Toward Contention Analysis for Parallel Executing Real-Time Tasks

Fabrice Guet
ONERA - The French Aerospace Lab, Toulouse, France

Luca Santinelli
ONERA - The French Aerospace Lab, Toulouse, France

Jérôme Morio
ONERA - The French Aerospace Lab, Toulouse, France

Guillaume Phavorin
IRT Saint Exupery, Toulouse, France

Eric Jenn
IRT Saint Exupery, Toulouse, France

Abstract

In measurement-based probabilistic timing analysis, the execution conditions imposed to tasks as measurement scenarios, have a strong impact to the worst-case execution time estimates. The scenarios and their effects on the task execution behavior have to be deeply investigated. The aim has to be to identify and to guarantee the scenarios that lead to the maximum measurements, i.e. the worst-case scenarios, and use them to assure the worst-case execution time estimates.

We propose a contention analysis in order to identify the worst contentions that a task can suffer from concurrent executions. The work focuses on the interferences on shared resources (cache memories and memory buses) from parallel executions in multi-core real-time systems. Our approach consists of searching for possible task contenders for parallel executions, modeling their contentiousness, and classifying the measurement scenarios accordingly. We identify the most contentious ones and their worst-case effects on task execution times. The measurement-based probabilistic timing analysis is then used to verify the analysis proposed, qualify the scenarios with contentiousness, and compare them. A parallel execution simulator for multi-core real-time system is developed and used for validating our framework.

The framework applies heuristics and assumptions that simplify the system behavior. It represents a first step for developing a complete approach which would be able to guarantee the worst-case behavior.

2012 ACM Subject Classification Computer systems organization → Real-time system architecture

Keywords and phrases Contention analysis, parallel executions, measurement-based probabilistic timing analysis, probabilistic worst-case execution time

Digital Object Identifier 10.4230/OASIcs.WCET.2018.4

1 Introduction

Today's multi- and many-core platforms provide an amount of computational resource unconceivable a decade ago for real-time systems. While performance increases due to the availability of multiple cores and the possibility for parallel execution, the determinism is heavily challenged by the use of optimization features like cache memories or pipelines.

18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018).
Editor: Florian Brandner; Article No. 4; pp. 4:1–4:13

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

With multiple cores being accessible, task concurrently running (co-running) on different cores suffer from interferences while racing for shared resources like cache memories and buses. Due to concurrent accesses and bottlenecks, shared resource interferences have a prominent impact on tasks executions. Timing anomalies and worst-case conditions appear more often and the worst-case task execution time increases.

Isolation techniques and deterministic policies like Round Robin bus scheduling may be applied for reducing interferences. They would come at the cost of expensive implementations and decreased average performances. A valid alternative would consist in effectively modeling the interferences that each task suffers under different execution conditions. This way, the interference effects can be made predictable resulting into deterministic models to tasks and system behaviors.

Probabilistic models are emerging as flexible and reliable representations for tasks worst-case executions [7]. In those models, the classical deterministic Worst-Case Execution Time (WCET) is generalized with a probability distribution, the probabilistic WCET (pWCET), where it is quantified how likely an execution time may be exceeded.

Measurement-Based Probabilistic Timing Analysis (MBPTA) can be used for estimating pWCETs. It is sensitive to the measurement scenario which has been considered for measuring execution times [1, 13]. By measurement scenario it is intended, for example, a specific task mapping on multi-core processors, specific task inputs, environmental conditions, etc.. Each scenario would enforce a particular interference pattern on system resources with its specific impact to the task behavior; a scenario is representative also of interference conditions. The trace of measurements depends on such scenarios and the EVT exploits the worst-case specific to the scenario applied. The pWCET estimate would be the worst-case for only the specific scenario applied [8, 13].

In order to have safe pWCET estimates, it is necessary to determine the scenario that leads to the maximum execution time measurements, and consequently to the maximum pWCET estimate. The scenario exploration has to be efficient, since the measurement scenarios within multi-core systems can be in huge number, and reliable in offering the maximum pWCETs.

Contributions. In this work, we propose a contention analysis to explore the measurements scenarios for parallel real-time applications executed within multi-core platforms. The goal is to characterize all the scenarios and identify the worst-case from which to estimate the maximum pWCET. We name it contention analysis because it focuses on modeling interferences from contentions within cache memories and memory buses. At this stage, we do not deal with data synchronization problems e.g., deadlocks in parallel executions. Graph analyses, interference models, and contentiousness metrics are developed to characterize the worst parallel execution condition that tasks may suffer. The MBPTA is used for qualifying and comparing the execution scenarios.

We target the case of multi-core platform with shared cache, and a parallel execution simulator is developed and applied to an avionic case study. At this stage, the solution proposed is a partial one, which applies heuristics and assumptions that simplify the system behavior. It represents a first step for developing a complete approach which would be able to guarantee the worst-case behavior.

Organization of the paper. Section 2 presents the background in terms of computational modeling and analysis tools applied. Section 3 details the contention analysis and its main contributions. Analysis complexity and safety guarantees for pWCET estimates are outlined. In Section 4, it is described the experimental setup and the results of the simulation-based evaluation. Section 5 is for conclusions and future work.

Related Work. Measurement-based [probabilistic] timing analysis relies on measurements of actual task execution times for estimating either deterministic or probabilistic upper-bounds [9, 7]. An open challenge to them is the coverage problem and the so called confidence/representativity of the input measurements [9]. Our work particularly addresses this challenge with regard to task parallel execution and contention due to shared memory and memory bus.

Within a probabilistic framework, the definition of measurement scenarios and the confidence in the estimate can be addressed [1]. The confidence is related to the observation of events whose probability of occurrence is very low e.g., 10^{-15}. In our work the confidence defines the ability of selecting the worst scenario as the scenario which defines the task worst-case execution times.

In multi-core settings with competition to access shared resources, the combination of local cache misses and interference delay can be large and highly variable. The analysis of contentions represents a big challenge for the predictability of real-time embedded systems. In recent years, some progress on WCET and memory interference analysis has been achieved for multi-core systems. Some approaches have considered the impact that contention has on WCET estimates [11, 3]. They act to enrich static timing analysis models accounting for interference impacts. Some other approaches' goal is to bound interference with scheduling choices [17]. Our work copes with MBPTA and the contention analysis characterizes worst-case execution scenarios for guaranteeing more confident pWCET estimates.

For validating our approach, it has been developed a processor simulator which is a simplified implementation of a multi-core processor. We use it to test the solution proposed and to validate the effect of certain changes to the system behavior. The simulator enables characterizing the simultaneous accesses by the cores to shared resources like cache memories and memory bus. With the assumptions made, the simulator oversimplifies the system behavior focusing on memory accessing only. The assumptions made to develop it will be released in future works in the effort to complete system modeling and converging to realistic system behaviors.

Existing simulators have been investigated before developing our own. For example, the gem5 simulator is a highly configurable architecture simulator that supports different computing architectures like ARM. It tends to simulate the real behavior of the system, but it makes it quite complex to extrapolate specific behaviors since it takes into account too many mechanisms that could happen.

SimSo [6] is a multi-core scheduling simulator that takes into account cache temporal impacts. In order to compute cache effects on task executions, SimSo makes use of the frequency of access model [6, 5]. As the number of cycles per instruction depends on the state of the processor, the best way to model is to use state machines instead of a deterministic function like the frequency of access model. Thus, the need for more realism with respect to the system behavior which is driving the development of our simulator. On the other hand, multi-core and cache aspects of SimSo inspired our work. Our cache simulator makes use of a trace of memory addresses to access, and so execute address by address. The multi-core simulation is represented with accesses concurrently applied.

2 Models and Tools

Probabilistic Worst-Case Execution Time Model. Interferences and contentions do not only increase task execution times, they also bring variability to the task behavior at runtime. Probabilistic models can better catch the underlying task execution uncertainty than deterministic models.

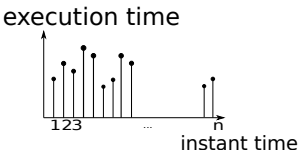

Figure 1 Trace $\mathscr{T}$.

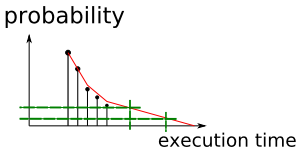

Figure 2 pWCET $\mathscr{C}^{\lambda}$ and two $\langle WCET; p\rangle$s.

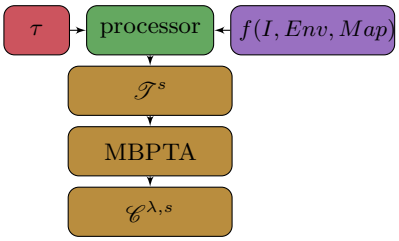

Figure 3 MBPTA classical work flow.

A sequence of execution time measurements C_j can be gathered in a trace $\mathscr{T} = (C_j)_{\ j\in[\![1;n]\!]}$. The MBPTA estimates pWCETs, denoted by $\mathscr{C}^{\lambda}$, by applying the EVT to a trace of measurements $\mathscr{T}$ [7, 8]. WCET thresholds $\langle WCET; p\rangle$ are extracted from $\mathscr{C}^{\lambda}$, and can be used to describe the task behavior, instead of using the whole distribution $\mathscr{C}^{\lambda}$. In $\langle WCET; p\rangle$, $WCET$ is the timing upper-bound on the task execution time and p is the probability for $WCET$ to be exceeded at runtime. With $\langle WCET_1; p_1\rangle$ and $\langle WCET_2; p_2\rangle$, by decreasing probability p $p_1 \leq p_2$, it is $WCET_1 \leq WCET_2$. This means that $WCET_1$ has more chances to be exceeded at runtime than $WCET_2$; the probability p can be seen as a level of confidence on the WCET threshold. Different probabilities can be considered e.g., 10^{-6}, 10^{-9} and 10^{-12} with the associated WCET thresholds.

Figure 1 and Figure 2 depict respectively, for an example task, a trace of execution time measurements and the pWCET estimate together with two WCET thresholds at given probabilities.

The MBPTA is sensitive to the execution scenario applied for measuring. A scenario s is an abstraction and represents a specific execution condition for the task and the system. It is an instantiation of the set of possible conditions e.g., task inputs I, environment state Env and task mapping Map. s is a function $f(I, Env, Map, \ldots)$ and it affects the behavior of the task and it can also change at runtime. *The trace of measurements $\mathscr{T}^s$ under s describes the expected execution behavior of the task under the condition. The pWCET estimation $\mathscr{C}^{\lambda,s}$ models the largest task execution times under s*, Figure 3.

An embedded real-time system has a finite number of execution scenarios $S = \{s_1, s_2, \ldots, s_k\}$. Among them, there would be the worst scenario s^{worst} as the scenario which ends up into the worst measurements and the worst pWCET estimates. The problem of enumerating all the $s \in S$, is a complex problem as it could exist a large, but finite, number of parameters defining the scenarios. This work aims at determining s^{worst} that includes the worst interference from cache and memory buses due to parallel executions. s^{worst} has to be guaranteed from an effective characterization of all the possible execution conditions, including the worst ones [9].

It is important to note that we call the pWCET from each trace "worst-case", but it is only the worst-case under the considered scenario. The contention analysis here is focusing on a subset of interferences/contentions conditions. It explores them efficiently, and it defines the worst scenario among them. For validating it we make use of the MBPTA tool called DIAGXTRM [8] which accepts traces of execution time measurements as input and it estimates pWCETs from those. DIAGXTRM evaluates the confidence of EVT applicability as well as the quality of the pWCET estimates for each measurement scenario applied. It also compares multiple scenarios in terms of both average and worst-case behaviors. DIAGXTRM is developed in R and is publicly available at `https://forge.onera.fr/projects/diagxtrm2`. DIAGXTRM and MBPTA in this work, are only used to verify the soundness of the contention analysis we propose.

Directed Acyclic Graph Model. In case of precedence constraints, the execution partial ordering between real-time tasks can be represented by a Directed Acyclic Graph (DAG) $G(V, E)$ where V is a set of N nodes and E is a set of directed edges [2, 10].

Each node $n_i \in V$ corresponds to a task and it can be weighted $w(n_i)$ by the task pWCET $\mathscr{C}_i^{\lambda}$ or WCET thresholds $\langle WCET; p \rangle$. Edges represent the order of execution between the tasks. The edge $e_{i,j}$ directly connects two nodes n_i and n_j, with n_i preceding n_j. An entry node in a DAG is a node with no predecessors; an exit node is a node without successors.

The precedence constraints encoded in DAGs impose that a node cannot start its execution before all his predecessors ended theirs. Edges can be weighted $w(e_{i,j})$ representing the communication delay which postpones task executions, i.e. task offsets. A path from n_i to n_j in a DAG exists if and only if it is possible to reach n_j from n_i; the path is the set of nodes from n_i to n_j and the sequence of edges, $\{\{n_i, n_k, n_r, \dots n_s, n_j\}, \{e_{i,k}, e_{k,r}, \dots e_{s,j}\}\}$. Tasks that are linked directly by sharing an edge or by a path are said to be functionally dependent because the activation of a task requires the termination of the other one. DAGs can be used to represent mono- and multi-rate task sets. In the latter, it necessary to define DAG reduction mechanisms with multiple task instances and communication buffers [12]. The buffers are for guaranteeing the correct communication pattern and the respect of the precedence constraints across multiple task occurrences.

3 Contentions from Parallel Executions

Parallelizing task executions, whenever it is possible, enables speeding up on average the real-time application. However, co-running tasks suffer from interferences and timing anomalies which could drastically reduce the worst-case performance. Those cases have to be scrupulously modeled in order to make the system predictable.

Independence Analysis. In case of precedence constraints, the tasks that can execute simultaneously on different cores are those functionally independent. Having $G(V, E)$ representing the real-time application, two tasks n_i and n_j are said to be independent, denoted $n_i \nabla n_j$, if and only if it does not exist any path from n_i to n_j in G. At runtime, independent tasks n_i and n_j are *contenders* since they can interfere with each other by introducing contention on shared resources.

The potential contenders $\Gamma(n_i)$ of a task n_i i.e. tasks that can execute in parallel to n_i or equivalently tasks independent from n_i: $\Gamma(n_i) = \{n_j \in G \mid n_i \nabla n_j\}$. Seeking for contenders consists in determining the complement of the undirected transitive closure[1] of the DAG. Then, by taking the complement graph $\overline{G}$ of the undirected transitive closure of the DAG, only independent tasks share an edge. The resulting graph $\overline{G}$ is called the graph of independences, as opposed to the initial G.

$\Gamma^*(n_i)$ is the set of tasks which are independent from each other and independent from n_i: $\Gamma^*(n_i) \stackrel{def}{=} \{n_j, n_k \in G \mid n_i \nabla n_j, n_i \nabla n_k, n_j \nabla n_k\}$, and is derived from $\overline{G}$. Note that, n_i is included in both $\Gamma(n_i)$ and $\Gamma^*(n_i)$ and $\Gamma^*(n_i) \subseteq \Gamma(n_i)$. We call this process *independence analysis*; the next steps are for exploring $\Gamma^*(n_i)$ and its possible subsets in the quest of contentiousness.

[1] The transitive closure allows adding an edge between two nodes if they are not independent i.e. if there exists a path from one to another.

Contender List. Given $\Gamma^*(n_i)$, we now seek for the list of possible contenders to n_i. A set of c tasks, including n_i, in which every couple of tasks share an edge in $\overline{G}$ consists of a clique $clique^c(n_i)$. The clique is relative to n_i and has size (cardinality) $c = |clique^c(n_i)|$. To note that any clique of n_i is a subset of $\Gamma^*(n_i)$, $clique^c(n_i) \subseteq \Gamma^*(n_i)$.

For each task n_i there exists a maximal size to its cliques. $clique^{\max}(n_i)$ is the maximum (largest in size) clique for n_i in $\overline{G}$; $clique^{\min}(n_i)$ denotes the minimum (smallest in size) clique for n_i in $\overline{G}$. Both the maximum clique and the minimum clique are not usually unique. For a $clique^c(n_i)$, the $c-1$ tasks $clique^c(n_i)/\{n_i\}$ ($clique^c(n_i)$ without n_i) are the potential contenders of n_i. The complete set of contender for n_i are all the cliques $clique^c s(n_i)/\{n_i\}$ for $c \in [|clique^{\min}(n_i)|, |clique^{\max}(n_i)|]$. For a M-core processor, only up to $M-1$ tasks can run in parallel to n_i. Then, cliques whose size c exceeds M are rejected because impossible to happen scenarios.

We assume that the more tasks are executed in parallel, the more interference between the co-running tasks there is. This is true with cache memories and the model we apply, where more concurrent tasks would evict larger portion of cache or more frequent eviction, increasing memory latencies. With memory buses it would be the same considering the same modeling with constant rates, where more concurrent accesses would increase memory communication latencies. Hence for, in order to identify the worst contention scenario, the contender list of n_i should only be composed of the largest sets of contender tasks within $\Gamma^*(n_i)$.

What can be called *valid contender list* of n_i $ContenderList(n_i)$, is defined as: $ContenderList(n_i) \stackrel{def}{=} \{clique^c(n_i)/n_i \ : \ c = \min(|clique^{\max}(n_i)|, M)\}$. The task sets in $ContenderList(n_i)$ are all of size $\min(|clique^{\max}(n_i)| - 1, M-1)$.

Contender Classification. The objective of the worst contention analysis is identifying for n_i, its $min(M-1, |clique^{\max}(n_i)|-1)$ worst contenders within $ContenderList(n_i)$. Running them in parallel together with n_i would foster the worst interferences for n_i.

The classical approach to worst contention analysis would consist in executing all the possible combinations for all the task sets in the contender list. However, the size of the contender list may be too large for an affordable exhaustive search. We define a learning procedure called *contender classification* in order to reduce the complexity of the worst contention analysis.

In this work we focus on contentions from the memory hierarchy and buses for parallel applications and co-running tasks. Inspired by it [15], we measure the number of accesses to the shared cache *accesses*. The number of accesses can be either lines fetched from the shared cache to the private caches, or writes to the shared cache. Such number of accesses has to be normalized, either in number of instructions or in time units t. We define the memory bandwidth usage *mem_band* as: $mem\_band \stackrel{def}{=} accesses/t$. The rationale behind the *mem_band* metric is that a task whose memory bandwidth usage is high, leads to a lot of potential interferences for co-running tasks. *mem_band* adds to the maximal cliques for the worst-case scenarios.

Tasks with different execution lengths would impact differently the contentions. If the task under analysis has a large execution time and runs together with small execution time tasks, then the larger task would not be impacted by interferences in the last part of its execution. We overcome this issue by letting all the tasks execute continuously for the entire execution of the task under analysis. This way, the task would experience the greatest amount of interference during its entire execution: there are not zones without contention. At this stage, we do not investigate the effects of offsets between tasks; future work will be devoted to

that. Figure 4 describes the assumption we make with different interference sections (vertical lines) from the two interfering tasks task_j and τ_k for task_i. The repeated executions are such that task_j is executed continuously twice and task_k is executed four times, while starting synchronously.

Given the *mem_band* defined as a constant rate, the synchronous case with repetition guarantees the maximum rate of contentiousness for the task under investigation. We are well aware that this is simplistic assumption and perhaps an unrealistic case, but it is a starting point to study parallel task contentiousness. Moreover, *mem_band* accounts only for the temporal interferences at the shared memory bus level. The approach in this work is an initial step toward an heuristic able to reduce the cost for identifying worst-case execution conditions. It is obvious that the final metric would take *mem_band* into account together with other effects, and has to be developed incrementally.

3.1 Contention Analysis Discussion

A task n_i running on a M-core processor Φ^M is denoted as $\Phi^M(n_i)$. Two tasks n_i and n_j running in parallel within Φ^M are denoted by $\Phi^M(n_i \parallel n_j)$; obviously n_i and n_j are running on different cores within Φ^M. The $\parallel$ notation also applies for p tasks $n_1 \parallel \ldots \parallel n_p$.

What we propose, is an approach for maximizing the execution time measurements with some possible worst interference task mapping configuration s^{worst} in order to produce a safe pWCET at system deployment. It is based on heuristics and restrictive assumptions to avoid exhaustive search. The main steps of the contention analysis can be summed up with the following four basic functions:

1. *Contender List Search* $\forall n_i \in G$, $ContenderList(n_i)$ is the result of the contender list search and the G transformation into $\overline{G}$. This step includes the independence analysis;
2. *Contentiousness Characterization*: $\forall n_i \in G$, $mem\_band(n_i)$ is measured using τ_{mon} in case of $\Phi^M\ (n_i||\tau_{mon}(t))$. $\tau_{mon}(t)$ is an artifact task used to monitor other task effect on shared resources;
3. *Contender Task Sets Classification*: $\forall n_i \in G$, $sort(ContenderList(n_i))$ sorts the task sets from the ones with the greatest sum of memory bandwidth usage values $sort(ContenderList(n_i))[\cdot]$ to those with the least sum of memory bandwidth usage values $sort(ContenderList(n_i))[|ContenderList(n_i)|]$;
4. *Worst Contention Scenario Measurements*: $\forall n_i \in G$, $\mathscr{T}(n_i)$ is the measurement trace under $s^{worst}\ \Phi^M\ (n_i||sort(ContenderList(n_i))[1])$ to which the EVT is applied.

This paper offers a narrow perspective to contentions from parallel executions for the assumptions and definitions made. The proposed contentions analysis guarantees the worst execution condition among those and accounted for here.

The computation time complexity of independence analysis and contention list search (step 1) is $O(3^{N/3})$ and depends on the number of tasks N, [16].

Thanks to their memory bandwidth usage, each task set in the contender list can be ordered from the most contentious to the least. The task contentiousness characterization and classification for all the N tasks in G would take $\sum_{i=1}^{N} t_{n_i}$, where t_{n_i} is the execution time of n_i. One execution per task is sufficient for the task contentiousness characterization because we consider single-path tasks. The simplistic single-path task assumption could resemble to an unrealistic case. Instead, it can be applied to any actual task representation where only the worst-case path is exercised all the time, As $\sum_{i=1}^{N} t_{n_i} < N \times T$, where T would be the largest task execution time in G, the computation time complexity of characterizing the contentiousness (step 2 and step 3) is $O(N)$ on the number of tasks.

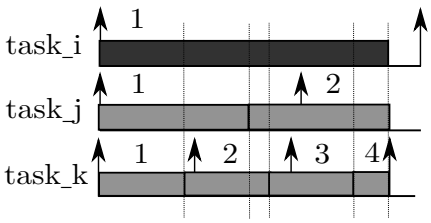

Figure 4 Task parallel execution with different zone of contention to task_i.

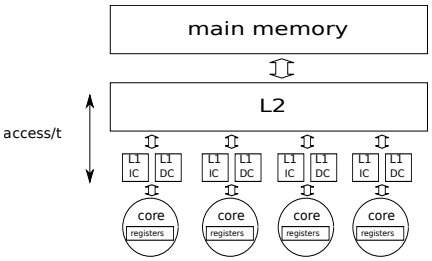

Figure 5 Memory hierarchy between cores with local and share memory relationship.

By executing n_i together with its most contentious tasks in its contender list, we assure the worst contention scenario s^{worst} for n_i. Measuring according to s^{worst} for all the tasks of the application (step 4) has computation time complexity $O(N)$ on the number of tasks.

4 Case Study and Simulation

In this section we present the case study used for the experimental evaluation of the proposed contention analysis. First, we briefly describe the simulator we develop for parallel execution and contention measurements. Then, we detail the real-time application we apply for validating our contention analysis. Finally, we outline the results of the contention analysis with the simulator developed and the application selected.

Platform Simulation. One of our driving interests is to develop a generalizable and realistic multi-core parallel execution simulator with memory hierarchy. The simulator allows observing and controlling specific resources like cache memory and memory bus.

The simulated architecture is composed of four cores, with two levels of cache, L1 Data Cache (DC) and Instruction Cache (IC), and L2. The L2 cache is shared between the four cores and is accessed through a bus. The cache memories make use of the LRU policy and the write-through policy. The L1 cache memories are 4-way associative with 8 blocks per set i.e. 32 blocks in total, and whose access penalty is 1 cycle. The L2 cache memories are 4-way associative with 32 blocks per set i.e. 128 blocks in total, and whose access penalty is 4 cycles. The bus makes use of the FIFO policy. This architecture is similar to the LEON4 processor [4].

The way it has been implemented, the simulator allows choosing for the size of the cache memories and their arbitration policy, as well as the bus policy. Tasks are modeled with a trace of memory accesses i.e. a sequence of reads or writes to a memory location. With this, we represent memory accesses and we cope with the defined *mem_band*. Such traces are randomly generated with different profiles, but they can be built from the assembly code of the task for a given platform. The task profiling with traces is generic enough to apply to different task characteristics. It would suffice adapt the memory access and reproduce different task behaviors. We stress what is randomly generated are memory accesses; cache misses results from the replacement policy implemented, and are not random.

More details about the platform simulator, like the core execution and the task profiling based on a trace of memory accesses, may be found at `https://forge.onera.fr/projects/multicore-simulator`. Figure 5 details the main elements already implemented on the simulator. *mem_band* defined describes the constant rate accesses to the cache memory shared between cores through the bus.

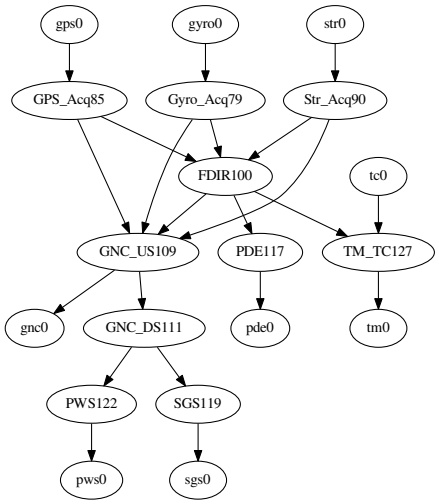

Figure 6 DAG G for the FAS case study with precedence relationships between tasks.

Table 1 Results of the contender list search applied to the FAS case study.

task	$\lvert clique^{\max}(n_i)\rvert$	$\#\ \Gamma(n_i)$
gps0	4	4
gyro0	4	4
str0	4	4
GPS_Acq85	4	4
Gyro_Acq79	4	4
Str_Acq90	4	4
FDIR100	1	1
tc0	4	26
GNC_US109	3	6
PDE117	4	31
TM_TC127	4	22
gnc0	4	50
GNC_DS111	4	6
pde0	4	31
tm0	4	22
PWS122	4	28
SGS119	4	28
pws0	4	28
sgs0	4	28

Application Setup. For this case study, we make use of the Flight Application Software (FAS) application of the Automated Transfer Vehicle designed by EADS Astrium Space Transportation for resupplying the International Space Station [12]. FAS is composed of 19 tasks and 21 precedence constraints, which can be represented by the DAG $G(V, E)$ given in Figure 6. It is a relatively small real-time application, but is already enough to show the advantage of using the contention analysis we propose.

For these experiments, we make use of the mono-rate version of the FAS application, but as already mentioned, our approach can cope with multi-rates as soon as their DAG representation is available. $\overline{G}$ for FAS is the closure representation of G; each arc from one node connects two independent tasks. Already with FAS, there exist lots of independent tasks, as high as 16 for task tc0. A smart and efficient contention analysis is much needed to avoid exhaustive exploration.

With respect to the implementation of the FAS tasks, we use a randomly generated trace of memory accesses for each task. Not knowing the exact original implementation, we have generated traces with different characteristic for the 19 tasks. They have been made from uniform distributions with different supports or Poisson distributions with different rates to reproduce different behaviors; the length of the traces is randomly picked from a uniform distribution. Task parameters like period and deadlines are defined according to [14]. The random generation of tasks profiles does not limit the generality of our work, since the contentiousness part would model the task for whatever access trace is considered. Also, real implementations can be included with measurements applied to characterize the access profiles.

4.1 Contention Analysis

Table 1 presents the results of the independence analysis and contender list search for the FAS case study, step 1 – $contenderList()$. As some contender lists are quite long, only the maximum clique size and the size of the contender list are given for each task. The maximal

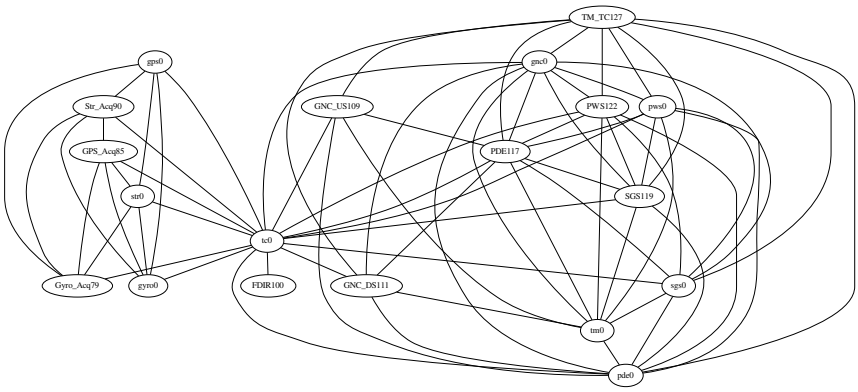

Figure 7 $\overline{G}$ for the FAS case study with independence relationships between tasks.

Table 2 Statistics of the contentiousness characterization for the tasks in the GPS_Acq85 contender list.

task	execution time (cycles)	L1 hit/misses	L2 hit/misses	bus accesses	*mem_band*	rank
GPS_Acq85	1025	123/177	93/84	204	0.68	-
Str_Acq90	811	225/75	28/47	118	0.40	2
tc0	271	89/11	0/11	34	0.34	3
gyro0	374	53/47	3/44	57	0.57	1
Gyro_Acq79	641	285/15	0/15	68	0.23	5
str0	261	94/6	0/6	32	0.32	4
gps0	355	55/45	5/40	54	0.54	-

found clique for all tasks in the graph is of size five. However, as the number of available processor is four, cliques of size greater than four are not considered for the analysis. Some tasks exhibit a high number of task sets in their contender list, up to fifty for the gnc0 task. That is the main motivation for the contender classification and to avoid evaluating all the combinations.

As representative example, we detail the worst contention analysis and its experimental evaluation for the task GPS_Acq85. From the independence analysis and the contender list search, the GPS_Acq85 contender list is: $ContenderList(\text{GPS\_Acq85}) = \{\{Str\_Acq90, tc0, gyro0\}, \{Str\_Acq90, tc0, Gyro\_Acq79\}, \{str0, tc0, gyro0\}, \{str0, tc0, Gyro\_Acq79\}\}$. There are six tasks to simulate for contentiousness and contention analysis.

All six tasks involved (GPS_Acq85, Str_Acq90, tc0, gyro0, Gyro_Acq79, str0) are first executed one time in isolation, $\Phi^4(n_j)$, in order to characterize their contentiousness. The results are given in Table 2 where execution times are in CPU cycles. Table 2 presents also the measurements of cache misses, bus access, and the resulting *mem_band*. As the platform simulator considered here has no prefetching mechanism, the memory bandwidth usage is computed as $mem\_band = \frac{bus\_accesses}{number\_of\_instructions}$, with the number of instructions specified by the task characteristics for the simulation. A ranking is attributed to each task, with the contentiousness defined according to the memory bandwidth usage *mem_band*. Rank 1 is for the largest memory bandwidth usage among the contenders. The task set *tsi* represents a possible execution scenario for GPS_Acq85 (cliques). The task ranking is propagated to the contender list, ordering the task sets from the most contentious set, denoted

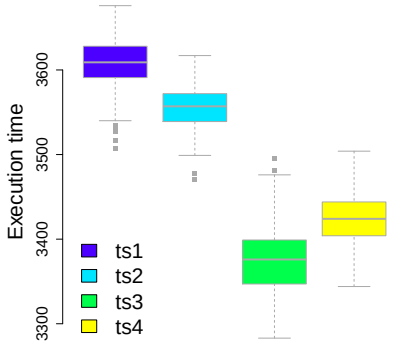

Figure 8 GPS_Acq85 and four interference scenarios tsi: expected behaviors and comparison with box plots.

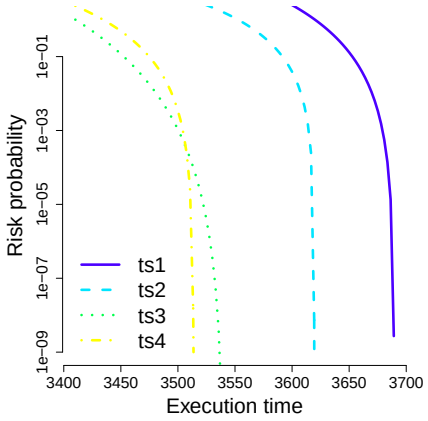

Figure 9 GPS_Acq85 and four interference scenarios tsi: the pWCET estimated and comparison in logarithmic scale and with the inverse cumulative distribution representation.

by $ts1$, to the least contentious task set $ts4$. It is $sort(ContenderList(\text{GPS\_Acq85})) = \{ts1, ts2, ts3, ts4\}$, with $ts1 = \{Str\_Acq90, tc0, gyro0\}$, $ts2 = \{str0, tc0, gyro0\}$, $ts3 = \{Str\_Acq90, tc0, Gyro\_Acq79\}$, and $ts4 = \{str0, tc0, Gyro\_Acq79\}$.

Once identified and ranked the tsi, for each tsi, 500 consecutive execution time measurements are obtained: the classification between tsis according to their contentiousness is: $\max(ts1) > \max(ts2) > \max(ts3) \simeq \max(ts4)$. The measurements, as expected behavior, validate the ranking proposed with the contentiousness. s^{worst} for GPS_Acq85 consists of executing it in parallel of the tasks in $ts1$ i.e. Φ^4 (GPS_Acq85||Str_Acq90||tc0||gyro0).

DIAGXTRM is applied to the four traces $\mathscr{T}^{tsi}$ and it provides the statistical analysis (box plots, first order, and second order statistics) as well as the worst-case analysis (pWCET) for each input trace. All the average behaviors are plotted in the box plot of Figure 8; the scenarios are compared on average. The pWCET estimates are plotted in Figure 9 with the inverse cumulative distribution representation. The worst pWCET, the greatest distribution between the four possible pWCETs, and it comes from the worst-case scenario $ts1$ $\mathscr{C}^{\lambda,ts1}$, $\Phi^4(\text{GPS\_Acq85}, ts1)$. The scenario comparison as in Figure 9 validates that the worst-case scenario among the 4 considered is the one from the most contentious clique. It is worth noting that DIAGXTRM applied to the 4 scenarios passes all the tests for confident pWCET estimates; only for space reasons this is not illustrated with a plot.

5 Conclusions and Future Works

We propose a contention analysis which enables measuring, at low cost, the worst contentions for parallel executing tasks. The maximum execution times are reproduced from the worst parallel execution conditions among those investigated. The whole procedure is an heuristic approach conceived to guarantee identifying the worst contention conditions and to reduce the computation costs. It represents a first step toward an efficient and complete contention analysis.

The proposed contention analysis framework still has some limitations that have to be addressed in future works. Among others, we intend to release the single-path assumption and consider multi-path tasks. Furthermore, we intend to enrich the contentiousness metric, mem_band to account for multiple effects.

References

1 J. Abella, E Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla. Heart of gold: Making the improbable happen to increase confidence in mbpta. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 255–265. IEEE, 2014.

2 A. Al Badawi and A. Shatnawi. Static scheduling of directed acyclic data flow graphs onto multiprocessors using particle swarm optimization. *Computers & Operations Research*, 40(10):2322–2328, 2013.

3 Björn Andersson, Arvind Easwaran, and Jinkyu Lee. Finding an upper bound on the increase in execution time due to contention on the memory bus in cots-based multicore systems. *SIGBED Rev.*, 7(1):4:1–4:4, 2010.

4 J. Andersson, J. Gaisler, and R. Weigand. Next generation multipurpose microprocessor. In *Int. Conf. on Data Systems in Aerospace (DASIA), Hungary*, 2010.

5 Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 340–351. IEEE, 2005.

6 Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms. In *5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–p, 2014.

7 L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *the 24th Euromicro Conference on Real-Time Systems*, 2012.

8 F. Guet, L. Santinelli, and J. Morio. On the reliability of the probabilistic worst-case execution time estimates. In *8th European Congress on Embedded Real Time Software and Systems (ERTS)*, 2016.

9 S. Law and I. Bate. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *28th Euromicro Conference on Real-Time Systems ECRTS, Proceedings*, 2016.

10 Alessandra Melani, Marko Bertogna, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, and Giorgio Buttazzo. Response-time analysis of conditional DAG tasks in multiprocessor systems. In *27th Euromicro Conference on Real-Time Systems (ECRTS)*, Lund, Sweden, July, 2015.

11 J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 109–118, 2014.

12 Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, Sep 2015.

13 L. Santinelli, F. Guet, and J. Morio. Revising measurement-based probabilistic timing analysis. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE*, pages 199–208. IEEE, 2017.

14 L. Santinelli, W. Puffitsch, A. Dumerat, F. Boniol, C. Pagetti, and V. Jegu. A grouping approach to task scheduling with functional and non-functional requirements. In *Embedded real-time software and systems (ERTS)*, 2014.

15 L. Tang, J. Mars, and M. L. Soffa. Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures. In *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*, pages 12–21. ACM, 2011.

16 E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

17 G. Yao, R. Pellizzoni, S. Bak, H. Yun, and M. Caccamo. Global real-time memory-centric scheduling for multicore systems. *IEEE Transactions on Computers*, 65:2739–2751, 2016.