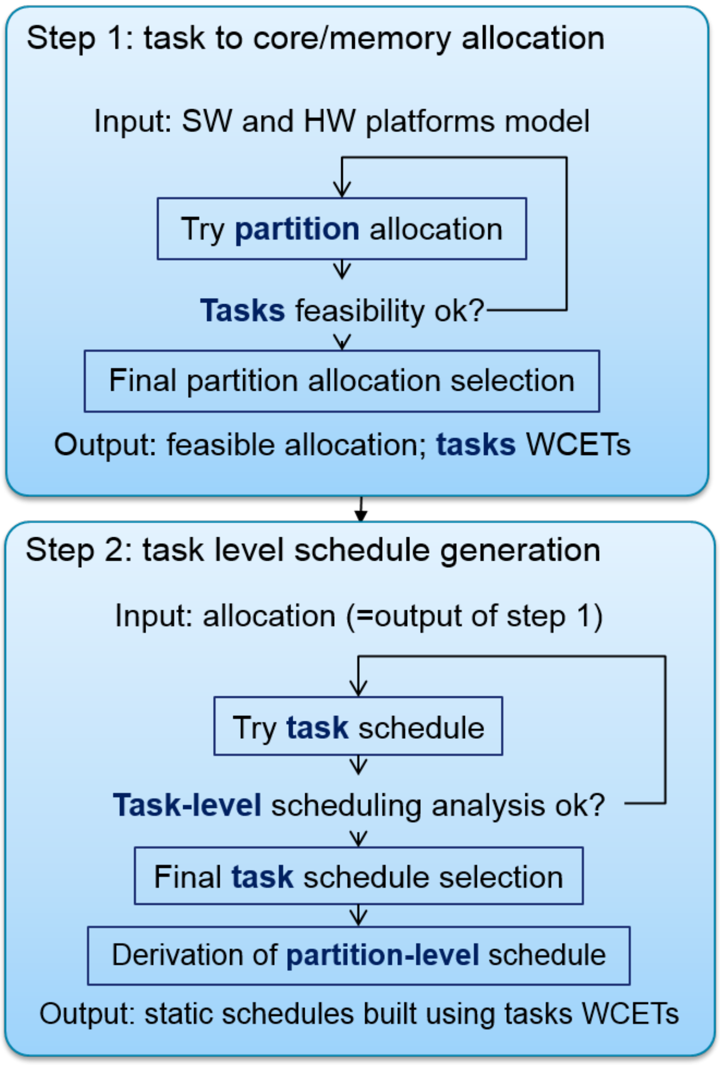
Inputs of ALLOCcp The inputs of the CP are the software and the hardware platform models as presented in chapter 4, in particular the partitions periods P_j and memory context budgets per path $pRam_j^k$, their respective tasks vectors $(C_i, T_i, D_i, H_i, prio_i, tRam_i)$, and matrices $PART$, msg and $prec$.

Variables of ALLOCcp The decision variables of the allocation CP are: a, $p2mc$ representing the software/hardware allocation, and w_i, J_i for the feasibility analysis. The auxiliary variables are R_i, E_i, both derived from w_i and J_i.

Outputs of ALLOCcp The output is a software/hardware allocation ensured to be valid.

Core Allocation Constraints

This subsection presents all constraints expressed in **ALLOCcp** that are related to the allocation of the partitions to the cores of the multicore platform.


Step 1: task to core/memory allocation
Input: SW and HW platforms model
Try partition allocation
Tasks feasibility ok?
Final partition allocation selection
Output: feasible allocation; tasks WCETs
Step 2: task level schedule generation
Input: allocation (=output of step 1)
Try task schedule
Task-level scheduling analysis ok?
Final task schedule selection
Derivation of partition-level schedule
Output: static schedules built using tasks WCETs
Figure 6.2: Steps of the One-to-One Integration Strategy

All partitions are allocated once Each partition must be allocated exactly once in the final allocation configuration. This is expressed in **ALLOCcp** as follows:

$$\forall \pi_i \in \mathcal{P}, \quad \sum_{p=1}^{N_C} a_{pi} = 1 \tag{6.38}$$

No overloaded core Similarly to the allocation search in the one-to-all integration strategy, no core should be overloaded in the finally selected allocation in the one-to-one integration strategy. This can be expressed by defining a constraint forcing each core utilization ratio to be under 100%. The corresponding constraint can then be expressed as follows:

$$\forall p \in [1; N_C], \quad \sum_{i=1}^{N_P} a_{pi} \times \frac{(C_i^p + C_{SW_p} + d_{RAM}(w_i, H_i^p) + d_{INT}(w_i, H_i^p))}{T_i} \leqslant 1 \tag{6.39}$$

All cores are used If there are more partitions than cores, then all cores must be used; this can be expressed as follows:

$$\forall p \in [1; N_C], (N_P \geqslant N_C) \implies \left(\sum_{i=1}^{N_P} a_{pi} \geqslant 1 \right) \tag{6.40}$$

Otherwise, if there are less partitions than the nuomber of cores to be used on the multicore, there should be only one partition per core in the selected allocation, in order to maximize as much as possible the number of cores used. This can be expressed as follows:

$$\forall p \in [1; N_C], (N_P < N_C) \implies \left(\sum_{i=1}^{N_P} \sum_{\substack{j=1 \\ j \neq i}}^{N_P} a_{pi} \times a_{pj} = 0 \right) \tag{6.41}$$

Memory Path Allocation Constraints

This subsection presents all constraints expressed in **ALLOCcp** that are related to the memory path allocation.

Epartition is allocated to at least one memory path Each partition should be allocated to a memory controller. Indeed, if a partition that is not allocated to any memory controller, then its tasks will not be able to access the main memory at runtime. As such, the following constraint is expressed to ensure that all partitions are allocated to at least one memory controller:

$$\forall \pi_i \in \mathcal{P}, \quad \sum_{k=1}^{N_{MC}} p2mc_{ki} \geqslant 1 \tag{6.42}$$

All memory paths are used All memory controllers must be used, in order to benefit from the available bandwidth and balance memory accesses as much as possible at runtime. This can be done by expressing the following constraint:

$$\forall k \in [1; N_{MC}], \quad \sum_{i=1}^{N_P} p2mc_{ki} \geqslant 1 \tag{6.43}$$

Realistic Memory Path Allocation Another verification that must be performed in relation with the memory is the coherence of the memory paths allocation with regards to the core-to-memory physical paths, i.e. verifying that the partitions have all been allocated to memory paths that are actually wired to the core they each have been allocated to. This can be done by comparing the values of *p2mc* to *c2mc*. In particular, a partition cannot be allocated to a path that is not wired to the core it is allocated to. The following equation expresses the corresponding constraint on the values of a and *p2mc*:

$$\forall k \in [1; N_{MC}], \forall \pi_i \in \mathcal{P}, \quad \left(\sum_{p=1}^{N_C} a_{pi} \times c2mc_{pk} = 0 \right) \implies (p2mc_{ki} = 0) \tag{6.44}$$

Indeed, the first sum enables to retrieve the index of the core p to which π_i has been allocated, corresponding to the only non-null term a_{pi} of the sum. Multiplying this non-nul term by $c2mc_{kp}$ for a given memory controller k enables to verify whether core p is physically wired to the controller k: the total sum is non-null if it is indeed the case, and null otherwise. And if core p is not wired to the memory controller k, then any partition π_i allocated to core p cannot be allocated to the memory controller k, i.e. $p2mc_{ki}$ must be equal to zero.

Coherent memory path allocation regarding tasks memory footprints In addition, the partition-to-path allocation must be coherent regarding the available memory space per path and the memory contexts of partitions allocated to each path. The following equation can be embedded as a constraint of the allocation problem in order to verify that the partition-to-memory-controller allocation is coherent with the partitions memory footprint:

$$\forall k \in [1; N_{MC}], \quad \sum_{i=1}^{N_P} p2mc_{ki} \times \left(\sum_{\substack{j=1 \\ pid_j = i}}^{N_T} tRam_j \right) \leqslant mcSize_k \tag{6.45}$$

Coherent memory path allocation regarding Message-Based Communications If a partition π_i is involved in some inter-partition communication with a partition on another core, then both partitions must be able to access the corresponding shared memory area. In our model, communications happen at task level: the sender task must be able to access the same memory area as the receiver task. At partition level, two partitions can be involved in IPC, and both containing some receiver and some senders. As a consequence, defining a constraint for path allocation in concordance with communications at task-level may be difficult depending on the complexity of the data exchanges defined in the software platform. To find a simpler alternative, we decide to specify that two partitions involved in communications with each other should use the same memory path(s). This corresponds to the following constraint:

$$\begin{aligned} &\forall \pi_i, \pi_j \in \mathcal{P}, \\ &(ipc_{ji} + ipc_{ij} \neq 0) \implies \left(\sum_{k=1}^{N_{MC}} p2mc_{ki} \times p2mc_{kj} \geqslant 1 \right) \end{aligned} \tag{6.46}$$

The term $(ipc_{ji} + ipc_{ij})$ enables to select couples of partitions (π_i, π_j) that are involved in inter-partition communications. For such couples, the partitions must have at least one memory path in common, i.e. there exists at least one memory path index k for which the term $p2mc_{ki} \times p2mc_{kj}$ is not null.

It is interesting to note that, coupled with an adequate objective function (targeting interference minimization for instance), this constraint is likely to have the CP solver automatically try to allocate partitions involved in communications with each other to the same core or directly to the same memory paths, in order to minimize simultaneous memory access requests, and therefore, minimize inter-task interference.

Schedulability Constraints

This subsection presents all constraints expressed in **ALLOCcp** that are related to the verification of timing-related properties of the evaluated allocations. Similarly to step AS1 in the one-to-all integration strategy, these constraints represent the feasibility analysis proposed in this thesis for early timing analysis and for guiding the allocation search in a safe manner by forcing the choice of allocation to correspond to an allocation that passes the feasibility test.

Message passing and precedence relations As in the one-to-all integration strategy, **ALLOCcp** must contain some constraints expressing each task jitter upon first activation J_i according to its precedence and message relations specified in the software architecture. Indeed as mentioned before, if a task does not have any predecessor or does not receive any message at runtime, its jitter J_i is null. On the contrary, if a task has predecessors or receives messages, its first activation must correspond to the end of execution of all its predecessors and/or any producer of a message it consumes. The equations corresponding to such constraints are the same as in step AS1 of the one-to-all integration strategy: as such, equations (6.11) and (6.12) defined page 155 are embedded in **ALLOCcp** as constraints to compute tasks jitters upon first activation.

Affinities and exclusion constraints The partition-to-core allocation must be in line with partition-to-core affinities expressed by the module integrator and/or system designers. As such, the following constarint must be expressed:

$$\forall p \in [1; N_C], \forall \pi_i \in \mathcal{P}, \quad \left(\mathit{coreAff}_{pi} \geqslant 1\right) \implies (a_{pi} \geqslant 1) \tag{6.47}$$

Similarly, partition-to-partition affinities must be respected in the partition-to-core allocation:

$$\forall \pi_i, \pi_j \in \mathcal{P}, \quad \left(\mathit{partAff}_{ij} = 1\right) \implies \left(\sum_{p=1}^{N_C} a_{pi} \times a_{pj} = 1\right) \tag{6.48}$$

Finally, the partition-to-partition exclusions expressed by the application supplier must be reflected in the selected partition-to-core allocation:

$$\forall \tau_i, \tau_j \in \mathcal{T}, \quad \left(\mathit{partEx}_{ij} = 1\right) \implies \left(\sum_{p=1}^{N_C} a_{pi} \times a_{pj} = 0\right) \tag{6.49}$$

All tasks are schedulable Last but not least, the feasibility of the tasks in the multicore allocation currently under evaluation is assessed. It is done using the system (5.7) in order to compute every task WCET w_i, WCRT R_i and jitter J_i, but also equation (5.51) in order to verify that all tasks WCRTs remain smaller than the respective deadline.

Objective function

As explained before in the one-to-all integration strategy, the most popular optimization feature for real-time software scheduling is the reduction of CPU workload. In addition to load balancing, the workload is proportional to inter-task interference delays, which add up to the tasks executions. Because of these delays, (i) the extra time available before tasks deadlines are reached at runtime is drastically shortened, and (ii) the integrator is given less flexibility for the setting of a valid global schedule for each core in the considered allocation. As such, reducing the total workload thus increases the flexibility of future design choices to be made when configuring the task schedule.

The defined objective function to minimize the total workload of the selected solution is the following:

$$\mathit{minimize} \sum_{p=1}^{N_C} \left(\sum_{i=1}^{N_P} a_{pi} \times \sum_{j=1}^{N_T} PART_{ij} \times \frac{C_i^p + d_{RAM}(w_i, H_i^p) + d_{INT}(w_i, H_i^p)}{T_i}\right) \tag{6.50}$$

Schedule generation

Once an allocation is selected, the module integrator then generates a static schedule. More precisely, he generates a task-level schedule, and the corresponding partition schedule is deduced from the task-level schedule.

To build a static schedule beforehand while saving time and effort, we propose a CP called **SCHEDcp**. In order to guarantee the validity of the generated schedule, **SCHEDcp** embraces the schedulability analysis defined in chapter 5.

Inputs of SCHEDcp The inputs of **SCHEDcp** are the entire software and hardware models (see chapter 4) and the partition-to-core and partition-to-memory path allocations that have been produced by **ALLOCcp**.

Variables of SCHEDcp The decision variables of the allocation CP are the tasks activation dates per MAF tO_i^k, along with the task instances WCETs w_i^k. The auxiliary variables are the tasks instances response times R_i^k derived from each w_i^k, the corresponding CPU time budget for each partition and frame E_j^k, and the partitions activation dates pO_i^k which are derived from their tasks activation dates.

Outputs of SCHEDcp The output of **SCHEDcp** is a task-level schedule that is guaranteed to enforce all timing-related constraints of the system. The partition-level schedule is then deduced from the task-level schedule and the task-to-partition allocation defined by $PART$.

Schedule Generation Constraints

Task instances definition One of the first constraints to be implemented in **SCHEDcp** is to ensure each activation date tO_i^k corresponds to the k^{th} instance of τ_i:

$$\forall \tau_i \in \mathcal{T}, \forall k \in [1; nbActiv_i], tO_i^k \in [(k-1) \times T_i;\, k \times T_i] \tag{6.51}$$

Non-overlapping executions on each core In the schedule to be generated, there should always be at most one running task per core at runtime, as overlapping executions on the same core are forbidden:

$$\begin{gathered} \forall p \in [1; N_C], \forall \tau_i, \tau_j \in \mathcal{T} \mid (i \neq j), \forall (k,m) \in [\![1; nbActiv_i]\!] \times [\![1; nbActiv_j]\!], \\ \left(\sum_{p=1}^{N_C} a_{p,pid_i} \times a_{p,pid_j} = 1 \right) \implies (\ (tO_j^m \geqslant (tO_i^k + w_i^k) \ \ \vee \ \ (O_i^k \geqslant tO_j^m + w_j^m)\ ) \end{gathered} \tag{6.52}$$

The left-hand side of equation (6.52) forces the execution of one of the two instances to finish before the start of the other.

Time sampling according to timer interrupt frequency Tasks instances activation offsets must be an even multiple of the corresponding core clock frequency.

$$\forall \tau_i \in \mathcal{T}, \forall k \in [\![1; nbActiv_i]\!], tO_i^k \equiv 0 \mod \left(\sum_{p=1}^{N_C} a_{pi} \times Clk_p \right) \tag{6.53}$$

Schedule Verification Constraints

Task-level scheduling verification Each task instance execution must complete before the corresponding deadline. As such, tasks instances WCETs w_i^k are computed first using the second equation in system (5.12) for non-preemptive tasks, system (5.11) for preemptive tasks, and equation (5.61) is embedded in **SCHEDcp** as well to check whether all instances respect their deadlines.

Partition-level scheduling verification Analogously at partition-level, all partitions time budgets must fit in one MAF. To ensure so, the partition level verification for the schedule generation problem represented by equation (5.60) is added as a constraint in **SCHEDcp**.

Message passing and precedence relations The generated activation dates for each task instance must be coherent with the message exchanges and precedence relations expressed in the system's requirements. This type of requirements has been studied in chapter 5 section 5.5 page 143. As such, the following equations must be expressed as constraints of **SCHEDcp**:

- Equations (5.64) and (5.65) for dependence relations due to precedence relations and message-based communications between equiperiodic tasks,

- Equations (5.66) and (5.68) for dependence relations where the preceding task is faster than the successor task,
- Equations (5.67) and (5.69) for dependence relations where the preceding task is slower than the successor task.

Linking inter-task interference with the knowledge of parallel executions Finally, we present the CP constraints expressing which equation should be used to compute w_i^k depending on the presence of other tasks running in parallel. A task instance suffers no interference if it is running alone on the entire multicore according to the defined core schedules:

$$\forall \tau_i \in \mathcal{T}, \forall k \in [\![1; nbActiv_i]\!],$$
$$\left(\sum_{p=1}^{N_C} a_{pi} \times \sum_{j=1}^{N_T} (1 - a_{pj}) \times \sum_{m=1}^{nbActiv_j} overlapping_{i,j,k,m} = 0\right) \implies \left(w_i^k = \sum_{p=1}^{N_C} a_{pi} \times C_i^p\right) \tag{6.54}$$

If on the contrary, there exist at least one task instance executed in parallel, interference delays suffered by τ_i^k in the worst-case are not null:

$$\begin{aligned} &\forall \tau_i \in \mathcal{T}, \forall k \in [\![1; nbActiv_i]\!], \\ &\left(\sum_{p=1}^{N_C} a_{pi} \times \sum_{j=1}^{N_T} (1 - a_{pj}) \times \sum_{m=1}^{nbActiv_j} overlapping_{i,j,k,m} \neq 0\right) \\ &\implies \left(w_i^k = \sum_{p=1}^{N_C} a_{pi} \times C_i^p + d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p)\right) \end{aligned} \tag{6.55}$$

where $d_{RAM}(w_i^k, H_i^p)$ and $d_{INT}(w_i^k, H_i^p)$ are computed using equations (5.38) and (5.41) respectively.

These two constraints come from commonsense: (i) in general in scheduling problems, schedules often present slack times where the CPU is idle; (ii) if a task instance τ_i^k is alone to run on an entire platform, then it has no competition to access the shared resources and therefore suffers zero interference delay, i.e. in our case, $d_{RAM}(w_i^k, H_i^p)$ and $d_{INT}(w_i^k, H_i^p)$ are both equal to zero. Then in such situations, computing w_i^k using equation (5.11) leads to unnecessary pessimism, as w_i^k is actually equal to the execution duration in isolation C_i^k.

Constraint (6.54) is not essential to scheduling problems as it does not represent a requirement of the system. However, without constraint (6.54), the corresponding CP expressed is not able to make the link between the temporal location of each task execution time interval in the schedule, with the presence of inter-task interference. If one task instance τ_i^k is alone to run on the multicore for instance, the CP formulation without constraint (6.54) would be unaware of the fact that τ_i^k has exclusive access to the memory and interconnect and therefore suffers no interference, and equation (5.11) would then be used, resulting in an overly pessimistic schedule.

To sum up, constraint (6.54) is not essential to the schedule generation problem CP definition as it does not represent a requirement of the system to be configured, but rather to an optimization feature since it will guide the search towards schedules where the least possible amount of parallel executions occur for a given software and hardware platforms. As such, constraint (6.54) acts as an optimization feature during system design.

Objective Function The objective function of **SCHEDcp** can be about CPU workload reduction:

$$\text{Minimize} \max_{p \in [1; N_C]} \left(\sum_{i=1}^{N_T} a_{p,pid_i} \times \sum_{k=1}^{nbActiv_i} \left(\frac{C_i^p + d_{RAM}(w_i^k, H_i^p) + d_{INT}(w_i^k, H_i^p)}{MAF_p}\right)\right) \tag{6.56}$$

6.3 Discussions

One-to-All Strategy: Partitions CPU Time Windows Computation. It is important to note that in step AS1, application suppliers do not schedule tasks, but rather just compute each partition time budget E_i^k per frame in the partition cycle, in order to be able to provide these budgets to the module integrator at the end of step AS1.

The resulting CPU time budgets could then be refined at the end of AS2, i.e. after a task-level schedule can therefore be configured. This would have enabled to compute each E_i^k budget using tasks instances WCETs w_j^m instead of using the same WCET w_j for all instances of τ_j in one partition cycle. The result would have been tighter partition windows durations, and possibly enough room on the corresponding multicore module for scheduling additional partitions.

However, this would have implied adding the WCET equations to **AS2cp**, along with defining new WCET variables $\tilde{E}_i^k$ to recompute the new partitions CPU time budgets, and then perform step MI1 again in order to use these tightened budgets to compute a new partition schedule, and then re-launch **AS2cp** in order to perform task-level verification, etc. To sum up, this would have meant increasing the complexity of **AS2cp** and extending the duration of the integration process solely for optimization purposes. In this thesis, we made the choice to keep the proposed strategies as simple as possible in order to gain in time and effort without loosing the safetiness and confidence in the exploited timing analysis metrics, the pessimism embedded in the produced bounds being seen as safety margins. This can be seen as interesting future work.

6.4 Summary

In this chapter we presented the two integration strategies proposed in this thesis for implementing multicore-based IMA systems. A general overview was given to ease comparison between the pros and cons of the two strategies, before presenting them in greater extent. Each strategy is divided in chronological steps performed by the different roles involved in the integration process. For each step, an approach for conducting the required verification and configurations is expressed. To save time and effort during the process, some steps are expressed through a CP, which formulation has been given as one of the contributions of this thesis. Each CP embeds all needed analyses for the result to be considered safe according to certification standards, and especially the DO-178.

Chapter 7

Evaluation Results

In this chapter, we present the evaluation of our contributions, and the obtained results. We first explain the goal of the tests performed to evaluate our contributions, before presenting the way the case study has been built and describing the hardware architecture. Then in section 7.4, we present the results about the evaluation of the gain in time and design optimization achieved thanks to our approach, and in section 7.3 we present the results of the application of our approach to a real target.

General Overview

The evaluation phase of the contributions proposed in this thesis is divided in two activities. The first one is the application of the proposed strategies for allocation and scheduling on an IMA case study and for a real target platform. The verification then consists in checking that no deadline miss occurs at runtime. To do so, a software and a hardware platform must be identified as inputs to the evaluation. They are modeled as suggested in this thesis, and one of the two strategies for allocation and schedule generation search must be applied on them. The outcome of the searches is then reproduced in the configuration of the software and hardware platforms so that the finally integrated system matches the allocation and the schedule that have been outputted by our approaches. The final system is then witched on and launched so that its runtime behavior can be observed. The applied strategy passes the evaluation for the given input platforms if no deadline miss occurs after a long period of observation. Several input situations can be evaluated: for instance, one can vary the number of cores to be used at runtime on the target platform. The absence of deadline miss in all situations ensures the validity of the generated schedule, but also the validity of the produced WCET upper-bounds. Such evaluation, along with the corresponding results, are described in section 7.3.

The second evaluation activity is theoretical, and focused on the allocation and scheduling search problems: we compare the solutions obtained when using one of our strategies against the solutions obtained when multicore interference is not taken into account. To do so, we proceed the same way as for the evaluation on a real target, without actually implementing the resulting allocation and schedule on the target; this is repeated twice per situation: once as suggested in this thesis, and another time while ignoring multicore interference in the computation of tasks WCET and WCRT, while still computing them in order to get the corresponding inter-task interference upper bound as computed in our models. The second run is also referred to in this thesis as *theoretical allocation and schedule generation search*, or *interference-oblivious allocation and schedule generation search.* For the same situation, the outputs of the two runs are compared in terms of average multicore interference, total workload and slowdown suffered by tasks due to multicore interference. The second test enables us to assess the optimization gain achieved thanks to our approach. Such evaluation, along with the corresponding results, are described in section 7.4.

For both activities, the same software case study an hardware platform models have been exploited. As no real IMA software was made available for use in the context of this thesis, we built our own case study. The rest of this chapter is organized as follows. We first present the software platform exploited in our tests, along with the corresponding explanations on how it has been built. We then present the hardware platform and its corresponding model, built following the approach proposed in this thesis. The results of the evaluation on a real platform is then presented, before introducing the theoretical comparison of our contributions with interference-oblivious allocation and scheduling searches. Finally, a section about certification considerations assesses our work compared to certification objectives currently set for multicore based IMA systems. This chapter then concludes with a "Discussions" section about the shortcomings and limitations of the evaluation regarding the reality of COTS, and a section summarizing the content of this chapter.

7.1 Software Case Study Generation

Our contributions being specific to aerospace industries, the ideal evaluation phase should have been performed with actual, IMA software. However in practice, for the sake of core business protection, the access to such software is often either confidential or carefully restricted. As such, it has not been possible to have access to actual IMA software in the context of this thesis. Instead of using actual software for the evaluation of this thesis, we built a mock IMA case study based on an academic, non-IMA avionics software architecture [108], open source benchmarks of the TACLeBench benchmark suite [13], and our knowledge of IMA architectures and IMA orders of values for each parameters such as partitions periods or MAF values. We describe in this section how the exploited software case study has been built.

General Architecture

The authors of [108] present a military avionics case study to be used for academic purposes. In this case study, the architecture of nine software functions are described in terms of tasks. For each task, the corresponding scheduling parameters are given, such as information on their periodicity or approximate duration in average on a single-core platform.

We took the case study in [108] as a basis and adapted it as follows:

- Software functions have been identified as one or several partitions;
- Tasks periods have been modified so that they consist of harmonic sets;
- Partitions periods have been set as equal to the smallest period of their respective tasks;
- Aperiodic tasks are turned into periodic ones by assigning them a period equal to the period of their respective partitions;
- Benchmarks from the TaCLEbench benchmark suite [13] have been used in order to construct tasks entry points based on their respective average duration in single-core environments.

The resulting software architecture consists in nine partitions and twenty nine tasks in total (cf. table 7.1).

The next step of building our case study consists in selecting the benchmarks to be exploited for each task entry point. To do so, each benchmark of the TACLEBench benchmark suite are analyzed in order to get an upper bound on their respective single-core execution duration and maximum number of main memory access. The resulting values will determine which benchmark is selected for which task. This is done as a part of the preliminary analysis required in the contributions proposed in this thesis, and explained in the next subsection.

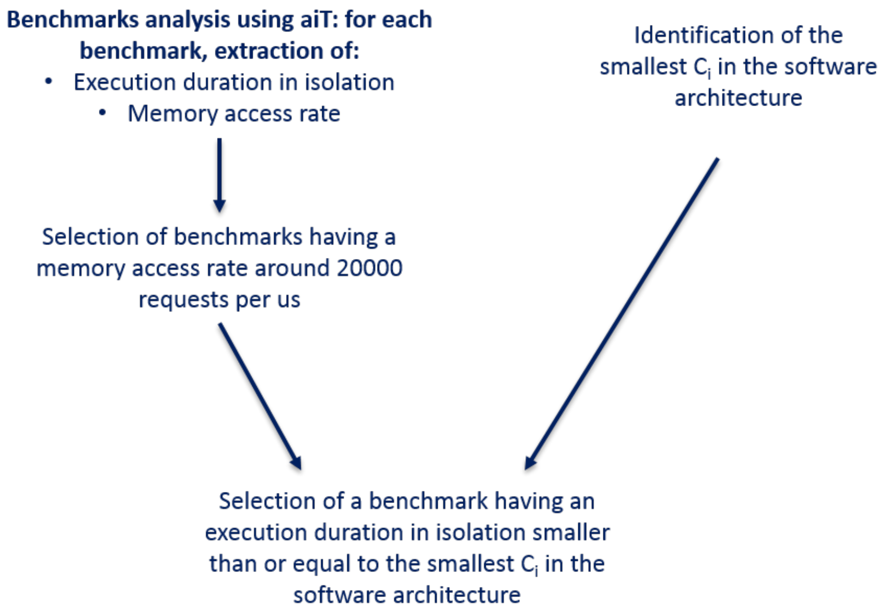

Benchmarks analysis using aiT: for each benchmark, extraction of:
• Execution duration in isolation
• Memory access rate
Selection of benchmarks having a memory access rate around 20000 requests per us
Identification of the smallest C_i in the software architecture
Selection of a benchmark having an execution duration in isolation smaller than or equal to the smallest C_i in the software architecture

Figure 7.1: Benchmark Selection Process [9]

Preliminary Analysis and Benchmark selection

One criteria of benchmark selection that is important when evaluating our work is the rate at which access to the main memory is requested at runtime. It must be realistic and match the expected access rate for multicore COTS currently under industrial evaluation for future usage. To our knowledge, this corresponds to a memory access rate of 20 000 requests per millisecond approximately.

The memory access rate parameter of benchmarks is linked to the maximum number of memory access requests that can be generated by a task, H_i, depending on the benchmarks called in its entry point. Another parameter of benchmarks which has an influence on a task calling it in its entry point is its execution duration in isolation.

Regarding tasks, two important parameters of the model of a task τ_i that must be given as inputs for the timing analysis and the integration strategies activities, are upper bounds on the execution duration in isolation C_i and the maximum number of main memory access requests per execution H_i. These two parameters are extracted from single-core analysis performed on the tasks respective entry points using aiT. In the software case study exploited for the evaluation phase of this thesis, tasks entry points consist in one or several calls to one of the TACLeBench benchmark suite.

One unique benchmark is selected as a unitary benchmark, and exploited for all entry points in order to enforce the same memory access rate for all tasks of the software platform. The number of times the benchmark is called in each entry point varies from one task to the other: for a given task τ_i, the selected benchmark is called as many times as needed in order for the resulting entry point to have an execution duration in isolation as close as possible to C_i. To do so, one must know the execution duration in isolation of the benchmark. In fact, as illustrated in figure 7.1, the execution duration in isolation of benchmarks actually is one of the two criterion for benchmark selection, the first one being the memory access rate which must be as close as possible to 20000 requests per millisecond. Indeed, the selected benchmark must have an execution duration in isolation that corresponds to a value that is smaller than or equal to the smallest C_i of the software architecture. The reason is that it must be possible to call the selected benchmark one or several times in each task entry point so that the final entry point has a duration in isolation as close as possible to the corresponding C_i value given in [108]. For instance if a benchmark matches regarding its memory access rate but has a duration in isolation in seconds while most of the tasks have durations in milliseconds, then the benchmark is not fit for usage in our tests.

As mentioned in figure 7.1, in order to extract an upper bound of each benchmark execution duration in isolation and maximum number of main memory access requests, we used aiT Analyzer [2] from the AbsInt company. AiT computes a safe upper-bound of tasks WCETs thanks to a static code-level analysis. To do so, aiT takes as input the binary executable of an embedded system, and constructs the corresponding control flow graphs of each task entry point. WCET bounds are then produced using global path analysis. AiT has been successfully used for timing verification in avionics and automotive systems in single-core environments [157, 89].

COTS processors currently supported by aiT are all single-cores. However, execution durations in isolation being unchanged whether the considered hardware platform is single-core or multicore, aiT can be exploited in multicore environments as well for deriving upper bounds of execution durations in isolation. In our case, we also need to derive an upper-bound of the maximum number of main memory access requests that each benchmark can generate at runtime. Such a parameter may be influenced by inter-task interference that can occur at runtime. For instance after a preemption, a task may experience more cache misses – and therefore, issue additional main memory access requests – because the task that preempted it evicted some of its data from the cache. The sharing of resources in multicore platforms may cause additional inter-task interference. However in our case, access requests to the main memory correspond to L1 cache misses, and the L1 is not shared by the cores of a multicore platform. As such from the point of view of main memory access requests, being in a multicore environment as such modeled in our approaches does not bring additional situations of inter-task interference than in sinngle-core environments. On the other hand, aiT is capable of deriving an upper bound on tasks worst-case number of L1 misses. As a result, aiT can be exploited in order to derive an upper bound of the maximum number of main memory access requests for each benchmark.

To sum up, we analyzed all benchmarks of the TACLeBench benchmark suite using aiT in order to derive, for each bench, an upper bound on the corresponding execution duration in isolation and the maximum number of main memory access requests per benchmark.

Tasks Entry Points Composition

In the end, when applying the analysis approach presented in the previous paragraphs and illustrated in figure 7.1, the benchmark from the TACLeBench suite that has been selected is *codec_codrle*: it has an execution duration in isolation of 574 microseconds and a memory access rate of 22 018 requests per millisecond.

To construct tasks entry points, *codec_codrle* was called as many times as needed inside each task, so that each C_i would match as much as possible the value given for C_i in [108]; the actual value of C_i but also the value of H_i are then derived according to the number of calls to *codec_codrle*. The resulting software case study used in order to evaluate the contributions proposed in this thesis is summarized in table 7.1.

7.2 Hardware Architecture Representation

The multicore platform exploited in this thesis for evaluation purposes is the QorIQ P4080 [9] evaluation board from Freescale/NXP. The RTOS embedded on the QorIQ is VxWorks653 3.1 Multicore Edition [14], the ARINC653-compliant IMA from Wind River. We provide in the next paragraphs information about the multicore platform and the RTOS required in their respective models presented in this thesis.

π_j	τ_i	$prio_i$	C_i (ms)	H_i	$T_i{=}D_i$ (ms)	P_j (ms)
1	1	2	7,462	164294	1600	1600
	2	1	5,74	126380	1600	
2	3	1	1,722	37915	400	400
	4	2	1,722	37915	400	
	5	3	1,722	37915	800	
3	1	3	0,574	12638	800	800
	7	2	0,574	12638	800	
	8	4	3,444	75829	800	
	9	3	1,722	37915	800	
4	10	1	0,574	12638	1600	1600
	11	4	1,722	37915	1600	
	12	2	0,574	12638	1600	
	13	6	6,888	151656	1600	
	14	5	5,74	126380	1600	
	15	3	0,574	12638	1600	
5	16	5	5,74	126380	1600	1600
	17	4	5,74	126380	1600	
	18	6	7,462	164294	1600	
	19	1	0,574	12638	1600	
	20	2	0,574	12638	1600	
	21	3	1,722	37915	1600	
6	22	1	0,574	12638	1600	1600
	23	2	0,574	12638	1600	
	24	3	0,574	12638	1600	
7	25	1	0,574	12638	400	400
	26	2	1,722	37915	800	
8	27	1	1,722	37915	800	800
9	29	2	4,592	101104	800	400
	30	1	0,574	12638	400	

Table 7.1: Data used for the Software Case Study

Wind River VxWorks653 3.1 Multicore Edition

Wind River's VxWorks653 3.1 Multicore Edition [14] is an ARINC653-compliant RTOS developed for scheduling IMA applications. VxWorks653 3.1 Multicore Edition offers AMP scheduling capabilities only: no dynamic allocation or migration features are allowed, and a given partition can only be assigned to a unique core. This corresponds to one of the main assumptions about the multicore environment and the context of this thesis; as such, all contributions proposed in this thesis remain compatible with VxWorks653 3.1, even the one-to-all integration strategy, where one application is running on all cores of the same multicore platform at any given time. To do so, the main requirement is that although the same application is running on all cores, each core runs its own set of partitions at runtime.

NXP P4080 Evaluation Board

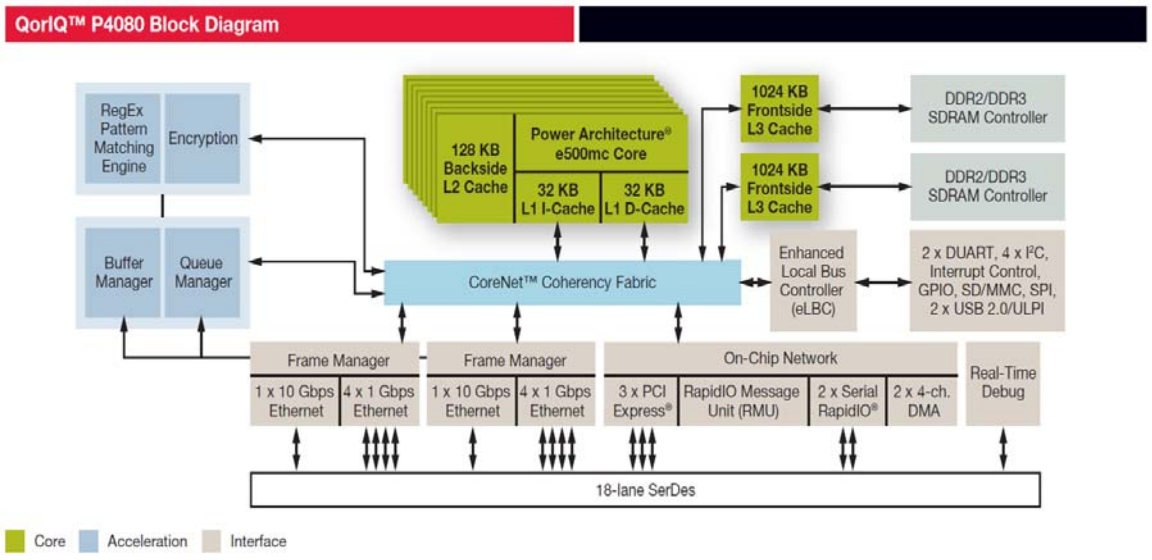

QorIQ™ P4080 Block Diagram
RegEx Pattern Matching Engine
Encryption
Buffer Manager
Queue Manager
128 KB Backside L2 Cache
Power Architecture® e500mc Core
32 KB L1 I-Cache
32 KB L1 D-Cache
1024 KB Frontside L3 Cache
1024 KB Frontside L3 Cache
DDR2/DDR3 SDRAM Controller
DDR2/DDR3 SDRAM Controller
CoreNet™ Coherency Fabric
Enhanced Local Bus Controller (eLBC)
2 x DUART, 4 x I²C, Interrupt Control, GPIO, SD/MMC, SPI, 2 x USB 2.0/ULPI
Frame Manager
1 x 10 Gbps Ethernet
4 x 1 Gbps Ethernet
Frame Manager
1 x 10 Gbps Ethernet
4 x 1 Gbps Ethernet
On-Chip Network
3 x PCI Express®
RapidIO Message Unit (RMU)
2 x Serial RapidIO®
2 x 4-ch. DMA
Real-Time Debug
18-lane SerDes
Core
Acceleration
Interface

Figure 7.2: Block Diagram of the Freescale/NXP QorIQ P4080 Processor [9]

According to NXP, the P4080 is intended for industrial applications such as aerospace, defense, factory automation and networking. Figure 7.2 displays the block diagram of the P4080 as presented by NXP [9]. As illustrated in figure 7.2, the P4080 features eight PowerPC e500mc cores capable of running at up to 1,5 GHz each. For the purpose of evaluating the outcomes of this thesis, all active cores are running at 1,5 GHz in every configuration implemented during our tests.

The P4080 features a three level cache hierarchy, with 32 KB of instruction and data L1 cache per core, 128 KB unified L2 cache per core and 2 MB of shared cache. In order to match the assumptions made in this thesis, the L2 and L3 cache levels have been deactivated at runtime in every configuration implemented during our tests.

As illustrated on figure 7.2, the main memory consists in two blocks of 2 GB of DDR3 each. There exists two memory controllers, and therefore, two paths to the main memory from the point of view described in this thesis. Each controller is linked to one of the two memory blocks. Moreover, the eight cores are implemented as two clusters of four cores, each cluster being wired to only one of the two memory controllers [29]. As such, if a core is wired to the first memory controller, it can only access memory addresses of the first main memory block, and it cannot access neither the second memory controller or the second main memory block.

L1 Cache In order to perform the preliminary analysis described in section 7.1, information on the L1 cache is required by aiT in order to derive accurate upper bounds on C_i and H_i for each task τ_i. In particular, aiT is capable of deriving a worst-case upper-bound on the maximum

Number of Sets	128
Associativity	8
Line Size	32 KB
Cache Policy	pseudo-least recently used

Table 7.2: L1 Cache Model (Identical for both Data and Instruction L1 Caches)

Parameter	Value	Unit	Definition
tCK	0.0015	μs	DRAM clock cycle time
tRP	8	cycles	precharge latency
tRCD	8	cycles	activate latency
CL	9	cycles	CAS read latency
WL	7	cycles	CAS write latency
BL	8	columns	burst length
tWTR	7	cycles	write to read delay
tWR	10	cycles	write recovery time
tRRD	11	cycles	activate to activate delay
tFAW	20	cycles	four activate windows
Ncols	1024	–	number of columns per row
Nreorder	12	requests	reordering window size
lbus	0.001	μs	bus latency

Table 7.3: Data used for the Main Memory Model

number of L1 cache misses corresponding to the entry point of a task τ_i, which also corresponds to H_i in our model. The number of hits and misses depends on the cache organization. In fact, caches are organized in lines and sets; the line in which some data (resp. instruction) can be stored in cache depends on its memory address; when a cache is full and a new data (resp. instruction) must be stored in cache, the cached data (resp. instruction) to be evicted from the cache in order to store the new data (resp. instruction) depends on the cache eviction policy that is implemented.

To take into account such cache organization information during the analysis deriving tasks C_i and H_i, aiT must be able to know the cache organization. As such, some essential parameters must be given as input to the preliminary analysis. In aiT, caches are modeled according to the following parameters:

- The cache line size, i.e. number of bytes in one cache line;
- The associativity of the cache, which describes which cache lines can store data located in which main memory addresses: the associativity corresponds to the number of locations in cache where a given data (resp. instruction) may reside depending on its memory address;
- The number of sets of the cache;
- The cache policy, determining, when the cache is full, which cache line must be used to store a new data (resp. instruction) – i.e., emptied and filled up with the new data (resp. instruction) to be stored in cache.

In the P4080, data and instruction caches are analogously organized, according to the parameters given in table 7.2.

Main Memory As mentioned in chapter 4 subsection 4.4, the main memory is represented through some of its hardware parameters which description can be found in the processor datasheet. Table 7.3 presents the values corresponding to the configuration of the main memory embedded on the P4080 evaluation board.

7.3 Validation on a Real Target

In this section, we describe the conclusions that could be drawn when we implemented the scheduling tables generated by our approach, on our HW platform to validate accuracy compared to reality of industrial systems and platforms.

Configuration Setup

The goal of this evaluation is to implement the configurations selected as output of our allocation and scheduling CP on a real target, and verify that no deadline miss occurs at runtime after observing several MAFs. If no deadline miss is indeed observed, two conclusions can be drawn :

- The timing analysis produced accurate WCET bounds for each task, and
- The partitions time windows and the generated schedules are accurately chosen, i.e. they enable to accurately respect all timing requirements of the system.

The evaluation process corresponding to such verification is the following:

- Step 1: Select a hardware configuration (CPU clock frequency, number of cores to be used, etc.) and an integration strategy to be implemented (one-to-one or one-to-all);
- Step 2: Perform the preliminary analysis enabling to obtain the C_i and H_i parameters for each task of the software platform; in the context of our evaluation tests, this step is only done once since the same software case study is exploited for all implemented configurations.
- Step 3: Run the allocation and scheduling CP problems corresponding to the integration strategy selected in step 1, and get the resulting solution;
- Step 4: Configure the hardware platform according to the configuration set in step 1 and the allocation and schedule selected in step 3. Figure 7.3) illustrates some configurations performed in this step.
- Step 5: Compile, boot and run on the embedded target, and check that no deadline is missed at runtime;
- Step 6: Repeat steps 1 to 7 for various parameters values in step 1 (number of cores used, strategy of integration, etc.)

For each of the one-to-one and one-to-all integration strategies, the process from step 1 to 6 is repeated several times since we varied the number of cores to be used on the P4080. The maximum possible number of cores is eight, corresponding to the situation where each partition is allocated alone on a core. Setting the number of cores to be used at runtime as the variable parameters of the tests allows a comparison of results obtained for a given strategy depending on the number of cores used, for instance in order to follow the evolution of the WCET bounds or the difference between computed WCETs and measured execution times. It also enables the comparison of both strategies with each other and the evolution of the results they provide with the number of cores. The corresponding results are presented in the next subsection.

Results

In both strategies, solutions – i.e. a SW/HW allocation and a schedule where all deadlines are met within a complete MAF – were found for up to five cores. Indeed, no solution has been found for six to eight cores for the given software case study and hardware configuration. The timing analyses proposed in this thesis relying on sufficient feasibility and scheduling conditions, this means the allocation CPs for each strategy respectively did not find an allocation in which

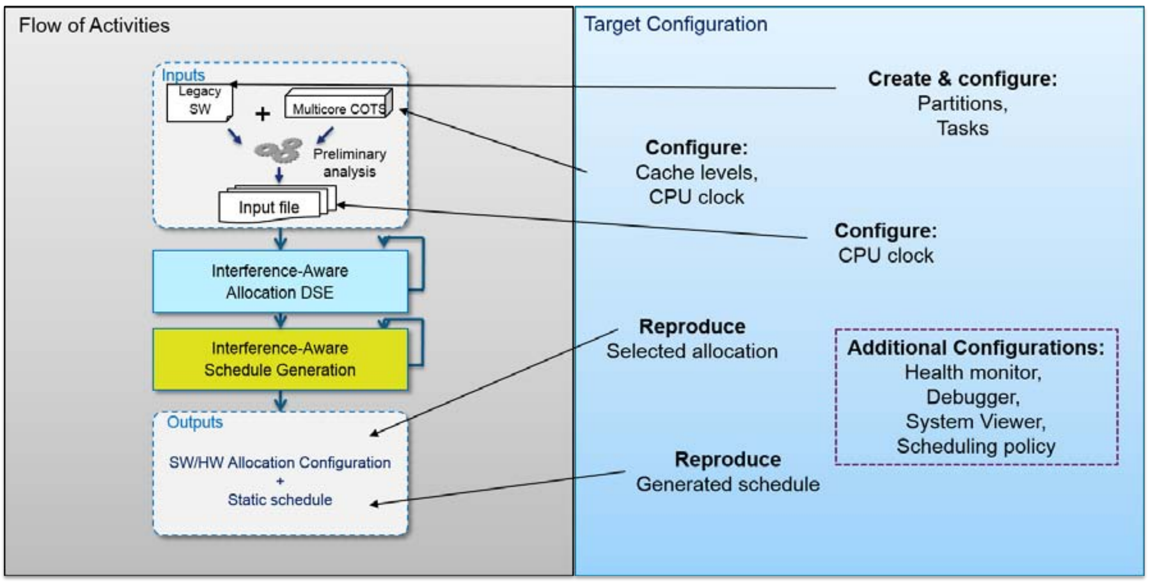

Flow of Activities
Inputs
Legacy SW
+
Multicore COTS
Preliminary analysis
Input file
Interference-Aware Allocation DSE
Interference-Aware Schedule Generation
Outputs
SW/HW Allocation Configuration
+
Static schedule
Target Configuration
Create & configure: Partitions, Tasks
Configure: Cache levels, CPU clock
Configure: CPU clock
Reproduce Selected allocation
Additional Configurations: Health monitor, Debugger, System Viewer, Scheduling policy
Reproduce Generated schedule

Figure 7.3: Target Configuration

all tasks response times remained smaller than their respective deadlines. It is therefore not possible to say whether or not there actually exist feasible allocations for the corresponding configuration.

Every solution outputted in both the one-to-one and the one-to-all integration strategies has been implemented on the target platform: partitions and tasks were allocated to the cores of the P4080 and their memory contexts stored in DRAM so as to respect the allocation of the partitions (resp. tasks) to the memory channels. The schedule generated at the end of the corresponding strategy has been configured as is in the VxWorks653 3.1 development environment. The resulting system configuration project was then compiled and bootloaded onto the P4080-based evaluation platform.

In the rest of this subsection, we compare the results of the execution times measurement to the corresponding WCET upper-bounds computed statically in the corresponding integration strategy implemented.

Safeness of the Computed WCET Upper-Bounds and Generated Schedules

The runtime behavior of each case study evaluated on the P4080 has been observed for several MAFs. In fact, one MAF is 1,6 seconds long, while the observation phase lasted for three to five minutes per run. Graph results are computed using data extracted with the VxWorks653 POS System Viewer for the first 30 seconds for each run, which approximately corresponds to 15 MAFs.

To provide some illustration example, figure 7.4 shows the schedule generated for five cores when applying the one-to-one integration strategy. Figure 7.5 page 181 shows the corresponding schedule resulting for the one-to-all integration strategy. A careful verification of the illustrated schedules revealed that all partitions and tasks deadlines have been enforced, the deadline of a partition being assimilated to its next periodic activation. This is true also in the schedule in figure 7.5 page 181 resulting from the one-to-all strategy, around 800 000 us: some dark blue partition seems to have a time window starting in the frame $[400000; 800000]\,us$, and finishing in the frame $[800000; 1200000]\,us$. When the schedule is zoomed around 800 000 us as done in figure 7.6, one can seen that it does not correspond to a partition deadline violation. In fact, it corresponds to the time windows of partition π_7 respectively belonging to frames $[400000; 800000]\,us$ and $[800000; 1200000]\,us$ being reserved almost consecutively.

Finally, both schedules shown in figures 7.4 and 7.5 have been executed at runtime on the

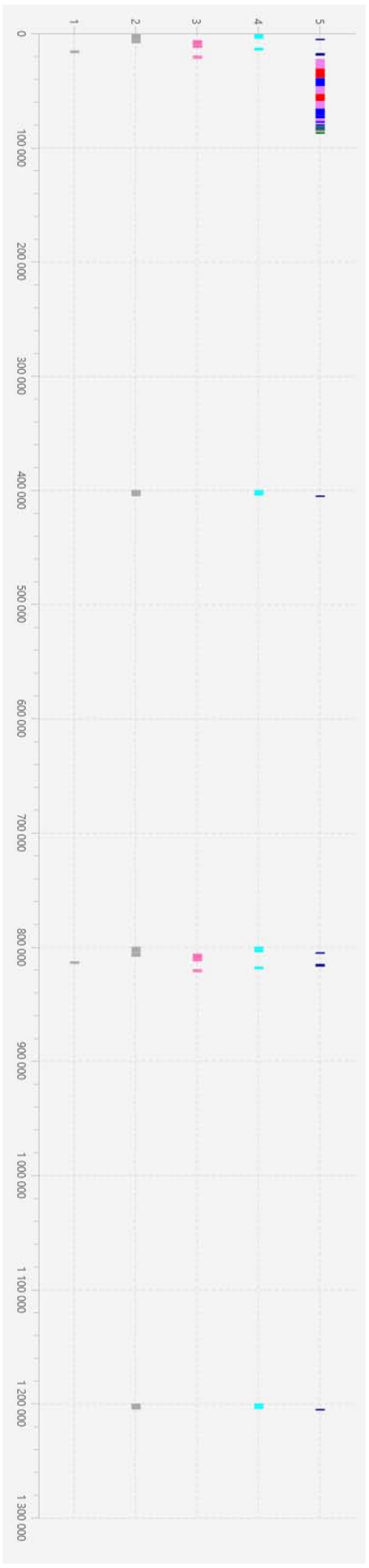


Figure 7.4: Schedule Resulting from Applying the One-to-One Integration Strategy to Integrate the SW Case Study on Five Cores of the P4080

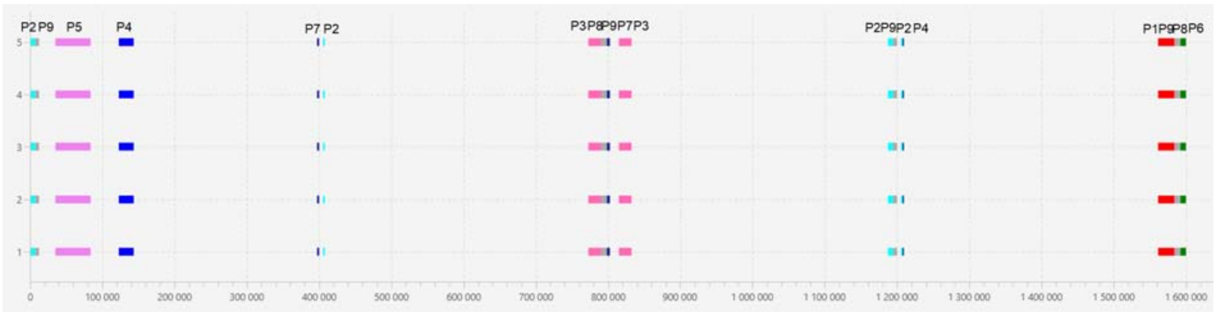
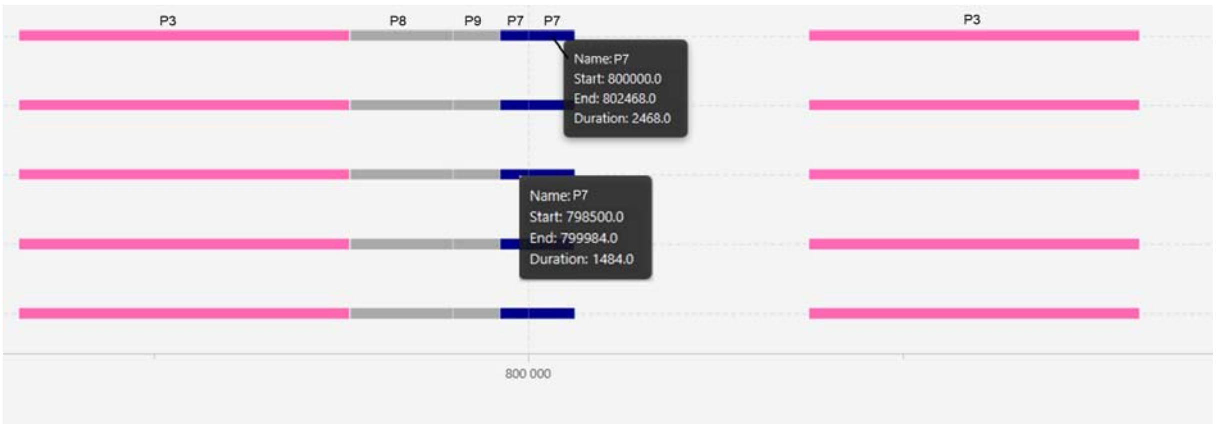


Figure 7.5: Schedule Resulting from Applying the One-to-All Integration Strategy to Integrate the SW Case Study on Five Cores of the P4080

target platform, leading to a respective runtime behavior free of deadline miss.

In general, for all the test configurations evaluated on the target platform, no deadline miss has been observed at runtime either. As such, the evaluation results of this thesis enabled an empirical validation of the safeness of the produced WCET bounds, the timing analysis and in general, the allocation and schedule generation processes.

To conclude, the presence of a significant amount of slack times in the scheduled illustrated in figures 7.4 and 7.5 shows that more than just the nine partitions of the exploited software case study could potentially be allocated to the five cores of the P4080, which is promising for the SWaP reduction capabilities of the one-to-one and one-to-all integration strategies.



Name: P7
Start: 800000.0
End: 802468.0
Duration: 2468.0

Name: P7
Start: 798500.0
End: 799984.0
Duration: 1484.0

Figure 7.6: Parallel Execution of Non-Interfering Tasks

Analysis of the Embedded Pessimism

The exact pessimism of the produced WCET upper-bounds cannot be known for certainty. As such, the difference observed between computed WCETs and maximum measured ETs consist in the maximum possible value of the actual pessimism of the timing analysis approach proposed in this thesis. However, in order to simplify the explanations in the rest of this chapter, and whenever there is no ambiguity, the difference observed between computed WCET upper-bounds and maximum measured ETs is sometimes also referred to as pessimism.

Figure 7.7 illustrates the evolution with the number of cores of the total difference between computed WCETs and measured WCETs in percentage and in each strategy respectively. In the single core case, the computed WCET upper-bounds present around 94.5% pessimism in both strategies. In the one-to-one integration strategy, the percentage of pessimism is stable around that value when the number of cores increases. On the other hand in the one-to-all integration strategy, the maximum pessimism is observed for two and three cores with around 97.5 %. It then decreases to 97 % and 96 % for four and five cores respectively.

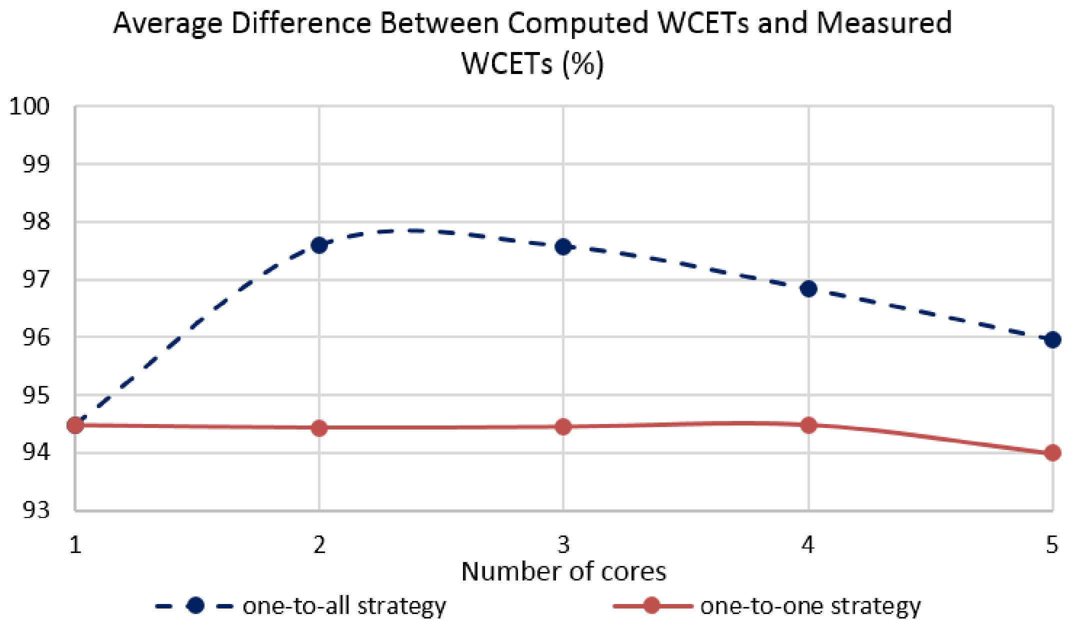


Figure 7.7: Comparison of the Observed Difference between Computed WCETs and the Respective Maximum Measured ETs in each Strategy

Without specific tools such as a probe for detailed trace analysis, it is not possible to examine the cause of this decrease. Multicore interference is not to blame since according to the results files, there is no multicore interference in the schedule, due to never overlapping tasks that share access to the same memory controller. The variability of runtime execution and the randomness of the measured execution times values may be responsible for the pessimism decrease when using five cores instead of four. Another factor ma be the fact that allocations lead to different WCETs depending on the number of context switches occurring on each core: in fact, when dispatching the same number of non-preemptive tasks to a greater number of cores, fewer context switchings occur in one MAF.

An overall observation that can be made from the results displayed in figure 7.7 is the fact that the observed pessimism does not excessively increase with the number of cores. This is an interesting characteristic of our approach, as it could be used to determine the number of cores of a multicore to be used in the final system depending on the percentage of pessimism tolerated by system designers.

Another observation that can be made is the fact that the pessimism value is close to 100%, which may seem to be a bad news. However, to our knowledge, rumor – at least in the academic community – has it that 100 % pessimism remains smaller than the percentage of pessimism that industries are ready to consider adding to measurement-based ET bounds for their future multicore-based systems, which stands around 200%. Such a margin is said to be added to measured maximum ETs with the hope that it will cover the worst-case scenario of interference latencies. Such an approach offers no guarantee since it relies on measurements, contrary to the work in this thesis, which relies on static analysis to produce safe WCET upper-bounds. As such, the safeness of the bounds produced by the timing analysis approach in this thesis, combined to the fact that the obtained pessimism after empirical verification on a real platform remains smaller than the 200 % rumored margin to be considered in future systems, comforts us in the fact that results showing 95 % of measured pessimism do not limit the interest that industries may have in implementing one of the two integration strategies presented in this thesis.

In the WCET upper-bounds pessimism, there already is a pessimism margin that can be blamed on the worst-case execution duration in isolation of tasks depending on the core they are running on C_i^p, computed during the preliminary analysis. This pessimism is not due to the timing analysis approach proposed in this thesis, but rather to the one exploited to compute the C_i^p parameters. Figure 7.7 displayed the total pessimism percentage; it may be interesting to distinguish, in the total pessimism, the percentage due to the C_i^p upper-bounds, from the percentage due to the WCET computational approach proposed in this thesis. To do so, figures 7.8 and 7.9 show the average difference between computed WCETs and the corresponding maximum execution time observed at runtime, in the one-to-all integration strategy and in the one-to-one integration strategy respectively.

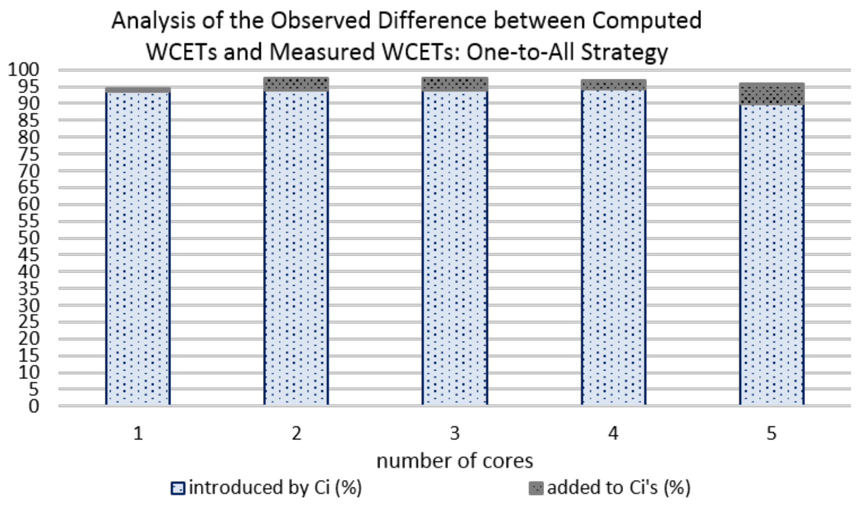


Figure 7.8: Average Difference between Computed WCETs and Measured WCETs in the One-to-All Strategy

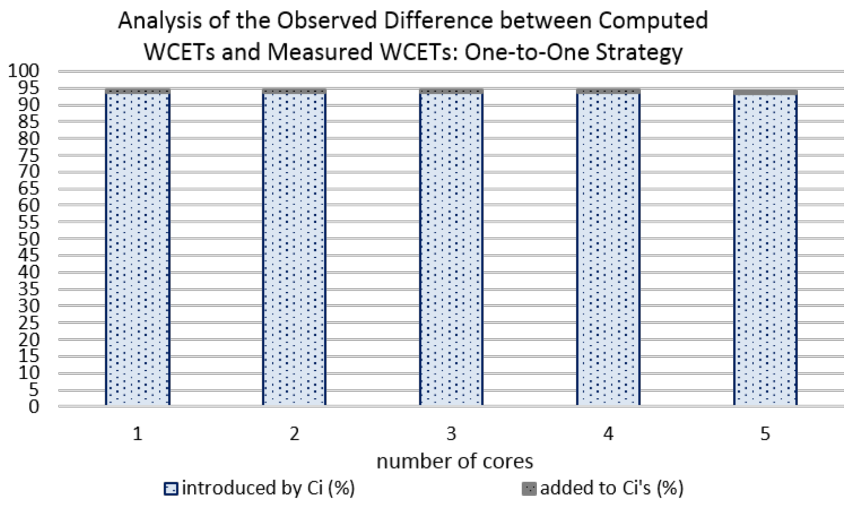


Figure 7.9: Average Difference between Computed WCETs and Measured WCETs in the One-to-One Strategy

In addition, as can be seen in figures 7.8 and 7.9 in the one-to-all and the one-to-one strategies respectively, the pessimism of the C_i^p upper-bounds represents the majority of the observed pessimism percentage. The C_i^p parameters are inputs to the integration strategies proposed in this thesis since the approach to be used for their respective computation is not covered. As such, the pessimism brought by the computation of C_i can be seen as an input pessimism to which an additional margin will be added by our timing analysis equations (see chapter 5).

Knowing this, the graph in figure 7.7 has been plotted with the distinction of the pessimism of the C_i^p upper-bounds. It is interesting to note that, independently from the pessimism of the C_i^p upper-bounds, the approach proposed in this thesis only participates to 0,8 to 6 % of the total pessimism, which is very promising for the contributions of this thesis.

Figure 7.10 presents a zoom in on the evolution of the pessimism percentage due to the thesis work, in order to ease the analysis of that pessimism.

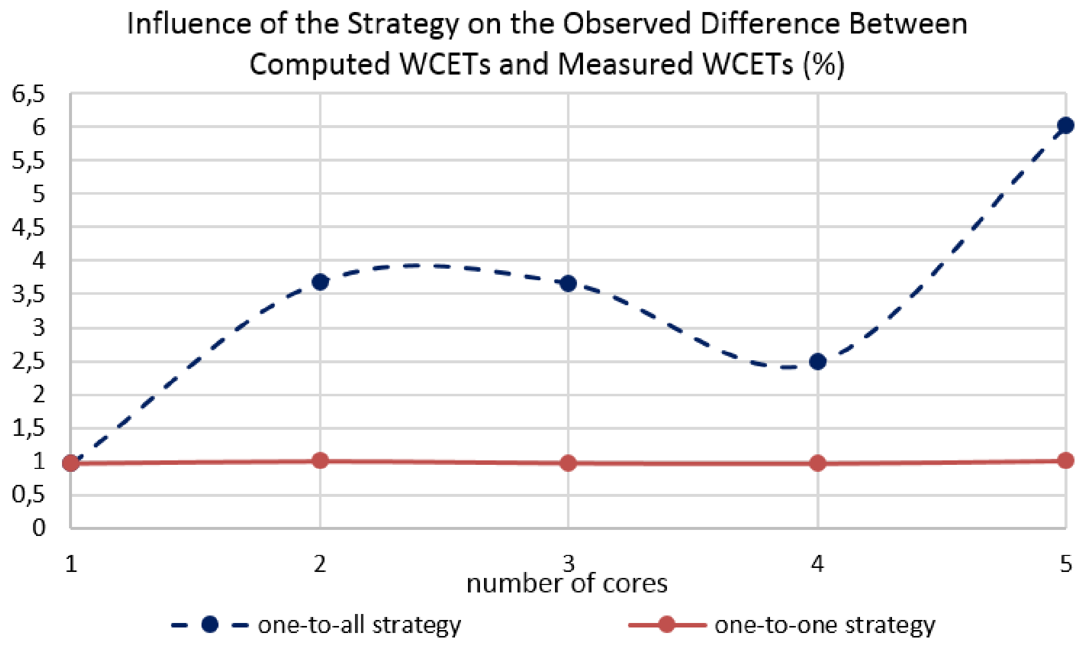


Figure 7.10: Pessimism Percentage due to the Timing Analysis Independently from the Pessimism resulting from the Preliminary – Single-Core – Analysis

In the one-to-one integration strategy, the evolution of the pessimism introduced by our timing analysis has a similar evolution to the total pessimism shown in figure 7.7. Indeed, it remains constant around 0,8 %, independently of the number of active cores. In the one-to-all integration strategy however, the evolution is similar to the total pessimism except for five cores: it increases while the total pessimism shown in figure 7.7 decreases. As such, for five cores, even though the total pessimism is slightly lower than with four cores for instance, the percentage of that total pessimism that is to blame on the timine analysis proposed in this thesis increased. This may seem counter-intuitive when comparing both graphs, and yet figure 7.10 represents another illustration of the dark grey parts of the graph in figure 7.7 by reusing the same data.

The difference between pessimisms (would it be total or exclusively due to the approach in this thesis), along with the decrease while the number of cores increases, might be explained by three facts. The first one is the fact that every data is computed in average, thus showing a general tendency that does not reflect what happens for each core separately.

The second fact is that the P4080 embraces various heuristics for average execution times optimizations that are undisclosed to the public, and which may have impacted the tasks execution at runtime.

The last fact is that upon creation of an integration project, kernel-level management tasks are created, and regularly executed at runtime independently of the tasks defined by the system designer. As such, at runtime, at every clock tick, kernel routines are executed at an hypervisor or supervisor level, contrary to tasks of the SW platform which are user level. As such, these management tasks are able to interrupt a user task to execute a specific routine. Unfortunately, without execution tracing capabilities as would be provided with a probe for instance, the time interval during which the management task was running on the corresponding core will then be seen during our observation phase as a time interval during which the user task itself was running. The corresponding task execution duration is then seen as bigger than it actually was the case. All the while, the schedule computed as output of the one-to-all strategy revealed the absence of inter-core interference, which raises questions about why the task execution is bigger despite the absence of interference. Such situations perfectly illustrate why it is very important for results interpretation to possess cycle-accurate execution trace tools when evaluating works about WCET upper-bounding.

Comparison of the Resulting Interference

As briefly mentioned before, the thesis evaluation has been performed in the absence of an electronic probe to snoop on execution traces. Without such a probe, it is not possible to extract detailed information related to runtime execution traces, and in particular about inter-core interference occurring at runtime. As a consequence, the evaluation on a real target that has been described earlier in this chapter does not display any information on the evolution of interference suffered by tasks with the number of cores used on the P4080. Nevertheless, it may be interesting to study the theoretical evolution of interference in the WCET bounds computed during the allocation and the schedule generation activities. Future work could then be focused on comparing the theoretical evolution of interference with the number of cores for a given software case study, with the actual evolution of interference measured when running on a real target.

We observed the computed interference delays occurring in the schedules generated by the one-to-one and the one-to-all integration strategies for two to five cores. The result is the same for all schedules: there is zero interference suffered by all tasks due to sharing the main memory or the interconnect access. While it may seem to be a mistake at first, it appears to be right after a careful analysis of each corresponding strategy output files schedule. To ease explanations in the following paragraphs we take the example of the schedule generated for five cores as a result of applying the one-to-one integration strategy. The resulting schedule has been drawn in figure 7.4 page 180.

As shown in the figure, the schedule is free of parallel execution of tasks except when they do not use the same memory path, or said differently, when they cannot cause interference delays to each other. Let for instance consider two tasks τ_a and τ_b, respectively allocated to cores p and q and belonging to partitions π_c and π_d. Then for every couple of tasks τ_a and τ_b executed at least partially in parallel at runtime, the corresponding matrix $p2mc$ always shows that π_c and π_d have been allocated to a different memory path.

For instance in the schedule generated for five cores according to the one-to-one integration strategy, a zoom in on the beginning of the corresponding schedule is provided in figure 7.11. The tags "Pi" matches the colors to the corresponding partitions, "Pi" corresponding to partition π_i. As can be seen in figure 7.11, two tasks tagged "T8" and "T28" and which respectively belong to partitions π_3 and π_9, are executed in parallel, respectively on cores number 3 and 2. Upon verification of the output file of the allocation process that preceded the schedule generation phase, π_3 and π_9 have been allocated respectively to the first and the second memory paths. As such, their tasks do not use the same path to the main memory and interfere neither when accessing the main memory nor the interconnect. "T8" and "T28" therefore do not interfere with each other at runtime even though they are scheduled in parallel.

This property has been verified for all couples of tasks executed in parallel on different cores in the MAF schedule. As a result, it is safe to say that the schedule has been configured so that all situation of inter-core interference has been successfully avoided.

Finally, in all schedules resulting from the one-to-one and the one-to-all integration strategies that have been implemented on the target platform, every situation of interference has been successfully avoided as well. No two tasks sharing the same memory path are scheduled in parallel in the corresponding schedules respectively. As a conclusion, the approaches proposed in this thesis for an automated and efficient integration successfully achieved a significant design optimization.

Being able to avoid interference whenever possible when generating a schedule gives an

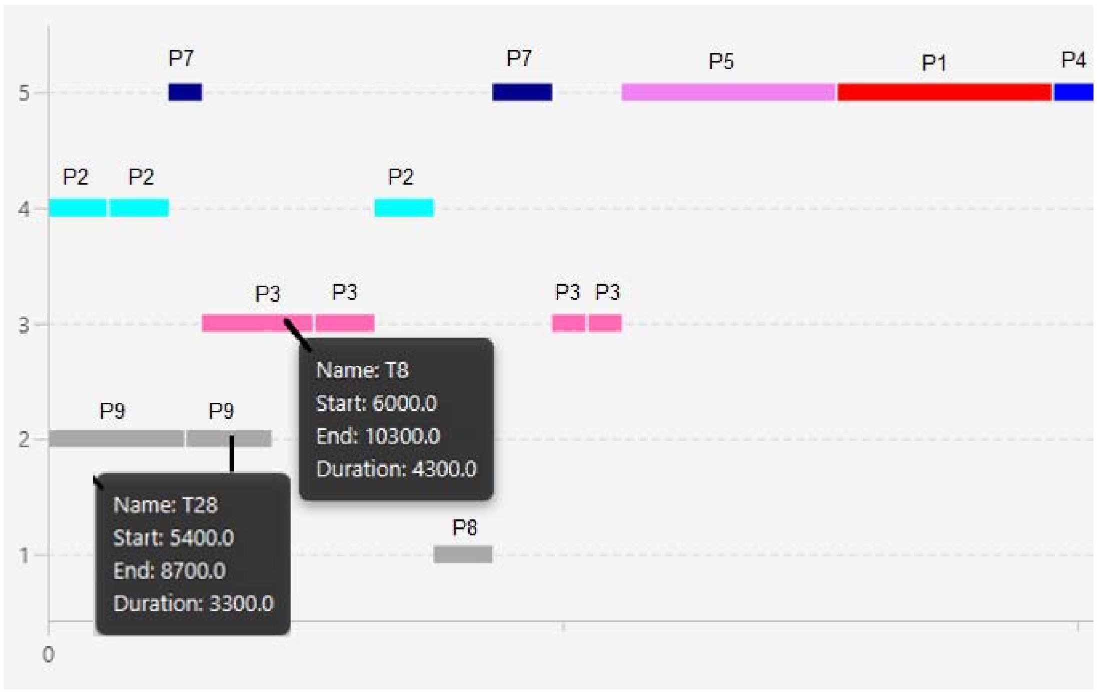


Figure 7.11: Parallel Execution of Non-Interfering Tasks

independence to the pessimism of the timing analysis exploited for the schedule generation. However, such an advantage is a double edge sword, as it can also hide a drawback. As shown in the results of the evaluation on a real platform, the computed WCET upper-bounds are at most 95 % pessimistic, which remains under the 200 % rumored industrial margin. However, this 95% pessimism has been obtained for interference-free schedules. Interference in multicores are said to be significant as a multicore-based system can become four to eight times slower than when it is on a singlecore platform. As such, it is important for future work to determine the pessimism percentage of the timing analysis proposed in this thesis in the presence of interference. In particular, it may be interesting to know whether it will remain under the 200 % rumored margin.

In order for future work to determine the answer to that question, one needs to have a SW case study that has enough workload to force any schedule to implement parallel, interfering executions in one MAF, while still respecting all deadlines despite multicore interference. This corresponds to a delicate trade-off to be found, which has also been the main difficulty of this thesis when constructing a SW case study, as will be explained in greater details in the "Discussions" section at the end of this chapter.

7.4 Theoretical Evaluation

Overview

As mentioned in this thesis, the review of the literature revealed the lack of allocation and schedule generation processes including safe interference or WCET upper-bounding. On the other hand, one goal of this thesis has been to address this lack by proposing an approach in order to take into account interference when computing WCETs and configuring schedules for hard-real time systems. As such, the main goal of the theoretical evaluation is to compare schedules produced by the one-to-one and the one-to-all integration strategies, with schedules produced by a classic, interference-oblivious schedule generation process. Indeed, the comparison should be made with approaches that do not include any multicore interference considerations. We consider such approaches as representing the way the allocation and schedule generation are performed according to the current state of the art, and refer to them as "interference-oblivious".

This section describes the comparison of the schedules resulting from applying the three processes on the same input problem, i.e. the software and hardware platforms. As mentioned before, the goal is to compare the solution returned by applying one of the two strategies proposed in this thesis, with the solution that would have been provided by an interference-oblivious allocation and schedule generation strategy. The comparison of the resulting three schedules is made on the basis of the total workload, the interference percentage in that workload, and the average slowdown suffered by tasks.

Configuration Setup

In the theoretical evaluation, the number of cores $N_C \in [1; 8]$ on which to allocate the software platform described in table 7.1 is the only variable parameter. The maximum possible number of cores is eight, corresponding to the situation where each partition is allocated alone on a core. The result of the integration process – allocation and schedule generation – is either a valid SW/HW allocation and schedule, or the answer that there exists no valid solution for the number of cores considered.

The evaluation process corresponding to such a comparison is the following:

- Step 1: Select a hardware configuration (number of cores to be used, etc.) and an integration strategy to be implemented (one-to-one or one-to-all);
- Step 2: Perform the preliminary analysis enabling to obtain the C_i and H_i parameters for each task of the software platform; in the context of our evaluation tests, this step is only done once since the same software case study is exploited for all implemented configurations.
- Step 3: Run the allocation and scheduling CP problems corresponding to the integration strategy selected in step 1, and get the resulting solution;
- Step 4: Repeat step 3 but with the interference-oblivious approach, and get the resulting solution;
- Step 5: Compare the output of step 3 with the output of step 4;
- Step 6: Repeat steps 1 to 5 for various parameters values in step 1 (number of cores used, strategy of integration, etc.)

For the one-to-all and one-to-one integration strategies, we implemented the CP formulations defined in this thesis using IBM ILOG CP Optimizer [6], on a Dell computer with an Intel i7 2.20 GHz processor with 16 GB of RAM. The same was done for the CPs formulating an interference-oblivious allocation and schedule generation problems respectively.

Results

As mentioned before in this section, the two integration strategies proposed in this thesis are respectively compared to an interference-oblivious allocation and schedule generation process. Four parameters are compared: the existence of solutions for a given number of cores N_C to be used at runtime, the total workload of the system – sum of the respective CPU workloads, the interference percentage, and the average slowdown suffered by tasks in the corresponding schedule.

Existence of Solutions

When using both integration strategies, it is impossible to find solutions for N_C greater than five. This means that all possible allocations on more than five cores of the same multicore processor might lead to deadline violations according to the timing analysis embedded in the integration process. In the one-to-one integration strategy, the absence of solution is stated at the end of the first step of the process, namely the allocation phase; this means that after performing a timing analysis, at least one task WCRT being bigger than the corresponding deadline.

In the one-to-all integration strategy, the first step – step AS1 – produces an allocation for each partition, but in the second step – step MI1 – it is not possible to compute a schedule where all timing requirements are met; this means that after each supplier provided the module integrator with CPU time budgets for their respective partitions, the module integrator cannot define non-overlapping time windows for each partition in one MAF and while respecting partitions periodicity requirements.

On the contrary, in the classic approach, the search always finds a solution, but after analysis of the output interference bounds, the solution appears not to be valid in reality: some bounds lead to actual WCRTs being bigger than the deadline of the corresponding tasks, which invalidates the feasibility of the selected solution. The difference between our strategies and the interference-oblivious strategy being the interference consideration, we can draw the conclusion that the proposed integration strategies embed constraints that efficiently prevent from choosing allocations that, later on during the schedule planning phase, appear to be infeasible. This also implies that our work enables to reduce the time spent during system design. More precisely, our approaches prevent from waisting time readjusting a schedule or an allocation that passed the feasibility test during the allocation search but has been tagged as violating some timing properties later when verifying the integrated platform runtime behavior.

In the end, it is important to note that a system designer that would build a schedule manually or using classic techniques without interference consideration as done in the work of this thesis would have no means for knowing the limit of the number of cores for which a valid schedule can be generated. As such, the approaches in this thesis also enable to further reduce the lead time by shortening the time spent exploring the design space and trials of optimization by parallelizing on as many cores as possible while hoping for the resulting interference to remain small enough for all deadlines to be met.

Workload Evolution

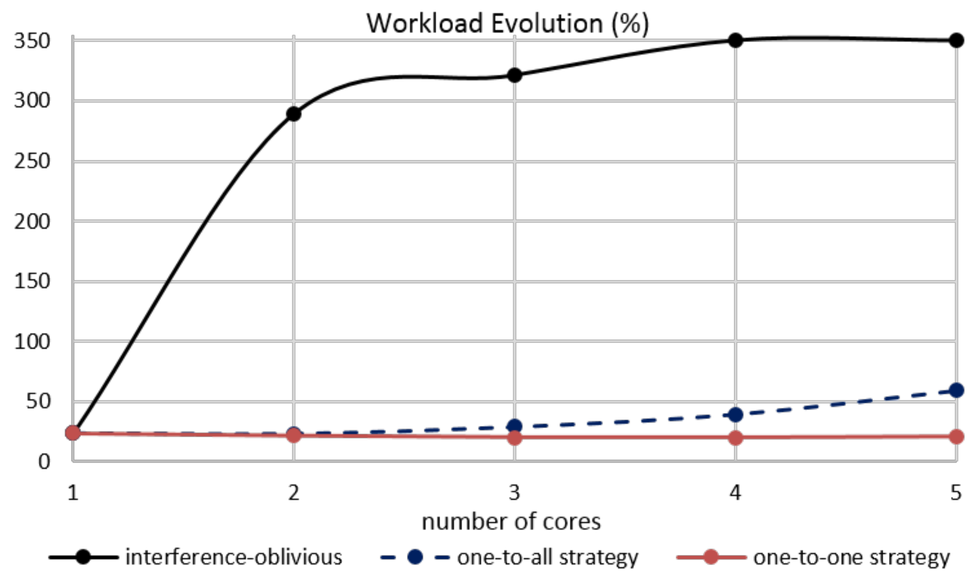


Figure 7.12: Workload Evolution with the Number of Cores (%)

Figure 7.12 shows the workload evolution with the number of cores when applying the one-to-one, the one-to-all and the classic – i.e. interference-oblivious – integration strategies respectively in order to integrate the software case study onto the P4080.

According to figure 7.12, the one-to-one and the one-to-all strategies always return better

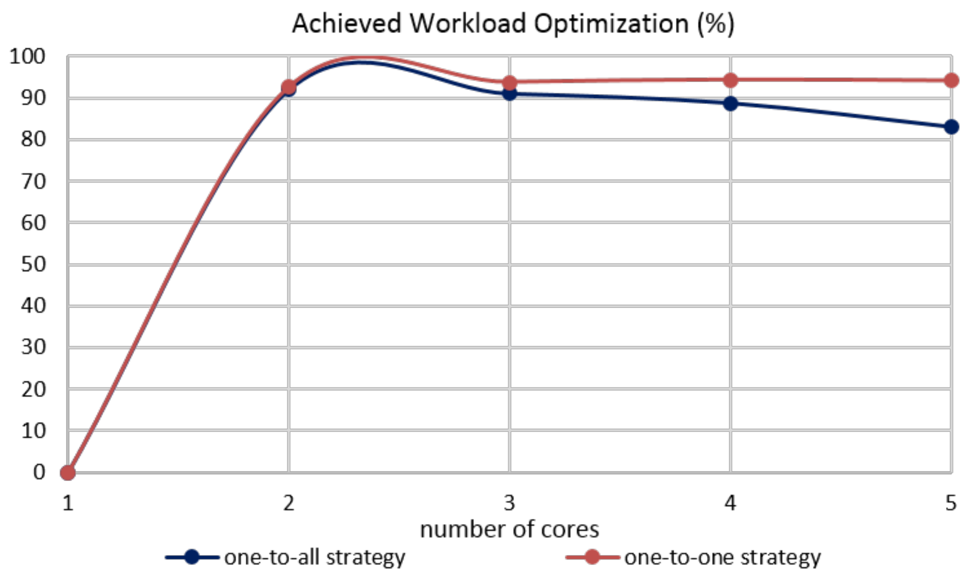


Figure 7.13: Achieved Workload Optimization compared to the Classic Integration Strategy (%)

solutions than the classic approach, with up to 94 The classic, interference-oblivious integration strategy corresponds to the most optimized version of a manual integration process thanks to using CP capabilities: when allocating and generating a schedule manually, without visibility or knowledge of the induced multicore interference, a module integrator is most likely not to end up with a system which has a smaller workload than the one corresponding to the solution obtained after automating the allocation and generation problems resolution using constraint programming. As a result, one can assimilate the curve obtained for the interference-oblivious strategy as the best possible curve obtained by a manual integration process.

When exploiting the interference-oblivious strategy, the workload increase is significant. Although the total single core workload is not very significant – 23,5 %, it already increases to up to 289 % in presence of a second core.

In comparison, in both strategies proposed in this thesis, the total workload of the integrated platform seems more acceptable, with close to no increase in the one-to-one strategy, and an increase from 28 to 59 % in the one-to-all strategy.

The workload evolution with the number of cores in the one-to-one strategy is stable around 23,5 %, which corresponds to the absence of multicore interference in the computed schedules for two to five cores. This has been achieved thanks to an optimized allocation of the SW platform to the cores and the memory controllers, along with a smart selection of start dates in the generated schedule, so that tasks that cannot interfere due to (i) belonging to the same core, or (ii) to not sharing access to the same memory controller, are not scheduled in parallel within a MAF schedule.

In the one-to-all strategy, a second core does not disturb much the workload, whereas the workload increases to 28, 39 and then 59 % for three, four and five cores respectively.

When comparing the one-to-one and the one-to-all integration strategies with each other, for the same case study, the one-to-one integration strategy achieved a 90 to 100 % workload optimization, against 80 to 90 % for the one-to-all integration strategy. As a result, for the given case study and target platform, the one-to-one integration strategy seems to be able to offer better SWaP reduction capabilities than the one-to-all integration strategy. This result was expected, since the SMP-like scheduling approach proposed in the one-to-all integration strategy is by construction constraining all cores to have slack times in their schedules in order to enforce partitions time windows that are common to all cores. However, the achieved workload optimization in the one-to-all strategy remains interesting and not so far away from what could be achieved with the one-to-one strategy. The one-to-all strategy is still able to offer up to 90

% workload reduction at least compared to a manual, interference-oblivious integration process for up to DAL A IMA applications.

Interference and WCETs Upper-Bounds Evolution during the Integration

An interesting observation can be drawn from the previous results presented in this chapter. Despite the usage of static analysis techniques which are said to be overly pessimistic, the workload increase in a multicore environment when compared to a single core one is significant in the classic strategy, whereas the one-to-one and one-to-all strategies do not suffer much from the increase of cores.

This is very encouraging for future work regarding further tightening of inter-core interference and WCET upper-bounds using static analyses. The interference models presented in this thesis are high level. And the higher the level of abstraction, the more important the overestimation of WCET upper-bounds. However, despite relying on high level interference and scheduling models, (i) interesting workload reduction capabilities were observed, and (ii) schedules free of interference were always selected as a result of applying one of the two strategies proposed in this thesis, for all configurations evaluated on a real target as described earlier in this chapter. Indeed, this is encouraging for the future development and improvement of static WCET analysis techniques for multicore environments, static techniques often being qualified as too overly pessimistic to be able to produce exploitable, pragmatic WCET upper-bounds that could e exploited to generate a relevant schedule.

It is important to note that two interference-aware WCET analyses are conducted in each integration strategy proposed in this thesis, and although the resulting schedules are interference-free, the interference bounds computed during the first phases of each integration process were not necessarily null.

When searching for a valid allocation, one must perform a feasibility analysis in order to determine whether a given allocation configuration will later lead to the existence of at least one valid schedule. The analysis is performed while no schedule exist yet, and must therefore rely on the worst possible scenario execution at runtime. On the other hand, in the presence of a schedule, task instances executions are precisely identified on each core thanks to their start dates and WCET set as the time interval reserved to them respectively, several worst-case situations considered during the allocation verification can be ruled out from the schedule verification.

For instance, let's consider two tasks that share access to the same memory controller and are allocated to two different cores. During the allocation verification, one of the worst-case situation considered is the situation where both tasks are scheduled in parallel and therefore interfere with each other at runtime. In such situation, they are seen as competing with each other in order to access the main memory since they are allocated to the same memory controller. On the other hand, lets consider that these two tasks are never executed concurrently in the schedule generated later on during the schedule generation phase. During the schedule verification, one has knowledge about the fact that the tasks respective execution time intervals are never overlapping in time, and therefore that they do not interfere. As a consequence during the schedule verification, the two tasks will never be considered as causing interference delays to each other, contrary to the allocation verification. Considering such a situation for every couple of tasks allocated to the same memory controller but not the same core, one can draw the conclusion that the WCET upper-bounds computed as a result of the allocation verification will always be bigger than the WCET upper-bounds computed as a result of the schedule verification.

To sum up, in all schedules reproduced on a real platform, interference delays have been reduced to zero in the last steps of the one-to-one and the one-to-all integration strategies, although the corresponding delays computed as a result of the allocation verification were non-null. As a conclusion, it is safe to say that the optimization features introduced in the CP formulations presented in this thesis successfully managed to refine the computed interference delays and WCET upper-bounds, from the early feasibility analysis performed in the first step of the integration process, to the schedulability analysis performed in the last step of the one-to-one and one-to-all integration strategies respectively.

Although interference delays computed in feasibility analysis when allocating, were significant, the resulting, respective schedules are free of interference. As a result, one can draw the conclusion that as long as the allocation phase determines that there exists feasible schedules, no matter the corresponding interference bounds. Since the generated schedule will always exhibit as less parallel execution of potentially interfering tasks as possible, the schedule will be the result of a minimization of inter-core interference as much as possible.

This is undeniably an advantage of the proposed integration strategies. However, it is also a double edged quality. In fact, as schedules are generated only with the condition that the allocation phase found feasible configurations based on the timing analysis proposed in this thesis, if that analysis produces too pessimistic WCRT upper-bounds, then there is a risk that the result of the allocation verification is the absence of solution in the first steps of the integration process, long before the step where schedules are generated.

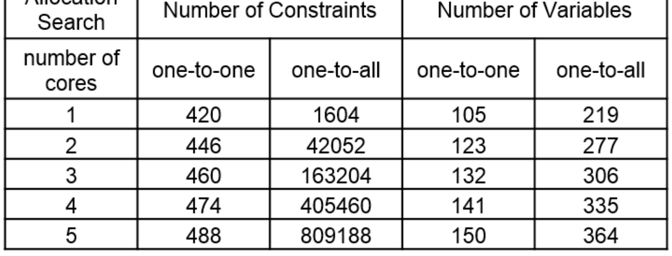
Evolution of the CP Complexity

Tables 7.14 and 7.15 present the evolution of the number of variables and constraints of the CP formulation for the allocation and schedule generation CPs respectively with the number of cores.

For the allocation CP problems, the number of variables and constraints increase with the number of cores; regarding the schedule generation CPs, they remain respectively constant independently form the number fo cores. Every increase of variables and constraints is linear with a small slope, except for the allocation CP in the one-to-all integration strategy. This is in line with the experimentations, where the allocation search has been the one taking the longest time before finding at least one solution.

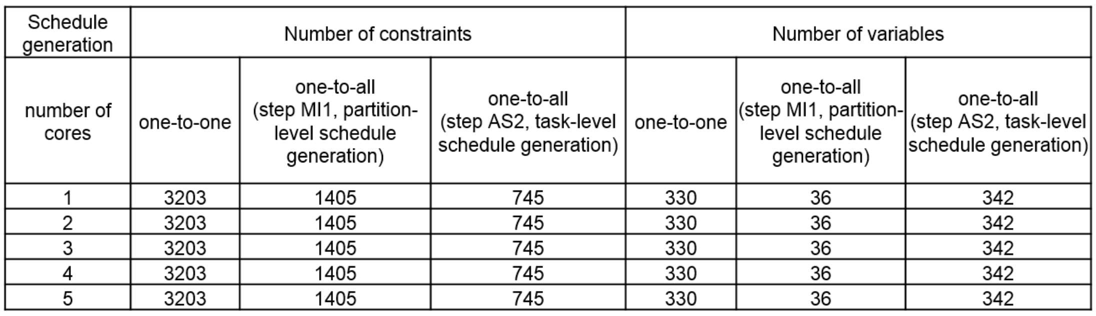
On the other hand in the one-to-all strategy, one has to keep in mind that for the sake of simplifying this thesis experimentation, we solved the allocation problem for all partitions altogether in one allocation CP: i.e. one solving process is launched in order to find an allocation to all nine partitions and their twenty-nine tasks in total. If used on an operational system, the allocation search would be performed on each partition separately: i.e there would be one solving process launched per partition, each process trying to find an allocation for the three or four tasks of each partition. The latter situation is likely to move from an exponential growth of the number of constraints, to a linear increase with a small slope similarly to all other CPs.

In general, the allocation and the schedule generation problems being NP-hard resolution problems, it is very interesting to note that we manage to escape the exponential complexity explosion for all CP except the one-to-all allocation search CP. In an actual implementation of one of the two strategies, in the context of a real industrial system, one can easily imagine oneself reusing the proposed CP formulations, as each finds more than one solution in less than three minutes.



Allocation Search	Number of Constraints		Number of Variables	
number of cores	one-to-one	one-to-all	one-to-one	one-to-all
1	420	1604	105	219
2	446	42052	123	277
3	460	163204	132	306
4	474	405460	141	335
5	488	809188	150	364

Figure 7.14: Evolution of the Number of Variables and Constraints for the Allocation Search



Schedule generation	Number of constraints			Number of variables		
number of cores	one-to-one	one-to-all (step MI1, partition-level schedule generation)	one-to-all (step AS2, task-level schedule generation)	one-to-one	one-to-all (step MI1, partition-level schedule generation)	one-to-all (step AS2, task-level schedule generation)
1	3203	1405	745	330	36	342
2	3203	1405	745	330	36	342
3	3203	1405	745	330	36	342
4	3203	1405	745	330	36	342
5	3203	1405	745	330	36	342

Figure 7.15: Evolution of the Number of Variables and Constraints for the Schedule Generation

7.5 Certification Compliance Evaluation

In 2014, the Certification Authorities Software Team (CAST) submitted a first position paper regarding multi-core processors [3]. This position paper sets the preliminary objectives for multicore usage in avionics systems; they have since been refined into a list of ten objectives to which one must show compliance to certification authorities.

The objectives are listed one by one before being discussed with relation to the thesis outcomes.

Software Planning One objective has been defined by the EASA regarding the software planning. The objective mainly relate to the applicant's software plans or other deliverable documents.

MCP_Planning_1: this objective asks for a clear identification in the certification documents of:

- The multicore processor to be used;
- The number of active cores;
- The software architecture to be used and all the software components to be hosted on the multicore processor;
- The architecture nature, namely, whether the multicore will host IMA applications or whether it will rather be used in a federated architecture;
- The degree of partitioning to be implemented, namely robust or non existent;
- The methods and tools to be used in order to develop and verify each application software component individually so as to comply to the EASA software guidance.

This objective is general and easily covered by documenting high level configuration choices. For the last item, one may identify the integration strategies proposed in this thesis as the means to be used in order to verify the enforcement of non-functional properties of the system.

Multicore Resources Planning and Setting Three objectives are defined in this category.

MCP_Resource_Usage_1: this objective demands for the applicant to have determined and documented the multicore processor configuration settings enabling the software and hardware embedded on the processor to satisfy the functional, performance and timing requirements of the system.

The thesis goal is to meet the timing requirements of a system to be implemented using multicores. As such, the overall integration strategies proposed in this thesis, along with the corresponding data and parameters values resulting from each step inside each integration strategy respectively, fulfill this objective.

MCP_Resource_Usage_2: this objective asks for the applicant to have planned, developer, documented, and verified a means that ensures an appropriate means of mitigation in the event of any of the critical configuration settings of the multicore processor being inadvertently altered.

This objective does not fall into the scope of this thesis, which is focused on non-functional aspects of a multicore-based system rather than ensuring robustness to unexpected alterations of the multicore platform.

MCP_Planning_2: this objective requires out of the applicant that the plans and documents forwarded to certification authorities:

- Provide a high-level description of how shared resources in the multicore platform will be used and how the applicant intend to allocate and verify their usage;
- Identify any hardware dynamic feature.

While the second bullet is out of the scope of this thesis, the first bullet is covered by the schedule generated as output of the strategies proposed in this thesis.

Interference Channels and Resource Usage Two objectives directly relate to interference due to shared resources.

MCP_Resource_Usage_3: this objective covers the identification of interference channels able to potentially affect software applications at runtime. In particular, it is requested out of the applicant to handle any interference channel at any time during system design, and to perform an interference analysis whenever deemed mandatory by the safety analysis, in order to meet this objective.

This thesis takes into account interference in multicores. To do so, it relies on the prior identification by the applicant of interference channels according to the usage they intend to do of the multicore platform. In the assumptions and the current state of the thesis, the only interference channels identified are the main memory and the interconnect. No I/O or shared cache levels are considered. In this configuration, an interference analysis is proposed, as requested out of objective **MCP_Resource_Usage_3**. To show compliance to this objective when more than these two interference channels can be identified (for instance, shared cache levels that the applicant intends to exploit at runtime), one must also update the thesis contributions in order to take them into account in the interference analysis.

MCP_Resource_Usage_4: this objective requests from the applicant to have allocated the resources of the multicore processor to the software applications, and verify that the demands for the resources do not exceed the available resources at runtime.

The thesis being focused on safe WCET upper-bounding, and embracing WCET upper-bounding annalysis in the allocation and schedule generation process, it is safe to say that the thesis outcomes comply to objective **MCP_Resource_Usage_4**.

Software Verification This section contains two objectives.

MCP_Software_1: this objective asks for the applicant to verify that the software components function correctly and has sufficient time to complete their execution when all the hosted software is executing in the intended final configuration.

While the functional correctness of software to be embedded on a multicore is out of the scope of this thesis, guaranteeing that each software application has sufficient time to complete its execution at runtime is covered by construction of the schedule in both integration strategies proposed in this thesis.

MCP_Software_2: this objective covers mandatory testing activities related to data and control coupling between the applications hosted on different cores. As such, it is out of the scope of this thesis.

Error Detection and Handling, and Safety Nets Error detection and handling is covered by one objective.

MCP_Error_Handling_1: this objective specifies that the effect of failures that can occur have been identified, implemented and verified for the mulciore platform to be used. The corresponding means to detect and handle these failures in a fail safe manner must also be provided by the applicant.

Error detection and recovery is out of the scope of this thesis.

Reporting of Compliance with the Objectives of this Document

MCP_Accomplishment_Summary_1: this final objective mentions that the applicant must explicitly demonstrate how the are complying to each of the previously mentioned objectives in the deliverables to be forwarded to certification authorities for approval.

This objective is general in the sense that it asks for argumentation and traceability of the objectives enforcement. As such, regarding the scope of the thesis and the related objectives, the proposed contributions help an applicant to show compliance to objective **MCP_Accomplishment_Summary** for instance by displaying how every activity in the integration process is conducted, and by providing detailed information on the way allocations and schedules are configured.

7.6 Discussions

In this section we review the relevance of the evaluation tests with regards to the real target platform and test environment. In particular, we mention the limitations encountered during the evaluation phase, due to the necessity to stick to assumptions of the proposed contributions. We also discuss about the shortcomings that occurred during the tests environment setup and which are due to unexpected difficulties.

Conclusions of the Evaluation

When applying the one-to-one or the one-to-all integration strategy on a software case study and a real multicore platform with an avionics RTOS, results have shown the absence of deadline miss at runtime after a significant observation period. As such, the evaluation results of this thesis enabled an empirical validation of the safeness of the produced WCET bounds, the timing analysis and in general, the allocation and schedule generation processes.

As mentioned earlier in this chapter, the exact pessimism of the produced WCET upper-bounds cannot be known for certainty. As such, the difference observed between computed WCETs and maximum measured ETs consist in the maximum possible pessimism value. According to the tests presented in this chapter, the difference between computed WCETs and maximum measured ETs remained stable around 95% in both strategies and with a number of cores from two to five, among which approximately 88 - 89 % is brought by the single-core maximum execution duration upper-bound, and the other 6 - 7 % is due to the rest of the response time computational model proposed in this thesis. Knowing the rumor has it that industries are considering adding a 200 % margin to measured maximum ETs when running on multicores, such results are promising for future time critical systems since: (i) it remains smaller than the envisaged margin, and (ii) it is based on statically computed – therefore safe – WCET upper-bounds, which are thus reliable upper-bounds contrary to measurement-based WCETs. In the future, more refined single-core analysis techniques may even help curb the 89 % obtained in the evaluation tests.

To sum up, the evaluation results showed the safeness of the produced WCET bounds, and the reliability of the generated schedules where every deadline is met at every MAF at runtime. This holds for both integration strategies. In addition to be based on a reliable timing analysis, each strategy proposed in this thesis can be used depending on the considered software platform. The one-to-all integration strategy is relevant for up to DAL A legacy IMA software, since each application can be analyzed separately from each other, and it enforces a robust time and space partitioning as required out of IMA systems. On the other hand, the one-to-one integration strategy fits up to DAL C IMA applications for which the necessity of a robust partitioning has not be stated. It can also be used to integrate multi-partition DAL A applications, i.e. systems where the considered multicore is exclusively running software corresponding to that only application. Finally, the one-to-one strategy can also be used for non IMA software, for example to port software that was, as of today, integrated according to a federated architecture.

Implementation with Regards to the Assumptions

This subsection discusses the limitations linked to the way we matched this thesis assumptions with the real environment of the evaluation platform.

The timing analysis proposed in this thesis takes into account one cache level only. On the other hand, the multicore chosen for the evaluation of this thesis presents a three-level cache hierarchy. As such, before launching any test, we deactivated the L2 and L3 caches available on the processor in order to match the assumptions of this thesis when performing the tests described in section 7.3.

In addition, the preliminary analysis for the extraction of single-core parameters C_i and H_i for each task τ_i has been done with aiT [2]. Such an analysis requires a model of the target processor. However, the target processor is not a single-core processor, and as such, no architectural model exists for static code level WCET analysis. As a workaround, we used the model of a similar core, the MPC7448C, instead of the model of the P4080. Although the two architectures are similar since they present several features in common, such a workaround raises the question of the relevance of the single-core analysis with respect of the accuracy of the multicore architectural model exploited.

Finally, the worst-case overhead of a context switch is unknown since the detail of operations performed at runtime during a context switch are hardware-dependent, and represent information that is usually undisclosed to the public. As such, the value of C_{SW} chosen for the evaluation tests relied on a couple of preliminary runs on the target platform in order to derive an upper-bound of all observed context switch durations.

Shortcomings

This subsection discusses the shortcoming encountered during the evaluation tests. This does not cover limitations due to the assumptions of this thesis, but rather unexpected difficulties occurring during the tests phase, and the limitations of the available test means.

This thesis is focused on the safe upper-bounding of inter-core interference that can occur at runtime, in advance at design time. As such, an important activity of the evaluation on a real target is the measurement of actual inter-core interference experienced by tasks at runtime. To do so, a probe is required. However, no such probe has been made available for this thesis evaluation purposes until the last weeks of the PhD. This represents one of the main limitations of the thesis tests, since it prevented precise interference measurement through detailed analysis of runtime traces.

In order to be used, the probe has to be configured for the target hardware platform it is going to snoop on and extract execution traces information. In particular, additional configurations specific to the exploited RTOS must be performed, in order to be able to distinguish which execution traces information corresponds to which task and partition at runtime. As such, configuring the probe requires some guidance from both the probe and the RTOS vendors at least in order for all configuration steps to be performed accordingly prior to the tests.

The absence of probe did not enable the test setup to be able to extract interference latencies experienced by tasks. The beginning and end of tasks could be traced thanks to VxWorks653 3.1 System Viewer [14], but no details on the amount of time in each execution corresponded to some interference. On the other hand with a probe, it would have been interesting to observe the evolution of multicore interference with the number of cores and the memory usage intensity on the target platform.

Another shortcoming encountered during the tests setup were difficulties related to configuring the target platform. Besides the usual amount of time spent for first manipulations of the platforms and training, additional time has been spent trying to find workarounds to some unexpected shortcomings.

For instance, an interesting experimentation of the thesis would have been to configure a different CPU clock for each core used in the P4080. To do so, beside understanding of the complex clocking system of the P4080 evaluation board, one would have to define a new board support package for the RTOS with the updated CPU clocks frequencies. Indeed, it is currently not possible to easily edit the value of a core clock, as it is the output of a PLL involving a platform clock, a core cluster clock et finally, a clock per core. A limited set of values can be possible according to the platform datasheet, and one would also have to check by themselves that the values chosen for each intermediary clock is valid according to the values given in the datasheet. To sum up, the complexity of the clocking system and the representation of such a system in the platform and RTOS respective configuration frameworks made it impossible within the time line of the thesis to experiment with different CPU clock frequencies.

Another shortcoming was the fact that tasks scheduling policy could not be changed to "user-defined". This had an impact on the enforcement of the schedules computed as outputs of the corresponding integration strategy implemented. In fact, the task-level scheduling policy was fixed priority preemptive. Static schedules can be defined at partition level, but not at task level. In order to still be able to enforce a static schedule at task-level, a non-negligible amount of time has been dedicated to finding a workaround approach, which was the following. In the RTOS development framework, we defined one partition per task, so that the start and duration specified for each time window of that partition would correspond respectively to the start of the corresponding task instance and its associated WCET. This lead to other concerns, such as ensuring there was enough memory space to store the context of each partition for instance.

Finally, the evaluation of the integration strategies proposed in this thesis required some software case study. The case study must be representative of a legacy IMA software as can be found in current IMA systems. However, any software produced for aircraft systems is considered confidential, and access to it is therefore restricted and carefully traced. As such, it was not possible to get actual legacy IMA software, let alone information on classic non-functional properties in general in current IMA software. Without such information, a non negligible amount of time has been dedicated to the specification and development of a case study representing an IMA software platform.

The resulting software has then been exploited during the evaluation phase for the tests to represent a proof of concept, and hopefully convincing of the relevance of exploiting the thesis outcomes in future aircraft systems. In order to do so, an important point has been made in generating a case study that was interesting enough in both the one-to-one and the one-to-all integration strategies; indeed, it was important to have a case study for which solutions of allocation and scheduling existed for more than just two cores, in both strategies, for the results to be interesting in terms of feedback of the thesis contributions.

However, it is a known problem of the real-time community that the software case study generation procedure can bias the results of a given schedulability analysis [30]. As such, the implemented case study generation approach presented earlier in this chapter cannot be exploited for evaluating the schedulability ratio as is usually done when evaluating a new scheduling algorithm. To do so would have required to spent more time on the unbiased generation of case studies as proposed by Bini and Buttazzo [30]. However, the IMA architecture imposing a second level of scheduling, the work of Bini and Buttazzo cannot be exploited in the evaluation of this thesis, which therefore would have lead to further time spent on how to derive an IMA version of the unbiased case study generation process proposed in [30].

Eventually, because of the amount of time spent designing a case study interesting enough in both strategies for the evaluation to be interesting, no intra- and/or inter-partition communications have been implemented.

7.7 Summary

This chapter presented the evaluation of the work presented in this thesis. The implementation of the test environment was presented, and we explained how we constructed a software case study for the purposes of the thesis evaluation. The tests performed were then presented, and the corresponding results commented. In addition, the outcomes of the PhD was assessed with regards to certification compliance, and its applicability to industrial, real systems. Finally, a ”Discussions” section summarized the shortcomings of the evaluation phase, with a clear identification of the limitations due to matching the test environment to the thesis assumptions, or to unexpected shortcomings occuring during the environment setup.

Chapter 8

Summary and Perspectives

8.1 Summary

This thesis addressed the challenges faced when considering the usage of multicore COTS platforms in future avionics systems embedding IMA applications. These challenges have been discussed in the context of an industrial usage, with strong certification constraints.

Motivated by the drawbacks of related works, the focus of this thesis is on the minimization of cost, lead time and adaptation from current engineering and industrialization processes involved in avionics systems design cycle. Indeed, the state of the art related to multicores and hard-real time systems always implied either:

- Proposing custom hardware designs which implied high costs for aircraft manufacturers;
- Redesigning IMA legacy software in order to be parallelized on the cores of a multicore system, which implied having found a multicore-adequate methodology to do so, along with high efforts of adaptation from suppliers but also engineers responsible for gathering all necessary information to convince certification authorities of the soundness and safety of the corresponding new design process;
- Approaches not suited to the IMA concept, which therefore implied getting rid of the IMA concepts.

The objectives of the thesis were the following:

- Minimize as much as possible changes to the current industrialization process for IMA systems implied by the proposed contributions;
- Focus on hardware-independent approaches so that it is not specific to one multicore chip or manufacturer;
- Be compliant to current certification requirements and enforce the IMA, robust partitioning and incremental verification concepts as much as possible.

Based on these objectives, the so-called one-to-one and one-to-all integration strategies, along with the associated WCET, feasibility and schedulability analyses have been developed.

The one-to-all integration strategy is based on a static SMP-like approach where at runtime only one application is scheduled on all cores of a multicore platform. The entire multicore platform is seen as one CPIOM (Core Processing Input/Output Module). Scheduling only one application on all cores at runtime preserves the concept of robust partitioning, and therefore enables the strategy to remain compatible with strong certification requirements for IMA systems. The one-to-all strategy can be used for up to DAL A IMA applications.

The one-to-one integration strategy relies on a static AMP approach where each core is considered as a separate CPIOM at runtime. The one-to-one integration strategy enables to achieve maximum SWaP reduction using our analysis techniques. However, the price to pay for such optimized systems is the absence of a strong time partitioning in the resulting system. As a consequence, this strategy can be applied for non-IMA software, or in the context of multi-partition IMA applications, as long as all partitions allocated to the multicore platform belong to the same application.

To cover the verification of the system runtime behavior, both strategies embrace: (i) a feasibility analysis exploited during the allocation configuration phase, and (ii) a schedulability analysis exploited during schedule configuration. The feasibility analysis is based on a derivation of the response time analysis to fit IMA architectures and multicore platforms. An interference model has been proposed in order to produce a safe upper-bound of inter-task interference delays suffered at runtime in the worst-case situation by each task. The schedulability analysis reuses the interference model, along with the knowledge of the other cores schedules in order to compute WCET upper-bounds for each task instance in one hyperperiod and verify that all corresponding deadlines and non-functional requirements such as precedences are met in the computed schedule.

Finally, all steps of each integration strategy are expressed in the form of CP (Constraint Programming) problems, in order to enable the automation of the allocation, schedule generation and timing verification activities. Besides automation, exploiting CP techniques also enables to save time and effort during the system design cycle. It also enables to insert optimization features via objective functions in order to smarten the configuration selection, which would have been very difficult to impossible to do manually.

The outcomes of the thesis have been evaluated on a COTS multicore platform that has been considered for usage in avionics systems at its apparition, the NXP P4080 [9]. CPs have been implemented using IBM ILOG CP Optimizer [6]. The software aiT Analyzer [2] has been used for performing single-core analysis, and Wind River VxWorks653 Multicore Edition RTOS [14] has been used in order to enforce schedules as specified in the output of the corresponding CP in each strategy.

The evaluation of both strategies revealed their soundness since no deadline missed has been detected after observation of the system runtime behavior for several MAFs (MAjor Frames) and in various configurations.

The design optimization gain when using CP has been observed when comparing the outcome of the one-to-one and one-to-all integration strategies with the classic, interference-oblivious allocation and scheduling CP problem. In particular, the experimentations have shown the efficiency of the proposed approach. Each CP is solved within seconds, compared to days or weeks when the allocation and schedule generation are done manually. As such, our approach brings time and effort gains when designing a system, without overlooking certification requirements thanks to safe timing behavior verification. In addition, using our approach suppresses the risk of late detection of non-valid configurations. It is also easier for system designers to try different configurations, the valid and invalid ones clearly and rapidly being exhibited by the CP. Eventually, another novelty brought to system designers by our approach is the gain of visibility over the impact of interference and resource usage intensity on the existence of valid schedules. The integration process is now more flexible, as the gain of time and effort gives more room to system designers to perform extra verification, try new combinations of allocations and further SWaP reduction, etc.

To sum up, the objectives towards certification-compliant multicore-based IMA industrial integration process has been achieved. Secondary objectives of design optimization and time-to-market reduction have been achieved as well through the exploitation of CP techniques to express the problem at hand.

In addition to the functional aspects of the contributions of this thesis, the generality of the proposed approaches has been verified. First, one may notice the independence of the strategies from the exploited hardware platform and RTOS. Second, it is possible to modify the interference models implemented in the WCET analysis in order to exploit other state of the art interference analysis models instead. This is allowed both by the composability of the response time analysis, the safety-net mechanism relying on HM capabilities to handle anomalies as faults, and the mathematical representation thanks to CP techniques.

Finally, both the one-to-one and the one-to-all integration strategies rely on a methodology and timing analysis metrics that are made available in this thesis. In particular, the one-to-all integration strategy also respects current certification requirements that are mandatory for acceptance by certification authorities. As such, this strategy can be immediately exploited in an industrial process as a basis of a future multicore engineering process.

8.2 Conclusions

Models Optimization to Avoid State Space Explosion

Boolean Matrices instead of Vectors of Integers. As presented earlier in this chapter, in order to represent the allocation of tasks and partitions to cores and memory controllers, boolean matrices are chosen instead of vectors of integers (a and $p2mc$ in the one-to-one integration strategy, na and $t2mc$ in the one-to-all integration strategy). Vectors of integers could have been used instead, the i^{th} value in the respective vector corresponding to the index of the allocated core or memory controller.

The justification of such a choice is the convenience of using boolean matrices when expressing some of the constraints of the respective CPs. In fact in such constraints, the corresponding equations use sums in which only the designated elements are non null. If vectors of integers were used instead, the sums would contain a condition on the corresponding integers in order to filter the elements that should be evaluated in the sum.

For example for the core allocation in the one-to-one integration strategy, if a vector was used instead of the boolean matrix a, the vector would be of size N_P, and each element of the vector would be an integer corresponding to the index of the core the corresponding partition is allocated to. The same changes would be true for the memory path allocation.

Such changes would then reflect on the complexity of equations of the allocation and schedule generation CPs defined in this thesis. Indeed, an example illustrating the convenience brought by boolean matrices instead of vectors or integers for the allocation matrices can be found when expressing equations reasoning at partition level – i.e. for each partition – but focusing on task-level information (for instance, tasks of a partition that are responsible for message sending/writing actions) and/or allocation-specific information (for instance, memory controllers allocated to a given task allocated to a given core of a given partition).

With boolean matrices – i.e. terms equal to one if a given condition applies to them, and zero otherwise – one can express one equation for all partitions and tasks inside them respectively, by using sums, as will be done in the next chapters. The boolean elements of the matrices corresponding to tasks to which the condition does not apply will be equal to zero by definition of these matrices, therefore enabling an automatic selection of all other tasks to which the condition actually does apply.

In contrast, with integer vectors instead of boolean matrices, the same relation may correspond to a more complicated equation. In order to retrieve the same information, several "*for each .. such that*" nested loops are likely to be necessary.

Some "for each" conditions may be necessary with boolean matrices as well; however, having boolean matrices representing instead of vectors truly helps reduce the complexity of the CP to be solved, but also save time during the CP solving process and therefore, the system time-to-market.

Another example is the computation of the cores MAF in the one-to-one integration strategy. For each core p, MAF_p is defined in equation (4.19) as the maximum value of the periods of the partitions allocated to core p. To do so, the selection of partitions is done using the a_{pi} term multiplied by the corresponding partition period P_i, consequently P_i's value if π_i is not allocated to core p, and therefore ruling it out of the choice of the maximum value. If an integer vector $(corealloc_1, ..corealloc_{N_P})$ was used instead of the boolean matrix a, excluding partitions of other cores would have been done by using the following equation instead of equation (4.19):

$$\forall p \in [1, N_C], MAF_p = \max_{\substack{\pi_i \in \mathcal{P}_{\rangle} \\ corealloc_i = p}} (P_i) \tag{8.1}$$

In the allocation CP, $corealloc_i$ would have been an unknown variable. As such, MAF_p would be expressed using a "max" relation, conditioned by variables of the corresponding CP.

However, it is forbidden to define a sum depending on a variable even in such intermediary expressions. This is the case even in ILOG IBM CP Optimizer [6] which is, to our knowledge, currently one of the most powerful CP solver on the market and which is usually capable of allowing more flexibility in the definition of a CP formulation using intermediary expressions depending on the CP variables. Similarly to MAF_p, all other parameters which expression depends on the allocation would have to be defined as variables of the CP, leading to an unnecessarily complex CP to be solved.

Such an increase of complexity is deemed unnecessary since using matrix a is able to prevent it easily. In addition, the allocation problem in general being NP-hard by itself already, the resulting CP is likely to easily become intractable. As such, defining a boolean matrix instead of a vector of integers to model allocation variables optimizes the expressed CP problem so that its complexity is minimized as much as possible.

Definition Domains and Representation of Time. One may notice that, in the schedule generation problem, no representation of time as an integer variable has been implemented. Indeed, it turns out that textbook formulations with time-indexed formulations are not well suited for tackling the problems of complexity arising in real-world applications. In fact, allocation and scheduling problems both ae NP-hard problems, even when treated separately. Explicitly representing time as in classic CP formulations implies defining an integer variable per CPU time unit, so that the value of each time unit variable is constrained to be equal to the ID of the task scheduled during that time instant.

Time instants must cover the entire MAF, which represents a significant number of additional variables for each CP. The time representation therefore significantly increases the complexity of the CP. The schedule generation CP may even become intractable, as has been observed when time-indexed representation was implemented in the first versions of this thesis contributions.

As a conclusion, we did not keep the explicit time representation in the final version of the thesis outcomes, and we therefore chose not to represent time as discrete dates as usually done in classic CP scheduling problems. Instead, each start date – would that be for tasks executions or partitions windows – is defined as an integer variable comprised in the interval $[\![0; MAF]\!]$, since the schedule to be built corresponds to one MAF of the system. To ensure the selected start dates correspond to the corresponding core clock frequency, a constraint is expressed in the corresponding CP to ensure only even multiples fo the clock frequency are selected.

Although intervals $[\![0; MAF]\!]$ are valid domain definitions for each task execution and partition window start date, they can be further refined by rejecting invalid values. For instance for the task-level schedule generation, variable tO_i^k corresponds to the activation date of the k^{th} instance of τ_i. As such, by definition of task instances, the valid timing interval for tO_i^k is actually $[\![(k-1)T_i; kT_i]\!]$, which is a smaller interval than $[\![0; MAF]\!]$. Reducing timing intervals for all activation dates as in this example helps reducing the total solving time of the CP, since each decision variable is defined with a smaller definition domain.

In practice however, index k corresponds to the frame number of the global schedule, and not to the number of instances of τ_i. All tasks do not necessarily execute at each frame of the MAF. If τ_i does not run in the k^{th} frame of the schedule, then tO_i^k does not correspond to an actual instance of τ_i, and must therefore be set to zero in the corresponding scheduling CP, in order to avoid reserving some CPU budget for anything other than actual execution instances of tasks.

To be able to set to zero all activation dates tO_i^k which do not correspond to actual task executions to zero, the domain definition of activation dates must include the value zero. As such, one should define $[\![(k-1)T_i; kT_i[\![\cap \{0\}$ as the definition domain of tO_i^k. However, it may not always be possible to define an intersection of intervals or sets as a domain definition in practice, depending on the capabilities of the CP solvers currently available. An alternative can be to define $[\![0; kT_i[\![$ as the definition domain of tO_i^k instead. In that case, an additional constraint in the corresponding CP must be expressed in order to reject all values between 1 and $(k-1)T_i$ as invalid values for tO_i^k. The expression of such a constraint will be given later in chapter 6.

Path Allocation and Memory Space Allocation. In this thesis, we consider the software allocation phase to also include allocation to the memory paths and to the memory space. Both are linked and taken into account in this thesis, however we chose to represent only the path allocation as variables of the allocation problem, and indirectly imply the memory space allocation from the path allocation.

In fact, the allocation of memory paths to partitions (resp. tasks), i.e. the action of deciding which partition (resp. task) is allowed to access which memory controller at runtime, impacts the memory layout in non unified architectures. One must always verify that a given path allocation is realistic, i.e. that the memory context of the partitions (resp. tasks) can indeed all be allocated to the memory area addressed by the corresponding memory controller they have respectively been allocated to.

Choosing to define the software to path allocation as variables of the allocation problem instead of defining the software to memory areas allocation as variables, enables to significantly reduce the number of implemented variables. Each memory page would have to be considered as a component that could be allocated or not to any partition (resp. task), which would lead to much bigger matrices than $t2mc$ and $p2mc$. Instead, one equation can be defined in the allocation problem as a constraint verifying that all memory contexts fit altogether in their respective memory areas. It is precisely the way it is done in this thesis, as will be explained in chapter 6.

Proposed Timing Analyses

Soundness of the Analyses. It is important to note that the implemented verification analyses are safe since they belong to the category of static analyses: the response time analysis is a sufficient condition of feasibility that bounds tasks WCRTs and WCETs, naming: (i) the worst-case execution duration of all code instructions of each task entry point, (ii) the time it spends being preempted and waiting to resume its execution; (iii) the time it spends waiting for access to shared resources. No assumption is made on the memory requests arrival times, so that the produced interference upper-bounds cover the worst-case scenario of arrival times by considering every request emitted on a core is the last one to be serviced at runtime. However, although this may lead to the production of safe WCET upper-bounds, the price to pay for it usually is significant pessimism in the produced bounds.

Modularity and Flexibility of the Proposed Interference Models and Timing Analyses. The timing analyses presented in this chapter enable to verify the validity of configuration choices. One noticeable advantage of the equations representing the interference model is their modularity: it is possible to define an allocation and/or schedule search algorithm in which the timing analysis is embedded, so that at design time, for each step of the design space exploration, an instantaneous answer can be obtained on the feasibility of the configuration under investigation. As such, as will be explained in chapter 6, the timing analysis proposed for allocation verification is exploited at the first stages of the integration as a sufficient condition for feasibility to guide the software/hardware allocation choices. The timing analysis proposed for schedule verification is exploited once an allocation has been configured to guide the exploration phase during which a static scheduling table is built.

In addition, defining interference through functions $d_{DRAM}()$ and $d_{INT}()$ depending on the corresponding tasks WCRTs is convenient for reusing the same interference functions for both the allocation and the scheduling problems without requiring any significant model modification, despite the fact that tasks and partitions are represented throughout different parameters in the allocation and the scheduling problems. As such, the resulting timing analyses proposed in this thesis can easily be transfered to any existing allocation and/or schedule generation framework. Moreover, if one wants to exploit hardware models for the main memory or the interconnect other than the ones presented in chapter 4, it is easy to update the timing analysis by modifying impacted equations accordingly. Such a characteristic is very convenient for controlling models obsolescence and regularly review state of the art multicore interference models.

Portability and Reuse. The two main hindrances to the extension of current static analysis techniques to multicore architectures are:

- The fact that multicore COTS embed complex components, which sometimes have non-deterministic arbitration policies;
- The fact that multicore COTS embed components which design features is undisclosed for IP protection purposes.

To our knowledge, as of today, every attempt at extending code-level WCET analysis from single- to multicore environments lead to a state space explosion because of the complexity of current COTS architectures.

The timing analyses proposed in this thesis are static analysis techniques, and yet they do not fall into the issue of state space explosion and intractability. Indeed, all necessary parameters for interference bounding correspond to data that will always be made available to COTS customers. In particular, the parameters for memory interference computation depend only on standard DRAM timing parameters that can be found in the JEDEC standard, to which all DRAM memories show compliance partly by disclosing their implementations of the core parameters of JEDEC-compliant memories. As such, our analysis techniques are applicable to COTS since the necessary parameters for our analyses will always be disclosed.

In addition, our analysis techniques will not lead to state space explosion as can be experienced with static code analysis. The response time analysis abstracts a task by its scheduling parameters plus some extra parameters linked to its entry point (see task models presented in chapter 4). In contrast with code analysis, it does not rely on the exploration of execution traces, and as such, suffers no risk of state space explosion if performed in multicore environments.

Pessimism. As mentioned before in this thesis, the implemented analysis techniques use coarse grain models, which saves time by easing the verification phase, but approximates timing bounds by upper values, which implies pessimistic bounds. The embedded pessimism is especially high for the interference models which have been overly approximated to cope with undisclosed information in COTS, such as for the interconnect for example.

Another pessimism-inducing situation is the non-distinction between reads and writes requests, which have both different hardware actions, and thus lead to different serving latencies. However, as was also mentioned before, it is already a common habit of aerospace industries to overestimate tasks WCETs by adding additional safety margins to the computed bounds, even in single-core environments. As such, the pessimism displayed by our approach is not a hindrance to the reuse of our contributions by aerospace industries, which is the final goal of this thesis: aerospace industries are likely to welcome the embedded pessimism as safety margins. Pessimism may reduce significantly the performance of a system statically configured, but all aerospace industries care about in the short term concerning multicores is determinism rather than performance.

Timing Anomalies and Applicability of our Work to COTS Architectures. As stated in the assumptions and presented in this chapter, tasks WCRTs and WCETs are computed as if the hardware architecture were anomaly-free. It is mostly not the case in current COTS multicores. Although anomalies are not faulty events, we assume them to be handled as if they actually were faulty events, e.g. by defining a specific recovery action to be undertaken in the context of health monitoring. The produced WCETs being safe upper-bounds except for the advent of an anomaly occurring at runtime, the situation where a task tries to overrun its WCET at runtime could be considered as the occurrence of an anomaly.

To cope with anomalies and allow our WCET model to be valid despite the anomaly-free assumption, we propose to enforce the computed WCET for each task at runtime by forcing any task to hold the CPU for the exact amount of time represented by the computed WCET upper-bound. At the end of such a budget for a given task, we assume the task context to be switched to the next task to be executed according to the static schedule defined or the scheduling policy enforced. In addition, we assume some monitoring mechanism to be able to detect whether a task execution was indeed finished or not, the latter case corresponding to the occurrence of an anomaly.

In such cases, a fault recovery action specifically defined to cope with anomalies is assumed to be undertaken by the OS, such as restarting the task or its partition window on another processing resource reserved for fault recovery. This mechanism corresponds to the safety net that has been implemented by Jan Nowotsch in the context of his PhD thesis [122]. Such safety net and health monitoring capabilities may therefore enable our work to remain applicable to COTS architectures that are not timing anomaly-free.

Proposed Integration Strategies

Why two Strategies. The choice of providing two different strategies instead of one has been motivated by the different properties of the two propositions. While the one-to-all integration strategy respects all mandatory requirements of aerospace systems, the second integration strategy offers as much SWaP reduction as can be safely achievable (i.e. using DO-178 compliant analysis techniques) for DAL A functions while using the timing analysis techniques proposed in this thesis. However, the one-to-one integration strategy does not comply with robust partitioning requirements of DAL A functions, which means additional verification steps would be needed, such as finding workarounds to robust partitioning for fault containment, or convincing certification authorities and system designers of the validity of said workarounds for instance. Such additional steps are time and effort consuming, meaning that such a solution would not be exploitable in the near future as the one-to-all integration strategy could be.

The one-to-one integration strategy is however still proposed as a contribution in this thesis because it may still be used for less critical functions for which it is not mandatory to forbid concurrent accesses to shared resources at runtime, for instance provided that sufficient monitoring recovery actions and specific protection mechanisms are used. For such functions, using the one-to-one integration strategy leads to optimized designs in which SWaP reduction has been achieved as much as possible. For all other applications requiring to follow an incremental certification acceptance process and requiring robust partitioning capabilities, the one-to-all integration strategy must be used.

Legacy IMA Software versus Future IMA Software. As explained in chapter 2, the integration process consists in discussions between application suppliers, system designers and a module integrator. The proposed activities take place once each to-be-designed avionics function has been assigned to application suppliers, and once each application supplier has broken down the to-be-designed function(s) it is responsible for into partitions and tasks. However, one may notice that the integration strategy proposed in this thesis actually remain applicable if used either with legacy IMA software or software under development as well. Such a property is convenient for aerospace industries, for which it means our one-to-all integration strategy can be exploited for future systems as long as more optimized integration and/or coding techniques able to fully benefit from parallelization capabilities are discovered. As such, we can say that the thesis succeeded in bringing a solution to the problem of implementing multicore-based IMA systems in the nearest future possible.

8.3 Future Work

The integration strategies and the associated timing analyses provide a basis for the allocation and schedule generation verified in advance for IMA applications to be integrated into multicore processors.

Some future work can be identified, for instance the experimentation of different optimization criterion for the schedule generation problems relative to each integration strategy. In fact, aside from the CPU workload, a common preoccupation of avionics system designers is the partition context switches implied by a schedule, as they represent additional runtime overheads unaccounted for in partitions time budgets. Partition switching is not free, and the time spent switching contexts might result in partitions overrunning their allocated time budgets during one MAF. As such, it might be interesting for industrial systems to optimize the schedule generated by our CP in order to reduce the number of partition switches induced.

Unfortunately, the worst-case overhead of a partition switch is usually unknown [106, 75]. There exists no model in the literature, since what is done during switch and how it is done is processor- and OS-dependent and usually never disclosed in detail due to IP protection. As such, it is not possible to safely bound context switch overheads. As a consequence, it is not possible to define them as a constant added to partitions time windows either, since no upper-bound of context switching overheads can be proved to actually be an upper-bound, and then used in the computation of tasks WCETs. This would correspond to an interesting continuation of the work presented in this thesis.

Chapter 9

Appendices

9.1 DRAM Parameters for Interference Computation

This appendix provides additional infos for further comprehension of the DRAM interference computational model presented in chapter 5 (cf. page 123). Unless explicitly stated, all equations shown in this appendix directly correspond to the approach proposed by Kim et al. [84].

Table 9.1 gives the JEDEC params used to compute DRAM latencies. Always be found in datasheet since standard parameters characterizing all JEDEC-compliant DRRs.

Computation of l_{max}

To compute l_{max}, one has to first compute the delay of each command achieved when accessing the main memory. In the worst-case situation a row-conflict is detected when trying to issue a given request, which leads to three successive commands to be issued, namely precharge (PRE), activation (ACT) and reading/writing (WR) depending on the type of the request. As such, l_{max} can be computed as follows:

$$l_{max} = L_{inter}^{PRE} + L_{inter}^{ACT} + L_{inter}^{RW} \tag{9.1}$$

where L_{inter}^{PRE} is the latency corresponding to issuing a precharge command and is computed using equation (9.2); L_{inter}^{ACT} is the latency corresponding to issuing an activation command and is computed using equation (9.3); and L_{inter}^{RW} is the latency corresponding to issuing a read or write command and is computed using equation (9.4). A precharge command takes one DRAM clock cycle to be issued:

$$L_{inter}^{PRE} = t_{CK} \tag{9.2}$$

The JEDEC standard specifies that two successive activation commands for different banks must be separated from each other of at least t_{RRD} time units, and no more tan four activation commands can be triggered during a time interval of t_{FAW}. As such, a safe bound on the latency suffered when issuing an activation command can be computed as follows:

$$L_{inter}^{ACT} = \max\left(t_{RRD}, \quad t_{FAW} - 3 \times t_{RRD}\right) \times t_{CK} \tag{9.3}$$

The maximum latency when issuing a read (RD) or write (WR) command can be derived from assessing the worst-case sequence of read/write access requests. If a WR/RD sequence comes after a RD/WR sequence, the data flow direction of the data bus causes a data bus turn-around delay, as extendedly described in [84] Two types of turn-around delay exist. In the case of a WR-to-RD, the RD request needs to wait for $WL + BL = 2 + tWTR$ cycles. In the case of a RD-to-WR, the WR request needs to wait for $CL + BL = 2 + 2WL$ cycles. The maximum latency when issuing a WR/RD command corresponds to the maximum delay of the two turn-around delays:

$$L_{inter}^{RW} = \max\left(WL + \frac{BL}{2} + t_{WTR}, \quad CL + \frac{BL}{2} + 2 - WL\right) \times t_{CK} \tag{9.4}$$

Reorderings

The number of maximum row hits that can be prioritized over older row-conflicts when reodering memory requests is denoted $N_{reorder}$, and computed in [84] using the following equation:

$$N_{reorder} = \min\left(\frac{N_{cols}}{BL}, \quad N_{cap}\right) \tag{9.5}$$

where N_{cap} is the hardware threshold that can be set to bound the number of re-ordering between requests.

The delay suffered by a task at runtime due to reordering effects depends on the core p considered, the memory sharings, etc. Let $reorder()$ be the function computing the delay suffered due to memory requests reorderings. Then $reorder()$ depends on the row-hit service time, the row-conflict service time, the consecutive row-hit service time and Nreorder.

Parameter	Symbol	Value	Units
DRAM clock cycle time	t_{CK}	0.0015	μs
Precharge latency	t_{RP}	8	cycles
Activate Latency	t_{RCD}	8	cycles
CAS Read Latency	CL	9	cycles
CAS Write Latency	WL	7	cycles
Burst Length	BL	8	columns
Write to Read Delay	t_{WTR}	7	cycles
Write Recovery Time	t_{WR}	10	cycles
Activate to Activate Delay	t_{RRD}	11	cycles
Four Activate Windows	t_{FAW}	20	cycles
Number of columns in a row	N_{cols}	1024	–
Number of maximum rowhits that can be prioritized over older row-conflicts when reodering memory requests	$N_{reorder}$	12	–

Table 9.1: Standard DRAM Parameters

Row-hit service time: it is the latency for a request for which the column is already in the row-buffer. It is denoted L_{hit} and computed as follows:

$$L_{hit} = \max\left(CL + \frac{BL}{2} + 2, \quad WL + \frac{BL}{2} + \max\left(t_{WTR}, \ t_{WR}\right)\right) \times t_{CK} \tag{9.6}$$

Row-conflict service time: it is the latency for a request for which the column is not already in the column-buffer. It is denoted L_{conf} and computed as follows:

$$L_{conf} = (t_{RP} + t_{RCD}) \times t_{CK} + L_{hit} \tag{9.7}$$

Consecutive row-hit requests: if m is the maximum number of memory requests present in the request buffer. For a given m, requests corresponding to the same row may be reordered by the memory in order to be issued consecutively. The maximum latency corresponding to servicing all consecutive row-hit requests can be computed as follows:

$$L_{conhit}(m) = \left\{\left\lceil\frac{m}{2}\right\rceil \times \left(WL + \frac{BL}{2} + t_{WTR}\right) + \left\lfloor\frac{m}{2}\right\rfloor \times CL + (t_{WR} + t_{WTR})\right\} \times t_{CK} \tag{9.8}$$

In the worst-case scenario, the number of memory requests present in the request buffer m corresponds to $N_{reorder}$.

Finally, the function deriving an upper-bound on the latencies due to reordering effects is computed as follows:

$$\forall p \in [1; N_C],$$
$$reorder(p) = \begin{cases} 0 \text{ if no core } q \text{ shares any memory area with core } p, \\ L_{conhit}(N_{reorder}) + \sum\limits_{\substack{q=1 \\ q!=p \\ shared(p,q)=\varnothing}}^{N_C} \qquad \text{otherwise.} \end{cases} \tag{9.9}$$

In our contributions, the $shared()$ sets have been defined slightly differently depending on the considered integration strategy. Equation (9.9) directly corresponds to the one-to-one integration strategy. For the one-to-all integration strategy, $reorder()$ has an additional variable: the index of the partition π_i considered. As such, in this thesis, the definition of $reorder()$ proposed by Kim et al. has been modified by the author of this thesis as follows in order to match the one-to-all integration strategy:

$$\forall p \in [1; N_C], \forall \pi_i,$$

$$reorder(p, j) = \begin{cases} 0 \quad \text{if no task of } \pi_i \text{ on another core than } p \text{ shares any} \\ \quad \text{memory area with core } p, \\ L_{conhit}(N_{reorder}) + \sum\limits_{\substack{q=1 \\ q!=p \\ shared(i,p,q)=\varnothing}}^{N_C} \qquad \text{otherwise.} \end{cases} \tag{9.10}$$

Bibliography

[1] ”ARINC 653 avionics application software standard interface”, 1996.

[2] aiT WCET Analyzers, AbsInt Angewandte Informatik GmbH.

[3] CAST-32, ”Multi-core Processors”, 2014.

[4] EASA Official Website: https://www.easa.europa.eu.

[5] Federal Aviation Authorities’ Official Website: http://www.faa.gov.

[6] IBM ILOG, Cplex CP Optimizer. Website: http://www-01.ibm.com/software/commerce/optimization/cplex-cp-optimizer/.

[7] Infineon Aurix Tricore Presentation. Website: http://www.infineon.com/cms/en/product/microcontroller bit-tricore-tm-microcontroller/channel.html?channel=ff80808112ab681d0112ab6b64b50805.

[8] Joint Aviation Authorities’ Official Website: https://jaato.com.

[9] QorIQ^{TM} p4080 Communications Processor, NXP (ex Freescale), Product Brief.

[10] RTCA Radio Technical Commission for Aeronautics, ”Software Considerations in Airborne Systems and Equipment Certification” (DO178), 1992.

[11] SAE International, ”Aerospace Recommended Practice ARP4754 – Guidelines For Development Of Civil Aircraft and Systems”, issued 1996; published 2010.

[12] SAE International’s Official Website: http://www.sae.org.

[13] TACLEbench, TACLe Benchmark Suite – TACLe (Timing Analysis at Code Level) ICT COST Action IC1202. Website: http://www.tacle.eu/index.php/activities/taclebench.

[14] Wind river VxWorks 653 3.0 Multi-Core Edition, Wind River Inc, Product Overview.

[15] Benny Akesson, Kees Goossens, and Markus Ringhofer. Predator: a predictable sdram memory controller. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 251–256. ACM, 2007.

[16] Ahmad Al Sheikh. *Resource allocation in hard real-time avionic systems: scheduling and routing problems.* PhD thesis, Toulouse, INSA, 2011.

[17] Ahmad Al Sheikh, Olivier Brun, and Pierre-Emmanuel Hladik. Partition Scheduling on an IMA Platform with Strict Periodicity and Communication Delays. In *18th International Conference on Real-Time and Network Systems*, pages 179–188, Toulouse, France, November 2010.

[18] Luis Almeida and Paulo Pedreiras. Scheduling within temporal partitions: response-time analysis and server design. In *Proceedings of the 4th ACM international conference on Embedded software*, pages 95–103. ACM, 2004.

[19] Sebastian Altmeyer, Robert I. Davis, Leandro Soares Indrusiak, Claire Maiza, Vincent Nélis, and Jan Reineke. A generic and compositional framework for multicore response time analysis. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems, RTNS 2015, Lille, France, November 4-6, 2015*, pages 129–138, 2015.

[20] Sebastian Altmeyer, Roeland Douma, Will Lunniss, and Robert I Davis. Outstanding paper: Evaluation of cache partitioning for hard real-time systems. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 15–26. IEEE, 2014.

[21] James H Anderson, Sanjoy K Baruah, and Björn B Brandenburg. Multicore operating-system support for mixed criticality. Citeseer.

[22] James H Anderson, John M Calandrino, and UmaMaheswari C Devi. Real-time scheduling on multicore platforms. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 179–190. IEEE, 2006.

[23] Nesrine Badache. *Allocation temporelle de systèmes avioniques modulaires embarqués*. PhD thesis, 2016.

[24] Andrea Baldovin, Alessandro Zovi, Geoffrey Nelissen, and Stefano Puri. The concerto methodology for model-based development of avionics software. In *Ada-Europe International Conference on Reliable Software Technologies*, pages 131–145. Springer, 2015.

[25] Sanjoy Baruah and Björn Brandenburg. Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 160–169. IEEE, 2013.

[26] Sanjoy Baruah and Gerhard Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 3–12. IEEE, 2011.

[27] Sanjoy K Baruah, Alan Burns, and Robert I Davis. Response-time analysis for mixed criticality systems. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 34–43. IEEE, 2011.

[28] Moris Behnam. Hierarchical real time scheduling and synchronization. 2008.

[29] Jingyi Bin, Sylvain Girbal, Daniel Gracia Pérez, Arnaud Grasset, and Alain Merigot. Studying co-running avionic real-time applications on multi-core cots architectures. *Embedded real time software and systems (ERTS'14)*, 2014.

[30] Enrico Bini and Giorgio C Buttazzo. Biasing effects in schedulability measures. In *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*, pages 196–203. IEEE, 2004.

[31] Konstantinos Bletsas and Stefan M Petters. Using nps-f for mixed-criticality multicore systems. In *33rd IEEE Real-Time Systems Symposium*, pages 36–36. ACM, 2012.

[32] Frédéric Boniol, Hugues Cassé, Eric Noulard, and Claire Pagetti. Deterministic execution model on cots hardware. In *International Conference on Architecture of Computing Systems*, pages 98–110. Springer, 2012.

[33] Frédéric Boniol, Pierre-Emmanuel Hladik, Claire Pagetti, Frédéric Aspro, and Victor Jégu. A framework for distributing real-time functions. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 155–169. Springer, 2008.

[34] Richard Bradford, Shana Fliginger, Rockwell Collins, Cedar Rapids, Sibin Mohan, Rodolfo Pellizzoni, Cheolgi Kim, Marco Caccamo, Lui Sha, et al. Exploring the design space of ima system architectures. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5–E. IEEE, 2010.

[35] Bach D Bui, Marco Caccamo, Lui Sha, and Joseph Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 101–110. IEEE, 2008.

[36] Paolo Burgio, Martino Ruggiero, Francesco Esposito, Mauro Marinoni, Giorgio Buttazzo, and Luca Benini. Adaptive tdma bus allocation and elastic scheduling: a unified approach for enhancing robustness in multi-core rt systems. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pages 187–194. IEEE, 2010.

[37] Alan Burns and Rob Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, 2013.

[38] Thomas Carle, Dumitru Potop-Butucaru, Yves Sorel, and David Lesens. From dataflow specification to multiprocessor partitioned time-triggered real-time implementation. *Leibniz Transactions on Embedded Systems*, 2(2):01–1, 2015.

[39] Karam S Chatha and Ranga Vemuri. Magellan: multiway hardware-software partitioning and scheduling for latency minimization of hierarchical control-dataflow task graphs. In *Proceedings of the ninth international symposium on Hardware/software codesign*, pages 42–47. ACM, 2001.

[40] Sudipta Chattopadhyay, Lee Kee Chong, Abhik Roychoudhury, Timon Kelter, Peter Marwedel, and Heiko Falk. A unified wcet analysis framework for multicore platforms. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4s):124, 2014.

[41] Sudipta Chattopadhyay, Abhik Roychoudhury, and Tulika Mitra. Modeling shared cache and bus in multi-cores for timing analysis. In *Proceedings of the 13th international workshop on software & compilers for embedded systems*, page 6. ACM, 2010.

[42] Micaiah Chisholm, Bryan C Ward, Namhoon Kim, and James H Anderson. Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. In *Real-Time Systems Symposium, 2015 IEEE*, pages 305–316. IEEE, 2015.

[43] Bekim Cilku, Alfons Crespo, Peter Puschner, Javier Coronel, and Salvador Peiro. A tdma-based arbitration scheme for mixed-criticality multicore platforms. In *Event-based Control, Communication, and Signal Processing (EBCCSP), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[44] Mikel Cordovilla, Frédéric Boniol, Julien Forget, Eric Noulard, and Claire Pagetti. Developing critical embedded systems on multicore architectures: the prelude-schedmcore toolset. In *19th International Conference on Real-Time and Network Systems*, 2011.

[45] Luis Alejandro Cortés, Petru Eles, and Zebo Peng. Quasi-static scheduling for real-time systems with hard and soft tasks. In *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, page 21176. IEEE Computer Society, 2004.

[46] Alfons Crespo, Ismael Ripoll, and Miguel Masmano. Partitioned embedded architecture based on hypervisor: The xtratum approach. In *Dependable Computing Conference (EDCC), 2010 European*, pages 67–72. IEEE, 2010.

[47] Dakshina Dasari, Bjorn Andersson, Vincent Nelis, Stefan M Petters, Arvind Easwaran, and Jinkyu Lee. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1068–1075. IEEE, 2011.

[48] Dakshina Dasari, Bjorn Andersson, Vincent Nelis, Stefan M Petters, Arvind Easwaran, and Jinkyu Lee. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1068–1075. IEEE, 2011.

[49] Dakshina Dasari, Bjorn Andersson, Vincent Nelis, Stefan M Petters, Arvind Easwaran, and Jinkyu Lee. Response time analysis of cots-based multicores considering the contention on the shared memory bus. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 1068–1075. IEEE, 2011.

[50] Dakshina Dasari, Vincent Nelis, and Björn Andersson. Wcet analysis considering contention on memory bus in cots-based multicores. In *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–4. IEEE, 2011.

[51] Robert Davis, Tullio Vardanega, Jan Alexanderson, Vatrinet Francis, Pearce Mark, Broster Ian, Azkarate-Askasua Mikel, Franck Wartel, Liliana Cucu-Grosjean, Patte Mathieu, et al. Proxima: A probabilistic approach to the timing behaviour of mixed-criticality systems. *Ada User Journal*, 2:118–122, 2014.

[52] Robert I Davis and Alan Burns. Hierarchical fixed priority pre-emptive scheduling. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 10–pp. IEEE, 2005.

[53] Dionisio De Niz, Karthik Lakshmanan, and Ragunathan Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 291–300. IEEE, 2009.

[54] Dionisio de Niz and Linh TX Phan. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 111–122. IEEE, 2014.

[55] Zhong Deng and JW-S Liu. Scheduling real-time applications in an open environment. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 308–319. IEEE, 1997.

[56] Olivier Desenfans, Antonio Paolillo, Vladimir Svoboda, Ben Rodriguez, Joël Goossens, and Dragomir Milojevic. Design and implementation of a multi-core embedded real-time operating system kernel.

[57] Roeland J Douma, Sebastian Altmeyer, and Andy D Pimentel. Fast and precise cache performance estimation for out-of-order execution. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 1132–1137. EDA Consortium, 2015.

[58] Guy Durrieu, Madeleine Faugere, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Embedded Real Time Software (ERTS'14)*, 2014.

[59] Stephen A Edwards and Edward A Lee. The case for the precision timed (pret) machine. In *Proceedings of the 44th annual Design Automation Conference*, pages 264–265. ACM, 2007.

[60] Friedrich Eisenbrand, Nicolai Hähnle, Martin Niemeier, Martin Skutella, José Verschae, and Andreas Wiese. Scheduling periodic tasks in a hard real-time environment. In *International Colloquium on Automata, Languages, and Programming*, pages 299–311. Springer, 2010.

[61] Cecilia Ekelin and Jan Jonsson. Solving embedded system scheduling problems using constraint programming. 2000.

[62] Xiang Feng and Aloysius K Mok. A model of hierarchical real-time virtual resources. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 26–35. IEEE, 2002.

[63] Christian Ferdinand, Reinhold Heckmann, Marc Langenbach, Florian Martin, Michael Schmidt, Henrik Theiling, Stephan Thesing, and Reinhard Wilhelm. Reliable and precise wcet determination for a real-life processor. In *International Workshop on Embedded Software*, pages 469–485. Springer, 2001.

[64] Gabriel Fernandez, Jaume Abella, Eduardo Quiñones, Christine Rochange, Tullio Vardanega, and Francisco J. Cazorla. Contention in multicore hardware shared resources: Understanding of the state of the art. In *14th International Workshop on Worst-Case Execution Time Analysis, WCET 2014, July 8, 2014, Ulm, Germany*, pages 31–42, 2014.

[65] Nathan Fisher, James H Anderson, and Sanjoy Baruah. Task partitioning upon memory-constrained multiprocessors. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 416–421. IEEE, 2005.

[66] Rudolf Fuchsen. How to address certification for multi-core based ima platforms: Current status and potential solutions. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5–E. IEEE, 2010.

[67] Georgia Giannopoulou, Nikolay Stoimenov, Pengcheng Huang, and Lothar Thiele. Mapping mixed-criticality applications on multi-core architectures. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2014, Dresden, Germany, March 24-28, 2014*, pages 1–6, 2014.

[68] Sylvain Girbal, Daniel Gracia Pérez, Jimmy Le Rhun, Madeleine Faugère, Claire Pagetti, and Guy Durrieu. A complete tool-chain for an interference-free deployment of avionic applications on multi-core systems. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pages 1–13. IEEE, 2015.

[69] M González Harbour, JJ Gutiérrez García, JC Palencia Gutiérrez, and JM Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Real-Time Systems, 13th Euromicro Conference on, 2001.*, pages 125–134. IEEE, 2001.

[70] Dip Goswami, Martin Lukasiewycz, Reinhard Schneider, and Samarjit Chakraborty. Time-triggered implementations of mixed-criticality automotive software. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1227–1232. EDA Consortium, 2012.

[71] Stefan Groesbrink, Luis Almeida, Mario de Sousa, and Stefan M Petters. Towards certifiable adaptive reservations for hypervisor-based virtualization. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24. IEEE, 2014.

[72] Stefan Groesbrink, Simon Oberthür, and Daniel Baldin. Architecture for adaptive resource assignment to virtualized mixed-criticality real-time systems. *ACM SIGBED Review*, 10(1):18–23, 2013.

[73] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Cache-aware scheduling and analysis for multicores. In *Proceedings of the seventh ACM international conference on Embedded software*, pages 245–254. ACM, 2009.

[74] Arne Hamann, Rafik Henia, Razvan Racu, Marek Jersak, Kai Richter, and Rolf Ernst. Symta/s - symbolic timing analysis for systems, 2004.

[75] Sanghyun Han and Hyun-Wook Jin. Resource partitioning for integrated modular avionics: comparative study of implementation alternatives. *Software: Practice and Experience*, 44(12):1441–1466, 2014.

[76] Damien Hardy and Isabelle Puaut. Wcet analysis of multi-level non-inclusive set-associative instruction caches. In *Real-Time Systems Symposium, 2008*, pages 456–466. IEEE, 2008.

[77] Mohamed Hassan and Hiren Patel. Criticality-and requirement-aware bus arbitration for multi-core mixed criticality systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–11. IEEE, 2016.

[78] Pierre-Emmanuel Hladik, Hadrien Cambazard, Anne-Marie Déplanche, and Narendra Jussien. Solving allocation problems of hard real-time systems with dynamic constraint programming. In *14th International Conference on Real-Time and Network Systems (RTNS'06)*, pages 214–223, 2006.

[79] Pengcheng Huang, Georgia Giannopoulou, Rehan Ahmed, Davide B Bartolini, and Lothar Thiele. An isolation scheduling model for multicores. In *Real-Time Systems Symposium, 2015 IEEE*, pages 141–152. IEEE, 2015.

[80] Xavier Jean. *Hypervisor control of COTS multicores processors in order to enforce determinism for future avionics equipment.* PhD thesis, PhD Thesis, Telecom ParisTech, 2015.

[81] Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

[82] Owen R Kelly, Hakan Aydin, and Baoxian Zhao. On partitioned scheduling of fixed-priority mixed-criticality task sets. In *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1051–1059. IEEE, 2011.

[83] Omar Kermia and Yves Sorel. Schedulability analysis for non-preemptive tasks under strict periodicity constraints. In *The Fourteenth IEEE Internationl Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2008, Kaohisung, Taiwan, 25-27 August 2008, Proceedings*, pages 25–32, 2008.

[84] Hyoseung Kim, Dionisio de Niz, Björn Andersson, Mark H. Klein, Onur Mutlu, and Ragunathan Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2014, Berlin, Germany, April 15-17, 2014*, pages 145–154, 2014.

[85] Hyoseung Kim, Arvind Kandhalu, and Ragunathan Rajkumar. A coordinated approach for practical os-level cache management in multi-core real-time systems. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 80–89. IEEE, 2013.

[86] Hyoseung Kim, Arvind Kandhalu, and Ragunathan Rajkumar. A coordinated approach for practical os-level cache management in multi-core real-time systems. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 80–89. IEEE, 2013.

[87] Jung-Eun Kim, Man-Ki Yoon, Sungjin Im, Richard Bradford, and Lui Sha. Multi-ima partition scheduling with synchronized solo-partitions for multi-core avionics systems. 2012.

[88] Yooseong Kim, David Broman, Jian Cai, and Aviral Shrivastaval. Wcet-aware dynamic code management on scratchpads for software-managed multicores. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 179–188. IEEE, 2014.

[89] M.T. Kirsch. *Technical Support to the National Highway Traffic Safety Administration (NHTSA) on the Reported Toyota Motor Corporation (TMC) Unintended Acceleration (UA) Investigation.* DIANE Publishing.

[90] Ondrej Kotaba, Jan Nowotsch, Michael Paulitsch, Stefan M Petters, and Henrik Theiling. Multicore in real-time systems–temporal isolation challenges due to shared resources. In *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2014.

[91] Angeliki Kritikakou, Christine Rochange, Madeleine Faugère, Claire Pagetti, Matthieu Roy, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time WCET controller for concurrent critical tasks in mixed-critical systems. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, page 139, 2014.

[92] NG Chetan Kumar, Sudhanshu Vyas, Ron K Cytron, Christopher D Gill, Joseph Zambreno, and Phillip H Jones. Cache design for mixed criticality real-time systems. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 513–516. IEEE, 2014.

[93] Tei-Wei Kuo and Ching-Hui Li. A fixed-priority-driven open environment for real-time applications. In *Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE*, pages 256–267. IEEE, 1999.

[94] Chang-Gun Lee, Hoosun Hahn, Yang-Min Seo, Sang Lyul Min, Rhan Ha, Seongsoo Hong, Chang Yun Park, Minsuk Lee, and Chong Sang Kim. Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE transactions on computers*, 47(6):700–713, 1998.

[95] Insup Lee, Joseph YT Leung, and Sang H Son. *Handbook of real-time and embedded systems.* CRC Press, 2007.

[96] Yann-Hang Lee, Daeyoung Kim, Mohamed Younis, Jeff Zhou, and James McElroy. Resource scheduling in dependable integrated modular avionics. In *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*, pages 14–23. IEEE, 2000.

[97] Benjamin Lesage, Damien Hardy, and Isabelle Puaut. Wcet analysis of multi-level set-associative data caches. In *OASIcs-OpenAccess Series in Informatics*, volume 10. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[98] Benjamin Lesage, Isabelle Puaut, and André Seznec. Preti: Partitioned real-time shared cache for mixed-criticality real-time systems. In *Proceedings of the 20th International Conference on Real-Time and Network Systems*, pages 171–180. ACM, 2012.

[99] Haohan Li and Sanjoy Baruah. Global mixed-criticality scheduling on multiprocessors. In *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems, ECRTS*, volume 12, 2012.

[100] Yan Li, Vivy Suhendra, Yun Liang, Tulika Mitra, and Abhik Roychoudhury. Timing analysis of concurrent programs running on shared cache multi-cores. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 57–67. IEEE, 2009.

[101] Giuseppe Lipari and Sanjoy K Baruah. Efficient scheduling of real-time multi-task applications in dynamic systems. In *IEEE Real Time Technology and Applications Symposium*, pages 166–175, 2000.

[102] Giuseppe Lipari and Enrico Bini. Resource partitioning among real-time applications. In *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, pages 151–158. IEEE, 2003.

[103] Giuseppe Lipari and Enrico Bini. A methodology for designing hierarchical scheduling systems. *Journal of Embedded Computing*, 1(2):257–269, 2005.

[104] Giuseppe Lipari, John Carpenter, and Sanjoy K Baruah. A framework for achieving inter-application isolation in multiprogrammed, hard real-time environments. In *RTSS*, pages 217–226, 2000.

[105] Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973.

[106] Fang Liu, Fei Guo, Yan Solihin, Seongbeom Kim, and Abdulaziz Eker. Characterizing and modeling the behavior of context switch misses. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 91–101. ACM, 2008.

[107] Lei Liu, Zehan Cui, Mingjie Xing, Yungang Bao, Mingyu Chen, and Chengyong Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. In *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 367–376. ACM, 2012.

[108] C Douglass Locke, David R Vogel, Lee Lucas, and John B Goodenough. Generic avionics software specification. Technical report, DTIC Document, 1990.

[109] Andreas Löfwenmark and Simin Nadjm-Tehrani. Challenges in future avionic systems on multi-core platforms. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on*, pages 115–119. IEEE, 2014.

[110] Will Lunniss, Sebastian Altmeyer, Giuseppe Lipari, and Robert I. Davis. Accounting for cache related pre-emption delays in hierarchical scheduling. In *22nd International Conference on Real-Time Networks and Systems, RTNS '14, Versaille, France, October 8-10, 2014*, page 183, 2014.

[111] Renato Mancuso, Rodolfo Pellizzoni, Marco Caccamo, Lui Sha, and Heechul Yun. Wcet(m) estimation in multi-core systems using single core equivalence. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*, pages 174–183, 2015.

[112] Thomas Mégel. *Placement, ordonnancement et mécanismes de migration de tâches temps-réel pour des architectures distribuées multicoeurs.* PhD thesis, 2012.

[113] Aloysius K Mok and Xiang Alex. Towards compositionality in real-time resource partitioning based on regularity bounds. In *Real-Time Systems Symposium, 2001.(RTSS 2001). Proceedings. 22nd IEEE*, pages 129–138. IEEE, 2001.

[114] Aloysius K Mok, Xiang Feng, and Deji Chen. Resource partition for real-time systems. In *Real-Time Technology and Applications Symposium, 2001. Proceedings. Seventh IEEE*, pages 75–84. IEEE, 2001.

[115] Malcolm S Mollison, Jeremy P Erickson, James H Anderson, Sanjoy K Baruah, and John A Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1864–1871. IEEE, 2010.

[116] Soukayna M'Sirdi, Wenceslas Godard, and Marc Pantel. A Multi-Core Interference-Aware Schedulability Test for IMA Systems, as a Guide for SW/HW Integration. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France, January 2016.

[117] Soukayna M'Sirdi, Wenceslas Godard, Marc Pantel, and Stephan Stilkerich. A New IMA System Integration Process for Multicore Transfer, with Compliance to Incremental Certification and Robust Partitioning. In *6th EASN International Conference on Innovation in European Aeronautics Research (EASN 2016)*, Porto, Portugal, October 2016.

[118] Soukayna M'Sirdi, Wenceslas Godard, Marc Pantel, and Stephan Stilkerich. Improved resource efficient allocation of ima applications to multi-cores. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pages 1–10. IEEE, 2016.

[119] Hamid Mushtaq, Zaid Al-Ars, and Koen Bertels. Accurate and efficient identification of worst-case execution time for multicore processors: A survey. In *2013 8th IEEE Design and Test Symposium*, pages 1–6. IEEE, 2013.

[120] Kartik Nagar and YN Srikant. Precise shared cache analysis using optimal interference placement. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 125–134. IEEE, 2014.

[121] Hemendra Singh Negi, Tulika Mitra, and Abhik Roychoudhury. Accurate estimation of cache-related preemption delay. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 201–206. ACM, 2003.

[122] Jan Nowotsch. *Interference-sensitive Worst-case Execution Time Analysis for Multi-core Processors*. PhD thesis, University of Augsburg, 2014.

[123] Jan Nowotsch, Michael Paulitsch, Daniel Buhler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 109–118, 2014.

[124] Roman Obermaisser and Donatus Weber. Architectures for mixed-criticality systems based on networked multi-core chips. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, pages 1–10. IEEE, 2014.

[125] Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, Pascal Richard, and Frédéric Madiot. Mosart framework: A collaborative tool for modeling and analyzing embedded real-time systems. In *Complex Systems Design & Management*, pages 283–295. Springer, 2015.

[126] Yassine Ouhammou, Emmanuel Grolleau, and Pascal Richard. Extension and utilization of a design framework to model integrated modular avionic architecture. In *Model and Data Engineering*, pages 16–27. Springer, 2015.

[127] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Robert I. Davis, and Mateo Valero. IAˆ3: An interference aware allocation algorithm for multicore hard real-time systems. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2011, Chicago, Illinois, USA, 11-14 April 2011*, pages 280–290, 2011.

[128] Marco Paolieri, Eduardo Quinones, Francisco J Cazorla, and Mateo Valero. An analyzable memory controller for hard real-time cmps. *IEEE Embedded Systems Letters*, 1(4):86–90, 2009.

[129] Jeff Parkhurst, John Darringer, and Bill Grundmann. From single core to multi-core: preparing for a new exponential. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 67–72. ACM, 2006.

[130] Rodolfo Pellizzoni. Managing memory for timing predictability. TORRENTS, 2014.

[131] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. A predictable execution model for cots-based embedded systems. In *2011 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 269–279. IEEE, 2011.

[132] Rodolfo Pellizzoni and Marco Caccamo. Toward the predictable integration of real-time cots based systems. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 73–82. IEEE, 2007.

[133] Paul Pop, Leonidas Tsiopoulos, Sebastian Voss, Oscar Slotosch, Christoph Ficek, Ulrik Nyman, and Alejandra Ruiz Lopez. Methods and tools for reducing certification costs of mixed-criticality applications on multi-core platforms: the recomp approach. *WICERT 2013*, 2013.

[134] Ragunathan Rajkumar, Kanaka Juvva, Anastasio Molano, and Shuichi Oikawa. Resource kernels: A resource-centric approach to real-time and multimedia systems. In *Photonics West'98 Electronic Imaging*, pages 150–164. International Society for Optics and Photonics, 1997.

[135] Jan Reineke, Sebastian Altmeyer, Daniel Grund, Sebastian Hahn, and Claire Maiza. Selfish-lru: Preemption-aware caching for predictability and performance. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 135–144. IEEE, 2014.

[136] Jan Reineke, Isaac Liu, Hiren D Patel, Sungjun Kim, and Edward A Lee. Pret dram controller: Bank privatization for predictability and temporal isolation. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 99–108, 2011.

[137] Pascal Richard. Analyse du temps de réponse des systèmes temps réel.

[138] Jakob Rosen, Alexandru Andrei, Petru Eles, and Zebo Peng. Bus access optimization for predictable implementation of real-time applications on multiprocessor systems-on-chip. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 49–60. IEEE, 2007.

[139] RTCA. *Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment.* RTCA, 1996.

[140] RTCA. *Design Assurance Guidance for Airborne Electronic Hardware.* RTCA, 2000.

[141] RTCA. *"Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations".* RTCA, 2005.

[142] Saowanee Saewong, Ragunathan Rajkumar, John P Lehoczky, and Mark H Klein. Analysis of hierarchical fixed-priority scheduling. In *ECRTS*, volume 2, page 173, 2002.

[143] Prabhat Kumar Saraswat, Paul Pop, and Jan Madsen. Task mapping and bandwidth reservation for mixed hard/soft fault-tolerant embedded systems. In *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 89–98. IEEE, 2010.

[144] John E. Sasinowski and Jay K. Strosnider. A dynamic programming algorithm for cache/memory partitioning for real-time systems. *IEEE Transactions on Computers*, 42(8):997–1001, 1993.

[145] Alexander Schiendorfer. Constraint programming for hierarchical resource allocation. In *Organic Computing: Doctoral Dissertation Colloquium 2014*, volume 4, page 57. kassel university press GmbH, 2014.

[146] Simon Schliecker, Mircea Negrean, and Rolf Ernst. Bounding the shared resource load for the performance analysis of multiprocessor systems. In *Proceedings of the conference on design, automation and test in Europe*, pages 759–764. European Design and Automation Association, 2010.

[147] Martin Schlueter. *Nonlinear mixed integer based optimization technique for space applications*. PhD thesis, University of Birmingham, 2012.

[148] Reinhard Schneider, Dip Goswami, Alejandro Masrur, and Samarjit Chakraborty. Qoc-oriented efficient schedule synthesis for mixed-criticality cyber-physical systems. In *Specification and Design Languages (FDL), 2012 Forum on*, pages 60–67. IEEE, 2012.

[149] Martin Schoeberl. A java processor architecture for embedded real-time systems. *Journal of Systems Architecture - Embedded Systems Design*, 54(1-2):265–286, 2008.

[150] Martin Schoeberl, Sahar Abbaspour, Benny Akesson, Neil Audsley, Raffaele Capasso, Jamie Garside, Kees Goossens, Sven Goossens, Scott Hansen, Reinhold Heckmann, et al. T-crest: Time-predictable multi-core architecture for embedded systems. *Journal of Systems Architecture*, 61(9):449–471, 2015.

[151] Andreas Schranzhofer, Rodolfo Pellizzoni, Jian-Jia Chen, Lothar Thiele, and Marco Caccamo. Worst-case response time analysis of resource access models in multi-core systems. In *Proceedings of the 47th Design Automation Conference*, pages 332–337. ACM, 2010.

[152] I Shin and I Lee. Compositional real-time schedulability analysis. *Handbook of Real-Time and Embedded Systems, I. Lee, JY-T. Leung, and SH Son, Eds. Chapman & Hall/CRC*, 2007.

[153] Insik Shin and Insup Lee. Periodic resource model for compositional real-time guarantees. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 2–13. IEEE, 2003.

[154] Insik Shin and Insup Lee. Compositional real-time scheduling framework. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 57–67. IEEE, 2004.

[155] Frank Singhoff, Jérôme Legrand, Laurent Nana, and Lionel Marcé. Cheddar: a flexible real time scheduling framework. In *ACM SIGAda Ada Letters*, volume 24, pages 1–8. ACM, 2004.

[156] Dario Socci, Peter Poplavko, Saddek Bensalem, and Marius Bozga. Multiprocessor scheduling of precedence-constrained mixed-critical jobs. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*, pages 198–207. IEEE, 2015.

[157] Jean Souyris, Erwan Le Pavec, Guillaume Himbert, Guillaume Borios, Victor Jégu, and Reinhold Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *OASIcs-OpenAccess Series in Informatics*, volume 1. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2007.

[158] Jan Staschulat, Simon Schliecker, and Rolf Ernst. Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In *17th Euromicro Conference on Real-Time Systems (ECRTS'05)*, pages 41–48. IEEE, 2005.

[159] Vivy Suhendra and Tulika Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *Proceedings of the 45th annual Design Automation Conference*, pages 300–303. ACM, 2008.

[160] Domitian Tamas-Selicean and Paul Pop. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 24–33. IEEE, 2011.

[161] Yudong Tan and Vincent Mooney. A prioritized cache for multi-tasking real-time systems. In *Proc., SASIMI*, 2003.

[162] Salvador Trujillo, Alfons Crespo, Alejandro Alonso, and Jon Pérez. Multipartes: Multi-core partitioning and virtualization for easing the certification of mixed-criticality systems. *Microprocessors and Microsystems*, 38(8):921–932, 2014.

[163] Theo Ungerer, Christian Bradatsch, Mike Gerdes, Florian Kluge, Ralf Jahr, Jörg Mische, Joao Fernandes, Pavel G Zaykov, Zlatko Petrov, Bert Böddeker, et al. parmerasa–multi-core execution of parallelised hard real-time applications supporting analysability. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 363–370. IEEE, 2013.

[164] Theo Ungerer, Francisco J Cazorla, Pascal Sainrat, Guillem Bernat, Zlatko Petrov, Christine Rochange, Eduardo Quinones, Mike Gerdes, Marco Paolieri, Julian Wolf, et al. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 5(30):66–75, 2010.

[165] Jacobus Reinier Van Kampenhout. *Deterministic Task Transfer in Network-on-Chip Based Multi-Core Processors*. PhD thesis, TU Delft, Delft University of Technology, 2011.

[166] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.

[167] Christopher B Watkins and Randy Walter. Transitioning from federated avionics architectures to integrated modular avionics. In *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pages 2–A. IEEE, 2007.

[168] Ingomar Wenzel, Raimund Kirner, Bernhard Rieder, and Peter Puschner. Measurement-based timing analysis. In *International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, pages 430–444. Springer, 2008.

[169] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.

[170] Reinhard Wilhelm, Christian Ferdinand, Christoph Cullmann, Daniel Grund, Jan Reineke, and Benoît Triquet. Designing predictable multi-core architectures for avionics and automotive systems. In *Workshop on Reconciling Performance with Predictability (RePP)*, volume 10, pages 2–3. Citeseer, 2009.

[171] Reinhard Wilhelm and Jan Reineke. Embedded systems: Many cores - many problems. In *7th IEEE International Symposium on Industrial Embedded Systems, SIES 2012, Karlsruhe, Germany, June 20-22, 2012*, pages 176–180, 2012.

[172] Alex Wilson and Thierry Preyssler. Incremental certification and integrated modular avionics. In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 1–E. IEEE, 2008.

[173] Jun Yan and Wei Zhang. Wcet analysis for multi-core processors with shared l2 instruction caches. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 80–89. IEEE, 2008.

[174] Ying Yi, Wei Han, Xin Zhao, Ahmet T Erdogan, and Tughrul Arslan. An ilp formulation for task mapping and scheduling on multi-core architectures. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 33–38. IEEE, 2009.

[175] Man-Ki Yoon, Jung-Eun Kim, Richard Bradford, and Lui Sha. Holistic design parameter optimization of multiple periodic resources in hierarchical scheduling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1313–1318. EDA Consortium, 2013.

[176] Man-Ki Yoon, Jung-Eun Kim, and Lui Sha. Wcet-aware optimization of shared cache partition and bus arbitration for hard real-time multicore systems. 2011.

[177] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, and Rodolfo Pellizzoni. Palloc: Dram bank-aware memory allocator for performance isolation on multicore platforms. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 155–166. IEEE, 2014.

[178] Heechul Yun, Rodolfo Pellizzon, and Prathap Kumar Valsan. Parallelism-aware memory interference delay analysis for cots multicore systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 184–195. IEEE, 2015.

[179] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 299–308. IEEE, 2012.

[180] Juan Zamorano and JA de la Puente. Open source implementation of hierarchical scheduling for integrated modular avionics. In *Proceedings of Real-Time Linux Workshop*, 2010.

[181] Michael Zimmer, David Broman, Chris Shaver, and Edward A Lee. Flexpret: A processor platform for mixed-criticality systems. In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 101–110. IEEE, 2014.