UNIVERSITÉ DE MONTRÉAL

HANDLING INFORMATION AND ITS PROPAGATION TO ENGINEER COMPLEX EMBEDDED SYSTEMS

IMANE HAFNAOUI
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2018

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

HANDLING INFORMATION AND ITS PROPAGATION TO ENGINEER COMPLEX EMBEDDED SYSTEMS

présentée par: HAFNAOUI Imane
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

M. ADAMS Bram, Doctorat, président
M. BELTRAME Giovanni, Ph. D., membre et directeur de recherche
Mme NICOLESCU Gabriela, Doctorat, membre et codirectrice de recherche
Mme BOUCHENEB Hanifa, Doctorat, membre
Mme MOSES Melanie, Ph. D., membre externe

DEDICATION

To all the black sheep...
Who embraced their...black sheepness...

And to my parents...
Who, despite everything...
Helped this black sheep flourish.

ACKNOWLEDGEMENTS

The journey that lead to this point wouldn't have been possible without the support of special people that I crossed paths with during the duration of this research. I am superbly grateful for Giovanni Beltrame whom opened doors to many an opportunity, for supporting my 'free agent' mentality and for giving the right advice when I most needed. I want to thank Gabriela Nicolescu for the continued support, Rabeh Ayari, my partner in crime and primary commiserator, Jacopo for the deep philosophical discussions, Chao for his patience and everybody I had the chance to meet and work with from the MistLab.

I extend my thanks to Houle-sensei (who hates being called so!) and the members of his laboratory in Tokyo for both expending my view of research, opening doors to interesting collaborations and the cultural experience. It was fantastic to have the opportunity to end my journey within your lab.

I am grateful for my parents and siblings whom, despite the distance, held me up when I was struggling and for their unwavering trust and confidence in my abilities. I am also grateful for my husband for going through hell and back to make this experience as comfortable as possible through moral and emotional support.

Last but not least, I feel blessed for crossing path with all the people that ultimately shaped this work, and myself in accordance, whom through shared circumstances became part of my family and hold a special place in my life.

Thanks for all your encouragements.

RÉSUMÉ

Avec l'intérêt que la technologie d'aujourd'hui a sur les données, il est facile de supposer que l'information est au bout des doigts, prêt à être exploité. Les méthodologies et outils de recherche sont souvent construits sur cette hypothèse. Cependant, cette illusion d'abondance se brise souvent lorsqu'on tente de transférer des techniques existantes à des applications industrielles. Par exemple, la recherche a produit divers méthodologies permettant d'optimiser l'utilisation des ressources de grands systèmes complexes, tels que les avioniques de l'Airbus A380. Ces approches nécessitent la connaissance de certaines mesures telles que les temps d'exécution, la consommation de mémoire, criteres de communication, etc. La conception de ces systèmes complexes a toutefois employé une combinaison de compétences de différents domaines (probablement avec des connaissances en génie logiciel) qui font que les données caractéristiques au système sont incomplètes ou manquantes. De plus, l'absence d'informations pertinentes rend difficile de décrire correctement le système, de prédire son comportement, et améliorer ses performances.

Nous faisons recours au modèles probabilistes et des techniques d'apprentissage automatique pour remédier à ce manque d'informations pertinentes. La théorie des probabilités, en particulier, a un grand potentiel pour décrire les systèmes partiellement observables. Notre objectif est de fournir des approches et des solutions pour produire des informations pertinentes. Cela permet une description appropriée des systèmes complexes pour faciliter l'intégration, et permet l'utilisation des techniques d'optimisation existantes.

Notre première étape consiste à résoudre l'une des difficultés rencontrées lors de l'intégration de système: assurer le bon comportement temporelle des composants critiques des systèmes. En raison de la mise à l'échelle de la technologie et de la dépendance croissante à l'égard des architectures à multi-cœurs, la surcharge de logiciels fonctionnant sur différents cœurs et le partage d'espace mémoire n'est plus négligeable. Pour tel, nous étendons la boîte à outils des système temps réel avec une analyse temporelle probabiliste statique qui estime avec précision l'exécution d'un logiciel avec des considerations pour les conflits de mémoire partagée. Le modèle est ensuite intégré dans un simulateur pour l'ordonnancement de systèmes temps réel multiprocesseurs.

Pour remédier à l'absence des spécification des modules d'un système, nous proposons une méthodologie permettant d'estimer les performances logicielles. Notre méthode fonctionne avec un accès limité à la plate-forme matérielle finale sur laquelle le système doit être déployé. Nous y parvenons en tirant profit de la précision des simulateurs matériels et de la rapidité

des techniques de régression, permettant ainsi une exploration plus rapide et plus étendue de l'espace de conception et une vérification rapide du système intégré.

Comme nous reconnaissons l'importance des informations pertinentes pour amener des connaissances fondamentales à la mise au point des systèmes complexes, nous concentrons nos travaux de recherche sur la propagation de l'information et son influence sur la compréhension et le développement du comportement des systèmes. Bien que cette propriété de l'information soit depuis longtemps adoptée dans l'étude des systèmes biologiques, elle n'a pas encore été exploitée dans le contexte des systèmes artificiels. Les systèmes multi-agents, tels que les essaims de robots et un grand nombre d'applications Internet des objets (IoT), font partis de cette catégorie dans laquelle la communication entre les agents conduit à un comportement observé au niveau du système. Avec l'incertitude qui entoure la transmission d'information entre agents, nous proposons un modèle probabiliste de propagation d'informations qui se base sur peu d'hypothèses de la propagation, le système ou son environnement. Avec cela, nous cherchons à révéler le potentiel de l'étude de la propagation de l'information pour expliquer les événements observés.

Cela ouvre à son tour un champ de potentiel à explorer afin de concevoir des systèmes plus sûrs, plus prévisibles et optimisés. En fait, nous développons une approche permettant d'obtenir une description pertinente de systèmes complexes de grande taille sous la forme d'un simple graphe à partir d'une connaissance limitée des modules du système, afin de faciliter davantage l'intégration de système et l'accès libre à un large éventail de techniques proposées dans la littérature. .

Nous exploitons également le modèle de propagation d'information dans la conception des systèmes plus efficaces tels que les réseaux de capteurs sans fil avec une approche pour identifier les éléments faibles de ces réseaux afin de stimuler la propagation de l'information. Certains de ces systèmes sont et feront l'objet de spécifications plus strictes, notamment en termes de temps, car leur utilisation s'orientera vers des applications plus critiques, telles que la conception d'essaims de robots en tant que secouristes. Nous abordons cette question en proposant une analyse probabiliste du temps qui étend le modèle d'information développé pour estimer les pires temps de propagation de l'information.

De notre point de vue, le travail dans cette thèse fournit des approches utiles pour construire des ponts entre le travail académique appliqué et les besoins industriels, ainsi qu'une nouvelle vision sur l'ingénierie des systèmes modernes d'un point de vue biologique.

ABSTRACT

In today's data-driven technology, it is easy to assume that information is at the tip of our fingers, ready to be exploited. Research methodologies and tools are often built on top of this assumption. However, this illusion of abundance often breaks when attempting to transfer existing techniques to industrial applications. For instance, research produced various methodologies to optimize the resource usage of large complex systems, such as the avionics of the Airbus A380. These approaches require the knowledge of certain metrics such as the execution time, memory consumption, communication delays, etc. The design of these complex systems, however, employs a mix of expertise from different fields (likely with limited knowledge in software engineering) which might lead to incomplete or missing specifications. Moreover, the unavailability of relevant information makes it difficult to properly describe the system, predict its behavior, and improve its performance.

We fall back on probabilistic models and machine learning techniques to address this lack of relevant information. Probability theory, especially, has great potential to describe partially-observable systems. Our objective is to provide approaches and solutions to produce relevant information. This enables a proper description of complex systems to ease integration, and allows the use of existing optimization techniques.

Our first step is to tackle one of the difficulties encountered during system integration: ensuring the proper timing behavior of critical systems. Due to technology scaling, and with the growing reliance on multi-core architectures, the overhead of software running on different cores and sharing memory space is no longer negligible. For such, we extend the real-time system tool-kit with a static probabilistic timing analysis technique that accurately estimates the execution of software with an awareness of shared memory contention. The model is then incorporated into a simulator for scheduling multi-processor real-time systems.

To address the lack of system module specification, we propose a methodology to estimate software performance. Our method works with limited-access to the final hardware platform on which the system is to be deployed. We achieve this by leveraging the accuracy of hardware simulators and speed of regression techniques, thereby enabling a faster and wider exploration of the design space, and an early verification of the integrated system.

As we recognize the importance of relevant information in bringing fundamental knowledge to engineering complex systems, we shift the focus of our research towards studying how information propagates and its influence on understanding and developing the behavior of systems. Although this property of information has long been adopted in the study of bio-

logical systems, it has yet to be exploited in the context of artificial systems. Multi-agent systems, such as robot swarms and a great number of Internet of Things (IoT) applications fall under this category in which the communication among the agents leads to an observed behavior at system level. With the uncertainty that surrounds the transmission of the information between agents, we propose a probabilistic model of information flow that makes very few assumptions about the propagation, the system and its environment. In doing so, we seek to reveal the potential of studying information propagation in explaining observed events.

This in turn opens a field of potential to investigate in order to engineer safer, more predictable, and optimized systems. As a matter of fact, we develop an approach to obtain a relevant description of large complex systems in the form of a simple graph from limited knowledge about the system modules to further ease system integration and open access to a wide range of techniques proposed in literature. We also exploit the information flow model in engineering more efficient systems such wireless sensor networks with an approach to identify the weak elements in these networks in order to boost the information propagation. Some of these systems are and will experience stricter specifications, timing in particular, as their use turn towards more critical applications such the design of robot swarms as emergency responders. We approach this issue by proposing a probabilistic timing analysis that extends the developed information model to estimate worst times for information propagation.

In our opinion, the work in this thesis provides useful approaches to build bridges between applied academic work and industrial needs, as well as a fresh look at engineering modern systems from a biological perspective.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

RÉSUMÉ v

ABSTRACT vii

TABLE OF CONTENTS ix

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF SYMBOLS AND ABBREVIATIONS xv

LIST OF APPENDICES xvi

CHAPTER 1 INTRODUCTION 1

1.1 Context and Motivation 1

1.2 Problem Statement 3

1.3 Research Objectives 3

1.4 Contributions and Impact 4

1.5 Dissertation Organization 6

CHAPTER 2 LITERATURE REVIEW 7

2.1 Information as Stored Data 7

2.1.1 Describing and Integrating Complex Systems 8

2.1.2 Timing Analysis of Real-time Systems 9

2.2 Information as an Emergent Property 11

2.2.1 Information Flow 11

2.2.2 From information flow to Breaking the status-quo 12

CHAPTER 3 ARTICLE 1 – A SIMULATION-BASED MODEL GENERATOR FOR SOFTWARE PERFORMANCE ESTIMATION 14

3.1 Introduction 15

3.2 Related Work 16
3.3 Proposed Approach 18
3.3.1 Linear Regression Models 18
3.3.2 Model Generator 20
3.4 Performance Evaluation 23
3.4.1 Experimental Setup 23
3.4.2 Experimental results 24
3.5 Conclusions 27

CHAPTER 4 ARTICLE 2 – AN ANALYSIS OF RANDOM CACHE EFFECTS ON REAL-TIME MULTI-CORE SCHEDULING ALGORITHMS 29
4.1 Introduction 30
4.2 Contention-Aware Scheduling with Probabilistic Caches 32
4.2.1 Interleaved Reuse Distance 32
4.2.2 Static Probabilistic Timing Analysis of Random Caches 34
4.2.3 Random Cache Model 35
4.2.4 Execution Time Model Integration 37
4.3 Experimental Evaluation 38
4.3.1 Experimental Setup 38
4.3.2 Experimental Results 38
4.4 Conclusions 44

CHAPTER 5 ARTICLE 3 – SCHEDULING REAL-TIME SYSTEMS WITH CYCLIC DEPENDENCE USING DATA CRITICALITY 45
5.1 Introduction 46
5.2 Related work 49
5.3 System Model 51
5.4 Graph Transformation Through Data Criticality 52
5.4.1 Error Propagation Approach (EPA) 52
5.4.2 Motivational Example 59
5.5 Error Propagation Approach: Experimental Evaluation 61
5.6 Case studies 64
5.6.1 Voice-band Data Modem 64
5.6.2 Full-Mission Simulator 65
5.7 Conclusions 67

CHAPTER 6 ARTICLE 4 – TIME IS OF THE ESSENCE: SPREADING INFORMATION AMONG INTERACTING GROUPS 69
6.1 Introduction . . . 70
6.2 Information from an Observation to a Conviction . . . 73
6.3 Probabilistic Worst Case Convergence Time . . . 75
6.4 Results . . . 77
6.4.1 A Rumour and its Counter-Rumour . . . 77
6.4.2 Timing and Resilience . . . 80
6.5 Discussion . . . 84
6.6 Methods . . . 85
6.6.1 Group interaction model . . . 85
6.6.2 Conviction Through Message Passing . . . 85
6.6.3 Probabilistic Worst Case Convergence Time . . . 87
6.6.4 Rumour Data Collection . . . 90
6.6.5 pWCCT Experiments . . . 91

CHAPTER 7 GENERAL DISCUSSION 94
7.1 Bridging the gap . . . 94
7.2 A Hidden Simplicity . . . 96
7.3 Information Propagation in Artificial Systems . . . 96

CHAPTER 8 CONCLUSIONS 100
8.1 Knowledge Transfer and Self-Reflection . . . 101
8.2 Future Ventures . . . 102

REFERENCES . . . 103

APPENDICES . . . 117

LIST OF TABLES

Table 3.1 Reference and Target Characteristics. 23
Table 3.2 Performance of Regression Model Built on Measured Data. 26
Table 3.3 Performance of Regression Model Built on Simulated Data. . . . 26
Table 4.1 Task-set Γ specifications with every task's release time at $A_i = 0$ and $t = \{C_t, D_t, T_t, cpu_t\}$ defined as the worst case execution time (used for no contention), deadline, period and allocation (when applicable) respectively. 40
Table 5.1 Probability of a component propagating an error from its inputs to its outputs. 60
Table 5.2 CEP for edges in the overlapping cycles Γ_2 and Γ_3 and non-overlapping cycle Γ_1. 61
Table 5.3 Number of removed edges as connectivity grows. 65
Table 5.4 CEP for edges in cycle Γ 66
Table 7.1 Summary of the main contribution of the disseration. 99

LIST OF FIGURES

Figure 3.1 Regression based Performance Estimation Framework. 19
Figure 3.2 Generated Regression Models. 25
Figure 3.3 Prediction Error Distributions. 26
Figure 3.4 Measured vs Simulation vs Predicted performance. 28
Figure 4.1 Reuse distance of memory accesses in a memory trace. 33
Figure 4.2 Integration of the cache model into the SimSo's ETM. 36
Figure 4.3 A two-level cache hierarchy. 38
Figure 4.4 Executed cycles obtained from gem5 and SimSo. 39
Figure 4.5 The worst response time with and without contention consideration. 40
Figure 4.6 Worst Case Response Time for a subset of Γ tasks under different schedulers for a random cache. 41
Figure 4.7 System's Minimum Slack reached under different scheduling policies while varying the L1 cache size of core 1. 42
Figure 5.1 Analysis and scheduling of a simple avionic system. 47
Figure 5.2 The effect of splitting a cycle in a real-time system. 53
Figure 5.3 Direct and Indirect Error Propagation results in weighted graph $\Omega(t_9)$. 56
Figure 5.4 Example: CEP for three different nodes 57
Figure 5.5 Dataflow graph transformation of an avionic subsystem with components executing at a rate of $T = 10ms$. 60
Figure 5.6 System CEP for different node connectivity degrees 63
Figure 5.7 The directed graph of a voice-band data modem. 65
Figure 5.8 Schedule accuracy for the components in (a) subsystem Sub_1 and (b) subsystem Sub_2 68
Figure 6.1 Information propagation from and to X through passing messages of type λ between X and its neighbors $Y_i \in \mathcal{N}(X)$ and π between $\mathcal{N}(X)$ and X. 74

Figure 6.2 The accumulation of information conviction for the network in (A) for iterations (B) $t = 1$ (C) $t = 3$ (D) $t = 6$. We can see that as we observe new states, the conviction of certain nodes grow larger than others. This is observed to not be entirely dependent on the distance to the broadcasting source where some nodes in different regions of the network exhibit higher conviction than immediate neighbours. 75

Figure 6.3 PMDs of A and B in which the arrows represent every possible time when both A and B have the information. The arrow linking times t and $t+1$ defines the case in which A recieves the information at time t but fails to propogate to B which recieves it at time $t+1$ instead. 76

Figure 6.4 Progression of the rumour and the counter-rumours in terms of time which reveals two main attempts at correcting the false claim with differing reactions. 78

Figure 6.5 (a) Heat maps illustrating the probability of the information spreading from (1) Alice and (2) Bob starting at the bottom-left corners to their neighbors. (b) Heat maps for critical users in the intermediate neighborhood of User Alice that exhibit higher probabilitties of information propagation than Alice. 79

Figure 6.6 For the scale-free topology in (a), we show the interaction graph in (b), the time for the swarm of Kilobots to reach complete agreement (c) and its corresponding estimation of pWCCT (d). . . . 81

Figure 6.7 pWCCT for a random geometric network in its original form, and different selction methods as the number of selected elements m is varied to (a) $m = 10$, (b) $m = 25$, and (c) $m = 50$. 83

Figure 6.8 Example graph to demonstrate (b) sequential propagation of information from Λ to x that requires convolution of PMDs and (c) the less pessimistic structure of seeing x as part of a branch, akin to parallel process, which requires the merge operator. 88

Figure 6.9 Time for a swarm of Kilobots to reach complete agreement and its corresponding estimation of pWCCT for (a) a random topology with obstacles and (b) a snake topology. 93

LIST OF SYMBOLS AND ABBREVIATIONS

CDF	Cumulative Distribution Function
CEP	Cumulated Error Propagation
CPI	Cycle Per Instruction
DAG	Directed Acyclic Graph
DBT	Dynamic Binary Translation
DFG	Data-Flow Graph
DSE	Design Space Exploration
EPA	Error Propagation Approach
ETM	Execution Time Model
ETP	Execution Time Profile
FMS	Full Mission Simulator
IMA	Integrated Modular Avionics
ISA	Instruction Set Architecture
ISS	Instruction Set Simulator
LLC	Last Level Cache
LRU	Least Recently Used
MBPTA	Measurement Based Probabilistic Analysis
MFA	Minimum Feedback Arc set
MSE	Mean-Squared Error
NII	National Institute of Informatics
OEM	Original Equipment Manufacturer
PMD	Probability Mass Distribution
pWCET	Probabilistic Worst Case Execution Time
pWCCT	Probabilistic Worst Case Convergence Time
RMS	Rate Monotonic Scheduling
RTS	Rela-Time Systems
SDF	Synchronous Dataflow Graphs
SME	Subject Matter Experts
SMS	System Minimum Slack
SPTA	StaticProbabbilistic Timing Analysis
WCET	Worst Case Execution Time
WCRT	Worst Case Response Time
WSN	Wireless Sensor Network

LIST OF APPENDICES

Appendix A IDENTIFICATION OF RELEVANT SUBSPACES BASED ON LOCAL INTRINSIC DIMENSIONALITY 117

Appendix B RESOURCE OPTIMIZATION OF LARGE COMPLEX REAL-TIME SYSTEMS 118

Appendix C TIMING ANALYSIS WITH A PERMANENT FAULT DETECTION MECHANISM 120

CHAPTER 1 INTRODUCTION

This dissertation covers the research work pursued to fulfill the requirements of the Philosophiae Doctorate degree in software and computer engineering within the MIST laboratory from May 2014 to December 2018. This document follows the structure of a thesis by articles – covering 4 published and submitted works presented as separate chapters. This chapter introduces the topics we explored and places our work among the state of the art.

1.1 Context and Motivation

There exists a relationship of duality between living organisms and man-made systems. It is circular in nature and connects understanding biology and engineering artificial systems. Biological beings have, throughout history, been the inspiration behind some of the most ingenious artificial systems. The Shinkansen, also known as the bullet train, encountered major sound issues due to its high speed; from the connections to the overhead wires and as a result to speeding into tunnels that generated atmospheric pressure waves at the speed of sound producing large sound booms. Solutions to both these issues were observed in nature from the way owl feathers dampen noise during nocturnal flights and the manner in which kingfishers dive from the air into a denser fluid like water without creating any ripples. These led to a major reduction of noise and inspired the famous shape of the bullet train nose [1]. This tendency to search for answers in biology continues to be observed in today's technological progress in which a recent study of the Galago (also known as the Senegal Bushbaby) inspired the design of robots with vertical jumping agility [2] followed by another study of high-jumping beetles to engineer agile robots [3]. This was motivated by the observation that jumping, as opposed to crawling, is a more effective way for robots to avoid obstacles and a better option to have when a robot loses functionality of a leg [4].

The other face of this duality is that engineered systems are being more and more sought out to explain certain behaviors observed in biology. When it comes to understanding collective behavior especially, proposed approaches varied from proposing models and simulation of fish school movements [5, 6] to employing tiny robots to uncover the capabilities behind the emergence of certain behaviors observed in animals [7]. A study of scaling in computers [8] developed a model based on energy and time delays. They suggest that the proposed scaling theory could explain evolutionary transitions such as sociality observed in humans and animals alike and interpret the flow of energy and information through biological systems such as the scaling of energy consumption and economic growth in human populations [9]

and that of the flow of river landscapes that minimize the gravitational energy loss [10].

An idea that had been introduced decades ago [11] and is seeing a growth in popularity [12, 13] is that the unifying concept to build a coherent structure and grasp of the laws behind the existence and emergence of life and biology is *information*. It advocates modeling living organisms as systems that gather information in order to learn and evolve. Information in all its states, whether being processed, stored or transmitted, has been observed to lead in one way or another to the emergence of some sort of event [14]. One manifestation is the visual cues, in the form of a waggle dance, that are found to be at the core of the emergence of collective behaviors such as foraging in honeybee colonies [**?**]. Although the concept of information is abstract and can be viewed from different angles (information capacity, algorithmic information, etc.), it is still instantiated in physical degrees of freedom and ought to be considered in the study of life in the same manner that energy and matter are regarded.

"Information is physical!"
– Rolf Landauer,
Physics Today, 1991. [15]

Similarly, in engineered systems, information is deep-rooted in the emergence of new approaches and the advancement of the current technology. When artificial systems are concerned, information falls under the principal '*information-as-thing*' introduced by Buckland [**?**] to define information. Engineering disciplines tend to embrace this definition by looking at information as a stored quantity, like data, that can be processed and transmitted. The availability of this type of information becomes essential to build high-performing systems and ensure the proper execution of the desired behavior. There is however a general assumption that information is available when describing systems, predicting their behaviors, and improving their performance. A technique to optimize the allocation of resources, for instance, will define fitness functions built on performance metrics that are assumed to be known and ready to be used. Yet, when going from mere theory to actual implementations, especially industrial applications, designers encounter a gap between what is proposed in research and the information available. Namely, a complete lack of data or little to no specification that is inadequate to build relevant information necessary for the deployment of proposed solutions.

A different angle to information, that has long since become prevalent in the study of biological systems, is interpreting information as an emergent property. Binder et al. [12] argues that information should be viewed as a transferable quantity that is characterized by flow, without necessarily being conserved. Many have embraced this definition [13, 16, 17, 18] to establish understanding of behaviors and in doing so, developed ways to model the information flow in all kinds of networks.

That being said, whereas research in engineering fields recognize the importance of 'information-as-thing' in designing better systems, the gains from studying information flow in today's growing industry of connected artificial devices, is a topic yet to be tapped into and further explored.

1.2 Problem Statement

This dissertation presents research work that views information from two different angles and addresses the challenges that stem from each. It introduces approaches that bridge the gap between theory and practice when it comes to advancing knowledge and technological design. In particular, it mitigates and tackles the following issues:

- Lack of relevant data to describe, optimize, and integrate complex large systems;
- Fluctuation of relevance of information from one phase to another during large complex system design;
- Overlooking the significance of technology advances on traditional approaches, particularly timing behavior of complex real-time systems;
- Inefficient exploitation of information flow models in analyzing and engineering modern systems;
- The complexity of existing information propagation models that might not be feasible under the lack of enough descriptive information.
- Lack of studies exploring time of information propagation in a practical sense.

1.3 Research Objectives

The research work presented in Chapters 3 to 6 aspires to develop approaches and techniques to provide relevant information and in turn exploit that information in engineering new and better systems. A set of research objectives are outlined in the following to undertake the issues identified in the previous section:

1. Provide a methodology to translate raw data into relevant information for the design space exploration of complex real-time systems (Chapters 3, 5);
2. Develop scheduling simulation tools that consider accurate execution time estimations with an awareness to memory contention (Chapter 4);

3. Introduce a technique that exploits the flow of information to construct relevant system descriptions from scarce data for easing system integration and optimization (Chapter 5);

4. Model information propagation under minimal assumptions for the purpose of studying its ability to explain events and behaviors and its worth to modern systems (Chapter 6).

5. Extend the developed propagation model with a timing analysis from embedded systems to networks of interacting agents to probabilistically estimate information propagation times (Chapter 6);

6. Exploit the model for information spreading to propose a method to strengthen the performance of connected networks against faults (Chapter 6).

1.4 Contributions and Impact

To the extent of our knowledge, the contributions presented in the body of this dissertation and outlined here are original in nature and hope to advance both academic and technological knowledge:

- Frameworks to provide the right information for engineers to integrate large complex software systems that allows for the translation of available data and techniques into pertinent information ([19], [20]);

[19] I. Hafnaoui, R. Ayari, G. Nicolescu and G. Beltrame, "A simulation-based model generator for software performance estimation," in *Proceedings of the Summer Computer Simulation Conference*, ser. SCSC '16, 2016, pp. 20:1–20:8.

[20] I. Hafnaoui, R. Ayari, G. Nicolescu and G. Beltrame, "Scheduling real-time systems with cyclic dependence using data criticality," *Design Automation for Embedded Systems*, vol. 21, no. 2, pp. 117–136, 2017.

- An addition to scheduling toolkit of multi-core real-time systems that takes memory sharing overheads into account ([21]);

[21] I. Hafnaoui, C. Chen, R. Ayari, G. Nicolescu and G. Beltrame, "An analysis of random cache effects on real-time multi-core scheduling algorithms," in *Proceedings of the 28th International Symposium on Rapid System Prototyping.* ACM, 2017, pp. 64–70.

- Timing analysis techniques built on probabilistic analysis and probabilistic graphical models to establish valuable timing characteristics of single- and multi-agent systems ([21],[6]);
- Probabilistic representation of information propagation in biological and artificial systems with minimal number of assumptions ([6]).

[6] I. Hafnaoui, G. Nicolescu and G. Beltrame, "Time is of the essence: Spreading information among interacting groups, Scientific Reports, 2018. (*Submitted*)

The scientific significance of the proposed work and its impact on both biology and industry include:

- Strategies to provide relevant data and information for the purpose of bridging the gap between advances in academic research and the design of industrial systems;
- More robust connected networks and better propagation of communicated information among interacting individuals or devices.
- A better understanding of the importance of flow of information in describing events and behaviors in collective settings and the engineering of better systems.

With the growing realization that the dependence relationship between academic research and industry is inevitable, more effort is needed to transfer applied research to industry [22] which places the work presented here at the forefront to fulfill this need. At the same time, although biological systems had their share of works investigating the role of information in explaining their intricacies, the study of artificial systems in the same context is fairly young and the importance of information and how it should be handled is of utmost importance, especially with the growing interest in connected networks of communicating agents such as those seen in the Internet of Things (IoT) and robotic swarm fields.

1.5 Dissertation Organization

The dissertation follows the traditional structure of a thesis by article which dictates that the body of the contributed articles, published and submitted, be presented in separate chapters.

- Chapter 2 introduces the reader to the topics addressed throughout this dissertation and situates the work presented among the current literature;
- Chapters 3 to 6 mark the body of the dissertation and presents the published/submitted research articles:
  - Chapter 3 introduces a model generator for estimation of performance, presented as a conference paper at the 2016 Summer Simulation conference in Montréal, Canada [19];
  - Chapter 4 contributes a scheduling simulator tools that is cache contention-aware at the 2017 Rapid System Prototyping symposium as part of Embedded Systems Week held in Seoul, South Korea [21];
  - Chapter 5 is a published article at Design and Automation of Embedded System in 2017 [20] providing a methodology to eliminate cycles in the representation of complex systems for better integration and optimization;
  - Chapter 6 details the study of information spreading through a simple propagation model and its utility in providing understanding of biological behaviors and engineering of more robust systems. The work is submitted to Nature's Scientific Reports and is under major revision;
- Chapter 7 discusses the contributions of the previous chapters and highlights the relations among them;
- Chapter 8 wraps the dissertation with concluding remarks, and future endeavors;
- The document concludes with a number of appendices that gives a glimpse at the works achieved during the NII internship and summarizes a set of articles that have been co-authors and are strongly related to the work presented here.

CHAPTER 2 LITERATURE REVIEW

This chapter examines the fundamental understanding and existing research that addresses information from two different perspectives: as a stored quantity and as a property characterized by flow. It particularly extends on the current approaches to handling stored information and the current state of information flow research in artificial systems.

2.1 Information as Stored Data

Looking at information as a 'thing' is common in artificial systems and building approaches to predict and enhance their behavior expects access to this information, which is not always the case. An instance of this is the design of large complex systems, such as those found in the aerospace and automotive industries. A particular example is Full-mission simulators (FMS) which are large embedded systems that simulate the different parts of aircraft such as the Airbus A380. The conception cycle of these systems can be abstracted into three phases: (1) development, (2) integration and (3) deployment. The first phase involves the initial design of the different parts of the systems, referred here as modules. Subject matter experts (SMEs) develop their modules on their local workspaces independently of the final platform. The integration phase ensures that the different modules are put together in a way that guarantees the well-functioning of the whole system according to a set of specifications. Since the addressed systems are time-critical, which enforce hard deadlines to the execution of the modules as well, the integration makes sure the timing behavior of the system respects certain timing constraints. The last phase deploys the integrated system on a final hardware platform.

This research focuses on the problems encountered at integration phase due to missing relevant information about the system modules produced in the development phase. The complexity and sheer scale of these systems requires the contribution of knowledge and expertise from different fields. This asks SMEs to have expertise in their respective fields and be able to provide information about the software performance of their modules. As a result, the developed modules are delivered with their own set of data with different degrees of accuracy and relevance that need to be processed in a way to make them applicable for system integration.

The timing behavior is of particular importance, and integration specialists must ensure that the system is schedulable, i.e. all its modules can be executed in an appropriate time frame.

Schedulability analysis verifies that the execution of the system modules do not exceed a specified deadline according to a scheduling policy. Failing to meet these deadlines could incur damages to the system and result in catastrophic events. Past research produced a plethora of schedulability tests - that mathematically reveal the potential of a set of tasks to miss their deadlines in a given scenario [23, 24] and scheduling simulators [25, 26, 27] capable of simulating schedulers for large systems and expose corner cases. Doing so avoids failures such as the priority inversion issue encountered in the Mars Pathfinder probe [28] in which a low priority tasks indirectly preempts the execution of a higher priority task by locking access to shared resources. That being said, these approaches are typically built on the availability of certain information such as the execution times of software. Estimating the timing performance of modules is dependent on the hardware platform on which they are executed and should be considered when providing these data. This lack of relevant information closes doors to the use of available techniques and tools that have the potential to improve the system's performance such as the optimization of resources and incorporating different and better existing solutions.

2.1.1 Describing and Integrating Complex Systems

A frequently established assumption in literature is that these complex systems can be modeled through simple graphical models that can clearly distinguish the relationships linking the different parts, especially their dependencies. In reality, due to the complexity of these systems, software modules are often made of either reused code or resort to the use of original equipment manufacturer (OEM)'s products presented in black box format. This lack of transparency often conceals relevant information that might help in discerning dependencies, and hence establish a proper integration of the system. This leaves engineers with minimal information, enough to build a raw representation of the system in the form of a dataflow graph (DFG). In large-scale systems, modules are interconnected in a way that naturally creates cycles within the graph representation, which can represent a problem during integration. In fact, when timing behavior is concerned, Directed Acyclic Graphs (DAG) are widely used representation to define precedence requirements between modules. The lack of cycles eases the assignment of the module execution order in a way that doesn't affect the functionality of the whole system. This doesn't mean that scheduling systems without a DAG representation is impossible: many have proposed ways to schedule the system under a graph representation that contains cycles by developing techniques to by-pass the cycles either by considering the particular functionality of the nodes [29] or by grouping the cycles as special nodes [30]. Once again, these approaches fall outside the scope of our inquiry since they require the knowledge of certain parameters that are generally not available. The same

can be said for synchronous dataflow graphs which assume the delay of information processing is available [31, 32]. Aside from that, techniques based on DAG representations tend to be less complicated and numerous in comparison which offers more desirable and flexible options for designers at integration time. Chapter 5, in that regard, assumes that little data is available to describe the system and tackles the issue of obtaining a DAG representation from a DFG.

At integration, certain metrics are required to ensure that the system will behave properly after deployment. These metrics, such as the worst execution time, memory consumption, memory overheads, are considered critical information that might not be available. As discussed previously, the fact that multiple expertise is required to build such systems creates a gap in the type of knowledge and hence data that is provided by SMEs. These metrics are generally dependent on the hardware architecture on which the software is deployed and can theoretically be provided by the SMEs thanks to multiple profiling tools. However, more often than not, the end hardware architecture is not accessible to SMEs or utterly unknown at design time. In some cases, optimization techniques are employed to decide the final hardware platform, which in fact requires these type of metrics, creating a circular dependency dilemma. Many have proposed the use of hardware simulators to estimate performance metrics with varying degrees of accuracy and speed [33, 34, 25]. However, the use of simulators becomes prohibitive when the number of configurations to be evaluated grows too large as discussed in Chapter 3. Especially when exploring the large design space of embedded systems, simulating potential configurations could takes days, even weeks, to reach a pseudo-optimum solution. The use of analytical models have been proposed as an alternative. Regression models, in particular, have been used to predict the performance of code on supercomputers [35], GPU-based systems [36] and heterogeneous MapReduce cluster [37]. This last work is similar to what is presented in Chapter 3 but takes it a step further by proposing an approach that merges simulation and regression to estimate performance independently of the purpose and functionality of the software module in a timely manner.

2.1.2 Timing Analysis of Real-time Systems

The integration of these kind of complex systems devotes particular effort to analyzing the timing of the system. Working with the right information is then crucial to ensure the system parts execute properly and respect their timing specifications.

Traditionally, estimates of execution times of software were severely conservative due to the reliance on deterministic architectures which place the Worst Case Execution Time (WCET) a great distance away from the actual maximum time reached by the execution of a module.

This pessimism led designers to investigate probabilistic architectures. The use of these architectures results in the avoidance of pathological cases and decreasing the pessimism of the WCET estimation [38]. One approach to estimating a probabilistic WCET is Static Probabilistic Timing Analysis (SPTA). Altmeyer and Davis's work [39, 40, 41, 42] provided progress in this field by exploiting the concept of reuse distance - the number of accesses between two consecutive calls to the same memory address - to provide running estimates for programs.

At the same time, multi-core systems have slowly but surely integrated a wide spectrum of disciplines, including the ones being addressed here. The exploration of multi-core systems, with real-time constraints especially, is still a fairly young field since predictability and ease of integration are widely desirable with these systems. Providing the right information and developing the appropriate tools is key. Research has produced schedulers to specifically address applications executing on multiple cores and a lot of effort has been put forth to evaluate these policies either through mathematical models or simulations.

The majority of proposed schedulers however are based on the assumption that sharing memory space among applications running on different cores (inter-core interference) or within privately shared caches (intra-core interference) is negligible and doesn't affect the execution time estimates. A study by Mars et al. [43] disagrees and shows that a performance degradation of 35% is observed when sharing the Last Level Cache space among four cores, a degradation that current scheduling simulators disregard altogether when used to design highly critical systems.

This is not to say that literature lacks in terms of works that model cache sharing. In fact, Ding et al. [44] nicely presents the advances in this topic over the past decades and emphasizes the role of reuse distance as a metric to reach more accurate cache models as opposed to others such as stack distance. Pan and Jonsson [45] introduced reuse distance to Markov chains to predict shared cache performance implementing different replacement policies such as random, Least Recently Used (LRU) and pseudo-LRU. More recently, a common framework that accurately models probabilistic caches based on absolute reuse distance was proposed by Beckmann et al. [46].

Since simulators are widely used by large system designers to test their configurations for schedulability and validating new scheduling policies, shared cache overhead should be accounted for. Yet most current scheduling simulators overlook this effect. A first attempt at mitigating this shortcoming was introduced by Cheramy et Hladik [47], who extended the SimSo simulator [27] with an LRU cache model that accounted for the overhead incurred by sharing cache space among the cores. No effort has been put forth to do the same for

random caches despite their growing popularity and that partly due to the fact that the models proposed so far are too complex and computationally heavy to be incorporated into existing simulators. The work in Chapter 4 extends the work proposed by Eklov et al. [48] to bridge this gap by proposing a cache model compact enough to be integrable in a scheduling simulator.

2.2 Information as an Emergent Property

Seeing as information is important in developing artificial systems, and keeping to the trend of falling back on biology to inspire new technological advances, a different interpretation of information is found. An emergent property refers to properties that are not particular to any part of a system, such as a single individual or entity, but emerge at larger scales. In other words, it is a characteristic that an entity gains as it becomes part of a larger system. Information can be viewed as a property that is not particular to one individual but one that, through flow, arises as a characteristic of the group. Information, in that sense, seems to take on a 'life of its own', and has been described as an emergent property [49]. In their endeavor to establish the origin of life and laws that govern biology, Davies and Walker [13] assert information as the key to entangle the perceived complexity of biological systems. Equally, with the growth of connected devices and artificial systems emulating human and animal behaviors, this concept of information is becoming more relevant to modern systems as well, which potential is yet to be tapped into properly.

2.2.1 Information Flow

The common approaches to the study of behaviors have often relied on experimental observations to prove or disprove a hypothesis and finding correlations between certain features and the emerging behavior. In studying human collective behaviors, for instance, de Vreeze and Matschke [50] emphasized, through empirical observation, the importance of taking the relationship of individuals in a group in the information exchange, especially in decision-making scenarios. The shifting in political adherences was attributed to social information in the form of popularity of certain information through an experimental study of petition signing by Hale et al. [51].

Others, however, pursued the idea that there is more to be learned by understanding how information spreads and relied instead on mathematical models and simulations of information propagation to uncover the elements behind emerging behaviors. Strandburg-Peshkin et al. [52] experimented with the idea that basing the interaction of individuals on spatial

queues such as a metric range or a fixed number of closest neighbors might not be enough to model information propagation and explain the emergence of schooling in gold shiner fish. They establish that the visual field is a better metric to explain the observed behaviors. Information propagation as visual cues in flocks of birds have been modelled by Brush et al. [53] to study the responsiveness of the flock to different types of information, such as food source or predators, and how the content of the information affects the behavior of the flock in deciding their actions. Models of infection spreading, on the microscopic level, within cells, and on a larger scale, within communities, where infection can be abstracted as a piece of information being transmitted between individuals, have also been introduced. The timing of the propagation especially was given a particular interest [17, 54, 55].

Although these works were proposed to study different systems under various and specific circumstances, the notable feature that is prevalent is the complexity of the information flow models. The infection models proposed in [56, 57, 17] are built on the assumption that the infection rate, the state of the individual, among others, are known. The work proposed in [18] goes as far as to require a pedestrian model to accurately estimate the infection transmission in air travel. The complexity introduced in these models might seem in accordance with the intricate systems under scrutiny, which might be the case, but the question that arises from the notion that simplicity might be a latent property of these systems is whether certain behaviors can be studied with simpler models of information.

This has been addressed in part in the context of infection spreading by Ottino et al. [58] whose purpose was to establish a take-over time through a simple model with the assumption that infection spreads to one node at a time from any infected node in the system. In an effort to offer a more practical solution, Chapter 6 introduces a probabilistic timing analysis, built on the timing framework introduced in Chapter 4, and on a simple but slightly different propagation model that assumes different affinities for the information propagation from one individual to its neighbors.

The fact that relevant data to describe the system are more likely to be inaccessible or difficult to acquire for practical applications gives significance to pursuing simpler models for the purpose of unraveling the emergence of certain behaviors.

2.2.2 From information flow to Breaking the status-quo

Information models, in the context of collective behaviors at least, have been wielded in a rather passive manner in which the sole purpose of developing such models was to unearth the main contributing aspects to emerging behaviors. In hindsight, the study of information flow has seen little progress to engineering modern systems as opposed to the common trend

of transferring knowledge acquired from studying biological systems to artificial ones. One venture saw the authors of [59] establish a framework to identify the root of causal anomalies in large scale complex systems by modeling the fault propagation in invariant networks. The idea of obtaining relevant information by following the flow of injected information in the network has also been exploited in the work presented in Chapter 5 to identify critical data.

Identifying major role players in a given scenario by studying the flow of information has also received a lot of attention. The work of Sankar et al. [60] developed a technique, inspired by the way bees spread relevant information during foraging, to identify the most influential users in a social network. The emergence of leadership was instead examined in [61] through a study of consensus convergence in heterogeneous groups. For all that, these works were still conceivably targeted at biological systems, namely human communities, despite the information propagating through artificial means. Chapter 6 exploits the simple model to introduce an approach to identify the weakest elements in the network that improve the overall performance of the system.

When it comes to designing systems such as robot swarms or connected Internet of Things (IoT) networks, the likes of mobile wireless sensor networks, great work has been done in proposing robust policies to communicate relevant information among the agents. However, if the timing performance of the system is not available, it becomes difficult to achieve certain behaviors. A simple example would be a swarm of robots autonomously agreeing on an allocation of a task set. In engineering time-critical systems, one essential requirement would be that the time to agree on an allocation does not exceed a certain deadline in which case having an estimate of the time for the information to propagate to all the robots becomes crucial. As presented previously, Ottino et al. [58] addressed this issue in the context of infection propagation. The proposed model might shed light into the kind of distributions that shape the takeover times as established by the authors, but it makes it difficult to transfer that knowledge to practical settings. The same can be said about the approaches proposed in [62, 55] which relied on Markov chains to model information propagation which use can become prohibitive as systems grow in size, which is more likely the case with modern systems. The work detailed in Chapter 6 provides a probabilistic estimate of the worst time for the information to propagate which has more of a flexible and practical use especially for designing hard real-time systems.

CHAPTER 3 ARTICLE 1 – A SIMULATION-BASED MODEL GENERATOR FOR SOFTWARE PERFORMANCE ESTIMATION

Preface

The first step towards accomplishing the outlined research objectives is addressing the gap created by the different expertise required to design large complex software systems such as those encountered in the automotive and avionics systems. We approach the problem of lack of software specifications necessary to the integration of these systems by relying on both simulation and machine learning techniques to provide estimates for software performance without having knowledge of the software functionality or necessarily access to the final hardware platform. As a matter of fact, we establish this framework in order to enable the exploration of design space of more appropriate hardware architecture in reasonable time frames.

Full Citation I. Hafnaoui, R. Ayari, G. Nicolescu and G. Beltrame, "A simulation-based model generator for software performance estimation," in *Proceedings of the Summer Computer Simulation Conference*, ser. SCSC '16, 2016, pp. 20:1–20:8.
URL: http://dl.acm.org/citation.cfm?id=3015574.3015594

ABSTRACT

With the rise of software system complexity, developers rely more on a modular approach to system design to reduce development cost. However, as a result, integrating a real-time system becomes a challenge. To be able to properly integrate the system, software developers are required to provide software characteristics such as the execution times of their components to ensure the correct timing behaviour of the overall system. Generally, engineers rely on profilers available on their workstations to collect execution times of software. Yet, the final target architecture is usually vastly different from that of the workstation. Further, the fact that the target platform is mostly inaccessible at design time calls for tools that can estimate the execution time of components on a wide range of architectures with reasonable cost. In this paper, we propose a methodology that relies on (1) fast simulation techniques and (2) analytical tools that build predictive models to estimate the execution times of components

on a target architecture with minimum detail. We show that the approach is able to predict the execution times of a set of benchmarks when migrated from reference architecture to a target platform with upto 8% improvement in prediction error when compared to simulation.

3.1 Introduction

The automotive and aviation industries deal with the development of very complex real-time software systems. Full Mission Simulators (FMS) are an example of such systems. The development of FMSs require the expertise of subject matter experts (SMEs) from a multitude of engineering fields. Yet, these SMEs have varying degrees of knowledge when it comes to software engineering and embedded systems. The new modular approach to software development allows the SMEs to work on their individual system components in their separate environments which allows them to test the functionality of their code independently from other components. This also opens the possibility to reuse code from different systems built for other purposes or that developed for the goal to be reused. Integration of the system refers to putting the system components together in such a way that they are able to operate collectively as a whole system in a coordinated manner. At this level, making sure that the individual components function properly is not enough. An integration expert has the important job of defining a software/hardware configuration that will ensure the correct functional and timing behaviour of the whole system on a particular hardware platform while optimizing a set of performance objectives. With complex systems such as FMSs, where the number of components could grow large, integration experts find themselves in a dilemma. They are faced with a number of configuration choices but a lack in data that describe the different components makes it hard to reach a decision on the final configuration. SMEs are able to provide some of these data such as execution times, memory consumption, etc. However, these data are collected at their working stations. The fact that the final target hardware architecture that will bear the deployed system has different architectural details makes the collected data irrelevant. When dealing with real-time systems, one of the data crucial to ensuring that the timing requirements are met at the integration phase is the execution time of the system components. Many approaches to estimate the execution time of software has been proposed. The work in [33, 34, 25] introduced simulators that replicate the behaviour of code on a predefined architecture with differing degrees of accuracy and at different levels of abstraction. That being said, relying solely on simulation to estimate execution times of a large number of applications without losing in accuracy becomes prohibitive even with the fastest simulators to date. Furthermore, simulators become impractical solutions when Design Space Exploration (DSE) is concerned since the design space is too large to be effi-

ciently explored in a sensible time range. In this work, we propose a model generator that builds analytical models describing the connection between two hardware architectures. The approach trades off accuracy of estimation with the speed of analysis to predict for execution time of system components on a given hardware architecture. To do so, our methodology relies on two key elements:

- A regression model that represents the relationship between a reference and a target architectures and predicts for the execution time of any software component when run on the target architecture.
- A fast and accurate hardware simulator that is used to construct a dataset of training examples from a set of benchmark suites. The suites used to build the database are known to cover the different types of workloads existing in industry.

Our approach takes advantage of simulation accuracy to build an accurate database. A regression model is trained and fitted using this database. The model provides fast performance estimates which allows for large design exploration without incurring in simulation overhead. While the scenario that we consider in this paper is that of predicting the performance of system components when migrated from a reference to a target architecture, our approach can be applied to related issues such as providing data to explore the design space for architectural choices that fit a set of performance objectives defined by a user; like for instance, optimizing the overall response time of the system or minimizing communication traffic on a network, etc. The remainder of the paper follows this structure: Section 3.2 provides a summary of the works related to this issue; our approach is detailed in Section 3.3 where we describe how the concepts of regression analysis are moulded to fit our problem, as well as the tools that are used to achieve a good performance estimation. This is followed by a set of experiments and discussion in Section 6.6.5; finally, we wrap up in Section 3.5.

3.2 Related Work

Throughout the years, designers have taken different roads to estimate the performance of a particular application on a given hardware platform. Some relied on architecture simulators to replicate the behaviour of software on a predefined architecture. Simulation techniques can vary in accuracy and complexity depending on the level of abstraction. Instruction Set Simulators for example offer high accuracy but are generally very slow since they reproduce the smallest detail of an architecture. To state a few of these types of simulators, Gem5 [33] is a full system simulator that supports multiple ISAs. [25] introduced a x86 architecture

simulator under the name MARSS. TSIM was proposed by [34] to simulate for the open source LEON processor. Simulators based on native execution were proposed to mitigate the issue with ISSs. Although they offer lower accuracies when it comes to performance estimation, they are relatively faster and are based on code instrumentation. The works in [63, 64, 65] proposed a simulation technique based on instrumentation of source code. The code at this level is at its highest abstract form and is architecture-independent which makes these techniques faster. Back annotating code at the intermediate representation level with timing details is discussed in [66, 67] in which the iSciSim simulator was based on. The authors of [68] were the first to propose instrumentation of binary code which offers accuracies closer to that of an ISS with faster timing ranges, followed by many others [69, 70].

While these approaches to performance estimation gave good accuracies, the significant simulation overhead prohibited designers from either estimating for a large number of applications or exploring the large design choices to decide on a fitting final architecture. For such, some work has been proposed that was based on analytical models. [71] took advantage of the fact that different intermediate representations of code can be produced by the GCC compiler. They used the GIMBLE representation, which is closer to binary code, and RTL code to build linear regression models to estimate performance of code with zero knowledge of the target architectural details. Their approach was validated through experiments considering different levels of compiler optimization. Regression has been employed in [36, 72, 35, 73, 74] for performance estimation. Regression analysis can accelerate the design space exploration by building predictive models that estimate for software performance as well as provide an understanding of the relationship between system parameters and their effect on the final performance. [35] built a regression model to predict the performance of applications on superscalar computers. They used a set of 26 of features describing the architecture and a cycle-by-cycle superscalar simulator to achieve this. Stargazer was proposed by [36], a stepwise regression-based design to explore design space of GPU-based systems. Similarly, the work in [37] extended Linear Regression analysis to estimate performance of heterogeneous MapReduce clusters. They built a regression model that migrates the performance from a host architecture to a target architecture by having the regression go through a learning step on a microbenchmark suite. This is similar to our approach in that it also uses regression models to migrate from one architecture to a newer one, in this case a MapReduce clusters. However, it differs from ours in that it considers specific information about the application and divides it into sub-units based on their functionality to estimate performance. Our approach does not depend on the awareness of the details of the software being studied but regards it as one block of code. In this work, we want to see how accurately the regression model can estimate the performance of applications with minimum knowledge of the

characteristics of the software.

3.3 Proposed Approach

To help in the proper integration of complex systems, we propose a framework that exploits machine learning concepts such as regression analysis to estimate the performance of system components on a given hardware architecture. In the context of our work, regression is used to build a model that will predict performance metrics related to an application given a particular architecture. Before delving into the specific of our methodology, as described in Figure 3.1, we give a background overview of what regression is and the ways it is employed.

3.3.1 Linear Regression Models

Regression models are used to estimate the relationship among variables in a certain system. In better terms, one of the uses of regression techniques is to predict how one variable, the dependent variable, varies as one or more independent variables, called features, change values. Our focus is on Linear regression which builds a model that represents a dependent variable Y as a linear combination of a set of parameters α_i, called partial regression weights, linking the independent variables X_i. Mathematically, the dependency between the variable that we want to predict, Y , and the independent variables, X_i , using linear regression can be represented with this formula,

$$Y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + ... + \theta_n X_n + \epsilon \tag{3.1}$$

with n Number of features When the model is built using a single independent variable X_1, the model is called a simple linear regression model. This is the simplest type of models which makes it the easiest to manipulate.

Model Performance Metrics

As for every technique that is based on a training step, there is a need for performance metrics to judge and tune the trained model. When it comes to regression, to be able to investigate the generated models, we look into the residuals as defined by:

$$e_i = y_i - \hat{y}_i \tag{3.2}$$

where y_i represents the true value and $\hat{y}_i$ the value predicted by the model.

The final model of prediction is decided by looking into the model's goodness-to-fit measures. Here, we present a few widely used measures:

- **Mean-squared error (MSE)**: Most linear regression techniques try to minimize this value when building the model. It represents the variance in the residuals and can be computed as follow:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.3}$$

- Coefficient of Determination R2: represents the estimate of how well the model fits the observed data. It represents the predictable portion of the dependent variable from the independent variables. It means that R2 percent of Y is predictable from X. The closer to 1, the better data are fitted.

$$R^2 = 1 - \frac{SSR}{SST} \tag{3.4}$$

where:

$SSR = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

$SST = \sum_{i=1}^{n}(y_i - \bar{y}_i)^2$

with:

$\hat{y}_i$: arithmetic mean of Y

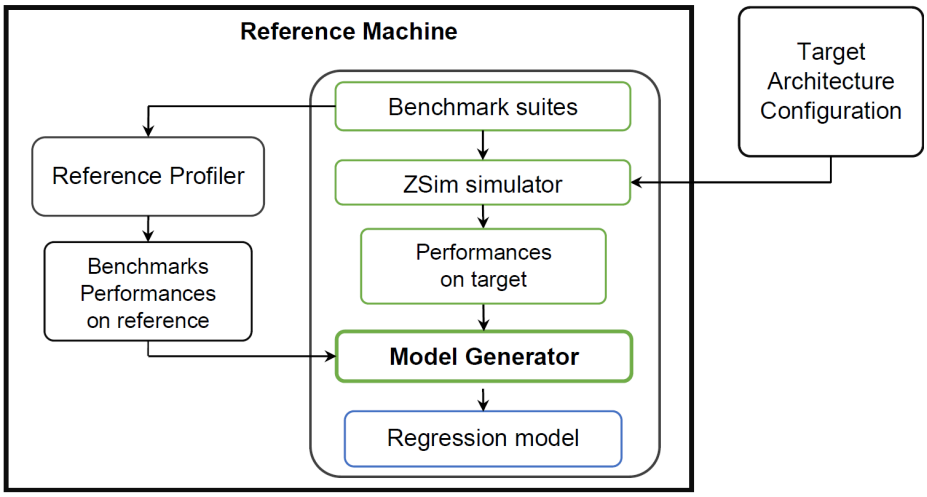


Figure 3.1: Regression based Performance Estimation Framework.

SSR: Residual sum of squares

SST: Total sum of squares

3.3.2 Model Generator

Integrating the system components involves ensuring that the timing requirements of the real-time system are met as well as optimizing the system resources according to a set of performance objectives. However, the lack of data that describe the system components, especially their execution times, creates a gap between component development and system integration. Our approach detailed here is proposed to bridge this gap; which will be key in decreasing the time-to-market and optimize the system resources. The schema in Figure 3.1 represents the approach. From this, we define some terms that will be used to describe the methodology:

- Reference Architecture: From the scenario studied in this paper, this represents a description of the workstation architecture in which the different components of the system are built. Since the developed code for the components differs in functionality and purpose, so does the architectural detail of the reference machines. However, this definition can cover any machine that will be used as a reference in building the model.
- Target Architecture: This describes the final hardware architecture at which the system components will be integrated to form the complete system. Considering the size and complexity of the integrated system, the target architecture generally involves more powerful features, such as the number of cores, clock speed, cache size and latency, etc.

Since the reference machines, on which developers build their components, have different architectural details to the target architecture, the performances computed at these machine, such as execution times, are not going to be of help to integrate the system and to efficiently allocate the components to the final target hardware platform.

Our venture into solving the performance estimation problem is to propose a model generator based on regression as depicted in Figure 3.1. This model generator builds a regression model that describes the relationship between a reference and a target architectures. This approach predicts for the speed-up expected when the individual components are executed on the target architecture. The proposed model generator has two main sections: a database builder and a regression-based tool to build a predictive model.

Database builder

Regression analysis is a branch of machine learning that goes through a learning step to build a model that fits a set of data points. For this purpose, there is a need for a training database that reflects the aspects that we want to predict. In our case, the training examples represent the execution time of a set of applications on a particular reference machine and the corresponding execution times of the same applications on a target hardware architecture. Application refers to any piece of software that can be executed on a hardware platform to which we want to predict the execution time. It could be one single method or, as in our case, a component that consists of a bundle of interconnected methods and classes. To be able to build this database, we require two elements:

1. Benchmark Suites: In order to obtain a regression model that will be able to predict the execution time of any type of application, the benchmark set needs to be diverse enough to cover most of the software design space. Diversity refers to aspects such as the application being computationor communication-intensive or if it is databased (e.g. matrix multiplication) versus instruction-based versus control-based. In addition, the applications can be developed for different domains. For a well-fitted model, regression also requires a large set of observations. In order to take these aspects into account, we build our database using three widely used benchmark suites:

   (a) Malardalen [75] A benchmark suite that was proposed for studies that involved WCET analysis tools and methods. It offers a set of small single-path programs that cover the most basic computational loads such as compression algorithm, binary search, Fast Fourier transform, filters like the Finite Impulse Response, etc.

   (b) MiBench [76] This benchmark covers applications from different fields such as automotive, industrial control, network, security, etc. It offers a range of programs for embedded systems that fall in the data and control-intensive categories. The embedded programs have an instruction profile that varies in the number of branches, memory and integer ALU operations which adds to the diversity of the suite.

   (c) PARSEC [77] Although this suite offers the option to test sequential code, it focuses hugely on parallelized code where multi-threading is a prominent feature in the benchmarks. The suite is also diverse as it provides workloads from different fields. A particular effort was put into making sure that the offered benchmarks are not skewed towards the High-Performance Computing domain. A subset of

emerging workloads that are assumed to be typical in the near future is included as well.

2. Architecture Simulator: To generate a sizeable set that represents the architectural details of an executing platform, we opt for the use of an architecture simulator based on instrumentation that can give accuracies close to that of a cycle-accurate simulator. ZSim [78] is a parallel multicore simulator that breaks the trade-off between speed and accuracy to simulate detailed architectural aspects. It employs Dynamic Binary Translation (DBT) to speed up sequential simulation during program instrumentation. As a result, it can simulate features such as branch prediction, limited fetch and issue width with less overhead. Since it works at the binary level, the accuracy of the generated profile is close to that of an ISS. When comparing simulated IPCs, ZSim has been reported to give absolute performance errors within 26% of the real system for 29 of the tested benchmarks on a Xeon processor. It was also compared in terms of speed of simulation and was noted that ZSim is $2-3$ orders if magnitude faster than other simulators such as Sniper and MARSS. It fits well with our purposes since it can model heterogeneous systems with arbitrarily configured memory hierarchies, and can run a wide range of x86 workloads.

Having defined the tools and notions that our approach relies on, we detail here the steps illustrated in Figure 3.1:

- Step1. The first step is to build the database. The benchmarks from the suite mentioned here are profiled on a reference machine to collect their execution times. This could be achieved either by (1) using the local profiler on the reference machine or (2) by making use of the architecture simulator to simulate the reference architecture.

- Step2. A second performance profile is gathered containing the execution times of the same set of benchmarks on a target hardware platform. ZSim is given the architectural details of the target architecture at its input. Details such number of cores, memory sizes, etc. are described in a file that ZSim will employ to simulate the behaviour of the benchmarks on that target machine.

- Step3. Concepts of simple linear regression analysis are extended in our model generator by re-writing equation1 as follows:

$$Y = \theta_0 + \theta_1 X_1$$

X_1 is a feature set that represents the execution time of applications computed at the reference machine (Step1). The model generator builds a model to predict the execution times at the target machine Y (Step2). The obtained observations by the database builder are divided into two subsets; a training and validation subsets. The core feature of the model generator is its ability to build a link between a reference and target architecture in such a way that any future new application can be given an estimated performance with zero simulation overhead.

3.4 Performance Evaluation

3.4.1 Experimental Setup

In our approach, the regression model is trained on execution time observations generated from running varying types of benchmarks on multiple architecture configuration. Following our motivating scenario, in these experiments, we predict the execution time of code when it is migrated from two reference machines to the same target architecture. The configurations of the reference and target architectures are summarized in Table 3.1.

The database generated for the purpose of the following experiments contains execution times of 80 benchmarks at the reference and target machines. To keep close to the scenario we are studying, the execution times of applications run on the target architecture are collected

Table 3.1: Reference and Target Characteristics.

Characteristics	Architecture 1	Architecture 2	Architecture 3
Processor	AMD A6-5350M	Intel Quad-Q6700	Intel i7-3770
Frequency	2.9 GHz	2.66 GHz	3.4 GHz
Caches			
L1i	64 Kb	32 Kb	32 Kb
	4-ways	8-ways	8-ways
	Latency 3 cycles	Latency 5 cycles	Latency 4 cycles
L1d	16 Kb	32 Kb 32 Kb	
	2-ways	8-ways	8-ways
	Latency 3 cycles	Latency 5 cycles	Latency 4 cycles
L2	1 Mb	6 Mb	256 Kb
	16-ways	24-ways	8-ways
	Latency 12 cycles	Latency 16 cycles	Latency 12 cycles
L3	\	\	8 Mb
			Latency 30 cycles
Replacement	LRU	LRU	LRU

from executing ZSim with the details of the target machine as in described in Table 3.1 and collecting the resulting estimated profiles of the benchmarks. On the other hand, two different sets of features are collected for the performances of applications on the reference machines. One set of features is measured using the local profiler of the reference machine which represent the real performance of code on that architecture. The other set is collected through the ZSim simulator in the same way the target performances were collected.

To validate the generated model, we need to ensure that the process of model building is isolated from that of future predictions. Cross-validation is a technique to validate a model based on statistical analysis which tests how the results of the model predictions generalize over an independent set of data, as well as the accuracy of the predictions. In other words, it ensures that the estimation of performance is not dependent on the choice of a particular training data.

We employ a k-fold cross-validation with a k = 10. This cross-validation technique will divide the database into k-folds; 1-fold, the testing dataset, will be left to validate the performance and accuracy of the built model while the rest, the training set, will be used to train the regression model.

3.4.2 Experimental results

We based our experiments on two scenarios of architecture migration as stated in Table 3.1 in which the benchmarks are migrated from two different reference architectures (Architecture 1 and Architecture 2) to a target architecture (Architecture 3). We introduce a metric Prediction Error that expresses how close the prediction are to the real values of the execution times, where

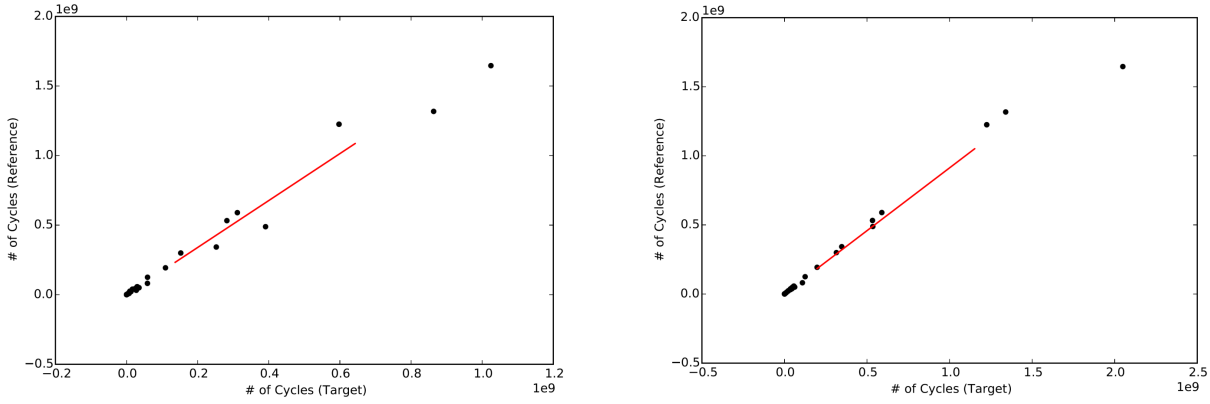
$$Pred_{Error} = \frac{Execution_{Pred} - Execution_{Real}}{Execution_{Real}}$$

Figure 3.2 represents the model generated by the regression to fit a set of execution times gathered at the reference and target machines. The regression tool used the real execution times of the benchmarks gathered by a reference profiler to build Model-1 generated in Figure 3.2a. Whereas Model-2 in Figure 3.2b used the data generated by ZSim when running the same benchmarks. As can be seen, the linear regression model is better fitted to the data points with the execution times collected through simulation from the ZSim as compared to the ones measured through a profiler. This is seconded by the R-Squared values depicted in Table 3.2 and Table 3.3 where the R-Squared value for Model-1 in Figure 3.2ais closer to 1 than that of Model-2.

This can be explained by the deterministic nature of the simulator ZSim in which Model-2 was generated by training the regression tool on the data extracted from running the benchmark on ZSim with the architectural details of the target architecture. To further verify the accuracy of the model resulted by our model generator, we plotted the real values of the execution time against that predicted through simulation and that resulted from the regression model.

It is good to mention that the accuracy of ZSim, as reported in previous sections, was validated by the authors against a few processor examples only, such as Intel's Xeon. As a matter of fact, the authors advice to not fully trust the results of ZSim with workloads and architectures different from their case study [79]. This information becomes relevant in the following section in which the performance of ZSim is observed to give a prediction error higher than 45%, as can be seen in Figure 3.4.

We observe that the models (H1TM and H2TM) of Figures 3.4a and Figure 3.4c give slightly better predictions than the models (H1TS and H2TS) shown in Figures 3.4b and Figure 3.4d . This is due to the fact that models H1TM and H2TM were generated using measured data at the reference architecture as opposed to models H1TS and H2TS which were generated using solely simulated data. This is due to the fact that the simulator is injecting an error since its performance predictions have low accuracies. As a matter of fact, since we employ the simulator to generate the predictions at the target architecture, the model generator is generating models that will try and fit to the data generated by ZSim. It can be seen in Figure 3.4 that the regression predictions are closer to the simulator estimations than the real measurements.



(a) Model-1: Performance Estimation Model from Measured Data.

(b) Model-2: Performance Estimation Model from Simulated Data.

Figure 3.2: Generated Regression Models.

Table 3.2: Performance of Regression Model Built on Measured Data.

Configuration		R^2	Prediction Error (%)			Improv.(%)
Ref.	Target		Mean	Std	Max	
Arch 1	Arch 3	0.768	13.7	21.3	43.8	8.29
Arch 2	Arch 3	0.912	2.99	20.6	37.4	0.22

Table 3.3: Performance of Regression Model Built on Simulated Data.

Configuration		R^2	Prediction Error (%)			Improv.(%)
Ref.	Target		Mean	Std	Max	
Arch 1	Arch 3	0.961	9.27	5.75	19.2	4.39
Arch 2	Arch 3	0.946	4.59	11.9	11.2	2.04

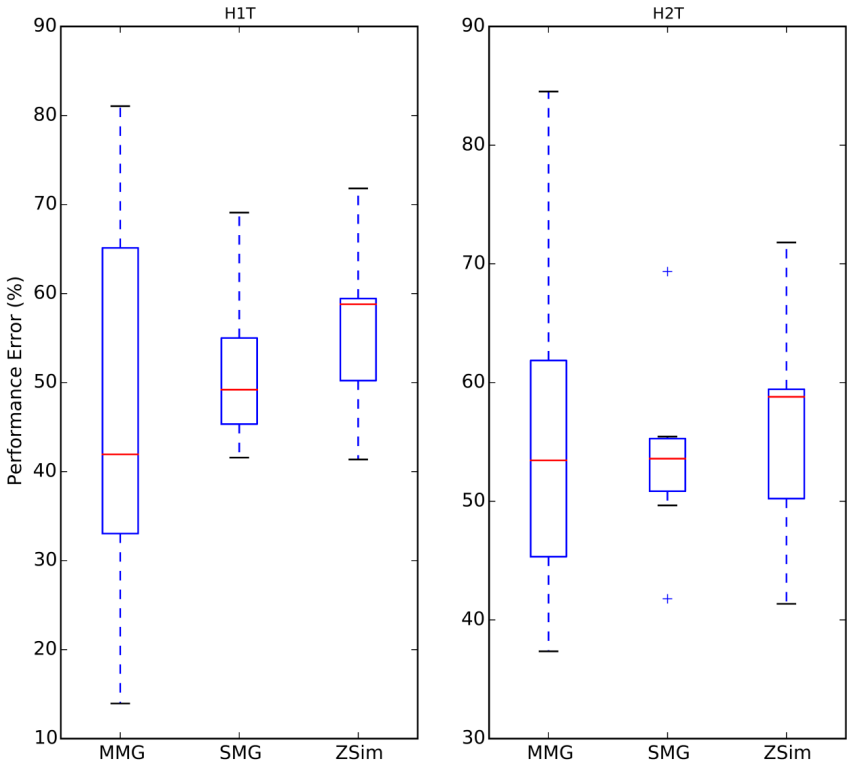
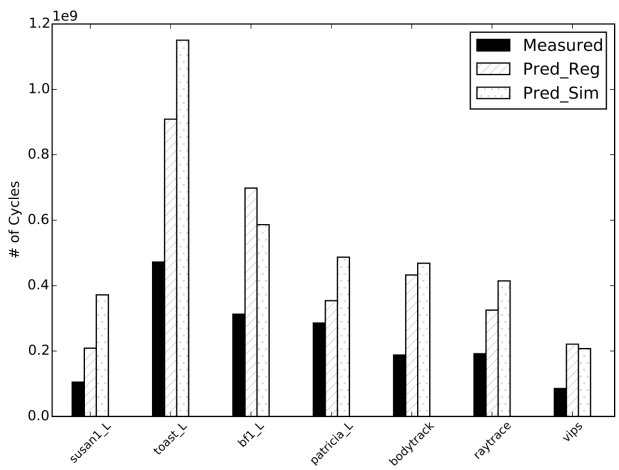


Figure 3.3: Prediction Error Distributions.

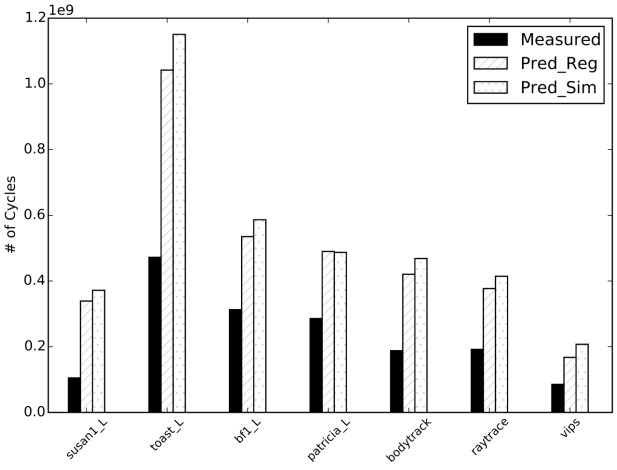
That being said, from the figure, it can be noted that the regression-based approach gives better estimations in most case. Even so, we refer to Figure 3.3 in which the prediction error distributions obtained using the simulator (ZSim) and regression (MMG, SMG) with regards to the real measured execution times are plotted. On average, the model generator gave upto 8% improvement in terms of prediction accuracy when compared to the ZSim simulator. Another remark is that the predictions observed from the models H1T and H2T, either by using the measured data (MMG) or simulated data (SMG), do not differ greatly. This means that the prediction on a target architecture are not quite affected by the choice of the reference architecture. As a side note, given the small sample of generated observations from these experiments, it is difficult to perform a statistical study since any test will give weak assumptions to support or reject the remarks we made so far. With problems that involve Design Space Exploration in which a large number of configurations need to be investigated, the model generator approach surpasses the simulation-based method. Since the model generator requires the simulation of a small number of benchmarks to predict the execution times of a very large number of applications (which could grow to thousands) as compared to simulation, which takes a prohibitive time to simulate the same number of applications, we can say that the model generator provides estimations closer to that of the simulator or better within a much reasonable time.

3.5 Conclusions

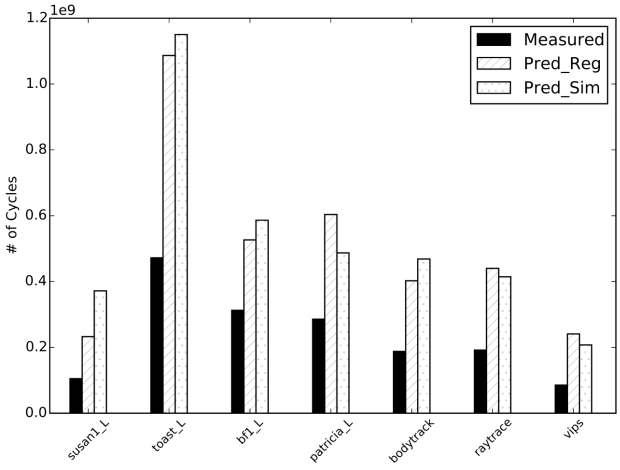
With complex real-time systems development, a gap is created between system developers and integration experts due to a limited knowledge in software engineering. The lack of data needed to integrate the system properly, ensure the timing requirements are met and optimize a set of performance objectives was motivation to propose the work presented in this paper. A model generator based on regression analysis is introduced to estimate the speed-up expected when migrating from a reference architecture, such as a component developer's machine, to a final target architecture to which the integrated system is deployed. The experiments showed that with minimum knowledge of the architectural details and no details on the specific behaviour of the application, the model generator was able to build models that could predict the execution times of a set of benchmarks with a 8% improvement in prediction compared to simulation predictions. The proposed framework is proven to be a better option for performance prediction of a large number of applications and especially for Design Space Exploration. We plan in the future to investigate the performance of the model generator with more accurate simulators and investigate the trade-offs between accuracy and speed that it could offer.



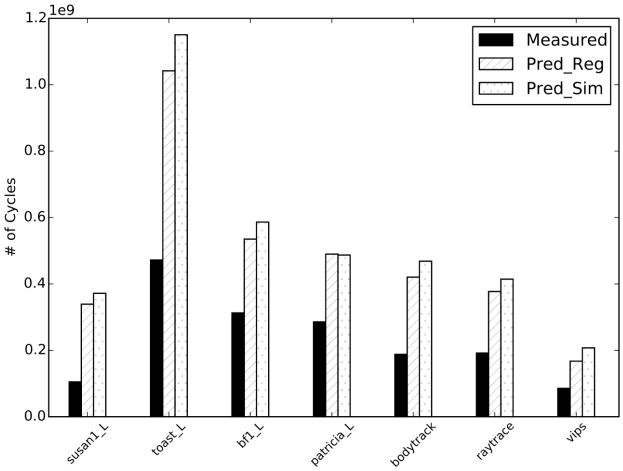
(a) H1TM: Model built on Measured data.



(b) H1TS: Model built on Simulated data.



(c) H2TM: Model built on Measured data.



(d) H2TS: Model built on Simulated data.

Figure 3.4: Measured vs Simulation vs Predicted performance.

CHAPTER 4 ARTICLE 2 – AN ANALYSIS OF RANDOM CACHE EFFECTS ON REAL-TIME MULTI-CORE SCHEDULING ALGORITHMS

Preface

As the lack of relevant information necessary to employ existing technologies, especially those developed for DSE and optimization of resources became apparent, our inquiry lead us to investigate characteristic such as memory contention that, with today's technological advances, both in terms of multi-core systems and scaling of technology, has lead to a non-negligeable trait in the interference measured when executing software on different cores sharing memory space. For such, the work presented in here re-evaluates the scheduling of applications of real-time multi-core systems by adding an awareness of random cache sharinf in the estimationg of the probabilistic worst case execution time (pWCET) and extending the RTS verification toolkit with a multi-core scheduling simulator.

Full Citation I. Hafnaoui, C. Chen, R. Ayari, G. Nicolescu and G. Beltrame, "An analysis of random cache effects on real-time multi-core scheduling algorithms," in *Proceedings of the 28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype*. ACM, 2017, pp. 64–70.
DOI: https://doi.org/10.1145/3130265.3130320

Abstract

The effect of sharing the last-level cache (LLC) among cores in a multi-core system has not been thoroughly investigated especially in the design of efficient scheduling algorithms. And with the growing interest in random caches, which allow for an easier estimation of the worst-case execution time of tasks in critical real-time embedded systems, tools that analyse the sensitivity of workloads to sharing the LLC become necessary. In this paper, we extend a real-time multiprocessor scheduling simulator, SimSo, with a framework that incorporates a random cache model for multi-level caches to evaluate emerging scheduling algorithms under the influence of shared caches. A set of experiments were performed to study the behavior of workloads with respect to worst-case response time, average slack time, and maximum

utilization, with varying cache designs under different scheduling algorithms.

4.1 Introduction

Extensive work has been proposed in literature to study caches and their effects on single and multiple path applications, especially when timing analysis is concerned. The pessimism of worst-case execution time analysis introduced by the determinism of some architectures led some researchers to investigate probabilistic architectures, namely those equipped with caches running with random placement/replacement policies. Since pathological cases are avoided by random behaviors, the pessimism of worst-case execution time (WCET) can hence be decreased [38]. These types of architectures opened doors to better timing analysis due to their predictability and hence offered tighter bounds than their counterparts. A pseudo-random replacement policy, for instance, has been integrated and applied to some ARM processor caches, such as the ARM Cortex-R5, which is targeted towards real-time embedded systems [1]. At the same time, due to the power wall, a term used by computer architects to refer to the roadblock that CPU manufacturers hit due to a higher heat production and power consumption, architects turned towards multi-processor/core architectures to counter the rise in demand for a faster processor clock and the ensuing power and temperature issues [80]. As a result, research changed focus from ways to accelerate execution of software on a single processing element to proposing approaches to concurrently execute software applications on a multiprocessing platform while optimizing certain objective metrics. Different schedulers have been proposed for this purpose and a lot of effort has been put to evaluate these techniques either through theoretical models or simulation techniques.

Simulators, for instance, help designers to choose the best configuration for their systems by providing metrics such as response time of tasks, processor load, etc., which are attributes that can change as scheduling algorithms and allocations are altered. Several scheduling simulators, both commercial and academic, have been proposed for the endeavour of investigating existing and emerging schedulers with varying characteristics. RTMultiSim was proposed by Hangan et al. [81] to simulate the behavior of multiprocessor real-time systems for the purpose of evaluating allocation and scheduling algorithms. It provides an abstract support of distributed systems in the form of message passing over a network. SPARTS [82] is a similar simulator in that it simulates scheduling properties of task-sets on a multiprocessor platform. However, it extends the classical scheduling simulator to incorporate power models, such as DVFS, which provide power related metrics such as energy consumption of tasks. Other simulators such SimSo [27] focused solely on providing a tool that can effort-

[1] http://www.ti.com/tool/launchxl2-rm46

lessly evaluate a large number of scheduler algorithms on a multi-core real-time system. The work proposed by the authors of [19] demonstrates the relevance of simulators in exploring design choices in which ZSim [83] was employed to obtain statistics to aid in predicting the performance of a wider range of architectures without incurring in the overhead of simulating every possible hardware/software configuration.

The common assumption that these simulators take is that overheads caused by the hardware and operating system is either included in the worst-case execution time (WCET) or considered negligible. Yet, as the number of cores increases, so does the overhead of sharing memory space among multiple software applications running at the same time. A study by Mars et al. [43] compared the cross-core interference on Intel's Core i7 Quad and AMD's Phenom X4 which share a large last-level cache (LLC) of 8MB and 6MB respectively, with their four cores. The performance degradation in these experiments were observed to reach 35%, which accentuates the need to study the effect of sharing the LLC when analyzing the timing behavior.

To be able to estimate the overhead of shared caches on multicore platform, various models have been proposed with varying degrees of accuracy and complexity. A simple model based on frequency of access (FOA) was proposed by Chandra et al. [84] to predict the execution time of an application when it shares its high level cache with another application. FOA is employed to estimate the application's cache occupancy and stack distance profiles are then used to estimate the execution times. Since relying on stack distance profiles could be computationally heavy, Sandberg et al. [85] proposed the use of the fetch rate as a function of cache size as a low-overhead input to estimate the shared cache contention for random and LRU caches by considering volatile and sticky data. Pan and Jonsson [45] proposed a framework that predicts shared cache performance that can be applied to configurations with different replacement policies such as random, LRU, Pseudo-LRU, etc. in which cache associativity is considered. The approach relied on the use of reuse distance as an input to a Markov chain for the purpose of predicting the cache miss rates. Reuse distances were employed as well by Eklov et al. [48] to obtain the interleaving stack distance profiles on a shared cache and estimate the miss rates of caches. A recent work by Sanchez et al. [46] proposed a common framework based on absolute reuse distances that accurately models probabilistic caches among other replacement policies.

Despite the wide-spread presence of research on shared cache performance estimation, their integration in real-time scheduling simulators stays limited. In a recent work by Cheramy et Hladik [47], the authors of SimSo have ventured into this field by extending the framework to integrate cache models; an LRU cache model in particular.

In this paper, we introduce a framework that incorporates contention consciousness in the timing analysis as well as scheduling algorithm design and hence a tool to optimize the hardware/software structures. In particular, our contributions can be summarized as follows:

1. We introduce a compact random cache model for multi-level caches on real-time multi-core systems based on the work of Eklov et al. [48], to estimate interleaving reuse distance in shared caches. We then extend the work [40] to calculate the cache hit probability and the Cycle Per Instruction (CPI) of tasks sharing the last-level cache. The proposed model offers a less laborious execution of the simulator as opposed to other models that require the calculation of multiple probabilistic distributions as is the case with the framework proposed by Beckmann et Sanchez [46]. The reliance of the model on reuse distances instead of stack distance profiles decreases the computational cost as well and makes it possible to integrate the random cache model into scheduling simulators.

2. We integrate the random cache model into SimSo's Execution Time Model, which estimates the miss ratio at all cache levels of an architecture and calculates the CPI of tasks executing under a certain scheduling policy and considers all scheduling cases. This way, we establish a framework that is able to evaluate scheduling algorithms in the presence of shared random caches.

The rest of the paper is organized as follows. Section 4.2 defines necessary notions on reuse distance and Static Probabilistic Timing Analysis (SPTA) technique for single-level random caches. In addition, we introduce a random cache model for multi-level caches and describe its integration into SimSo. Experimental evaluations of the proposed framework are presented and discussed in Section 4.3. Finally, we give our conclusions and future work in Section 4.4.

4.2 Contention-Aware Scheduling with Probabilistic Caches

To be able to design schedulers conscious of the memory contention, we extend a previously proposed model for private random caches [40] based on reuse distance to offer a framework that conservatively models a shared random cache.

4.2.1 Interleaved Reuse Distance

The reuse distance represents the number of memory accesses between two consecutive uses of the same memory address. Figure 4.1 shows a sequence of memory accesses. The reuse distance is agnostic to the memory addresses accessed between two consecutive accesses to

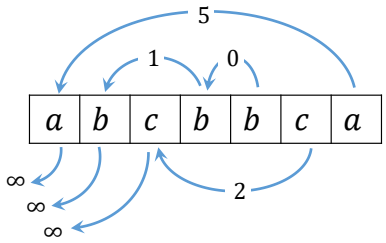


Figure 4.1: Reuse distance of memory accesses in a memory trace.

the same memory address. It rather reports the frequency of memory accesses, as can be seen for the reuse distance of the second access to address a which is equal to the number of accesses $rd_a = 5$.

The static probabilistic timing analysis (SPTA) we propose (defined in Section 4.2.2) relies on this definition of reuse distance to estimate the WCET. For private caches, the reuse distance can be obtained by extracting the program's memory trace. However, employing the same method to obtain the reuse distances for the LLC when tasks are sharing memory space is unrealistic. There have been extensive studies on shared cache impact, such as [84, 48, 85]. We were inspired by the technique proposed by Eklov et al. [48] to obtain an interleaved reuse distance, defined as the reuse distance of a task sharing memory space with tasks running on other cores. The models employing this definition of the reuse distance reported a mere 2% error compared to those of simulations. In this paper, the reuse distance of a shared random cache for a system with 2 cores has been extended to n cores.

Since reuse distance abstracts the address of the memory accesses, interleaved reuse distance implies that a number of instructions n_t of task t_j running on core j are fetched in an interval of time T. This interval of time is defined as the time between two accesses to the same memory address of task t_i co-running alongside t_j on core i. From this, we introduce the formula,

$$pmi_i = \frac{\#memory\_accesses_i}{\#instructions_i} \tag{4.1}$$

where pmi_i is the proportion of memory instructions for task t_i. Then, the time interval T for reuse distance of rd_i is

$$T = \frac{rd_i}{pmi_i} cpi_i \tag{4.2}$$

where cpi_i is the CPI of task t_i on core i without interference of other tasks. Equation 4.2 builds the relationship between reuse distances rd_i and rd_j on two cores as follows

$$rd_i = rd_j \frac{pmi_i}{pmi_j} \frac{cpi_j}{cpi_i} \tag{4.3}$$

For a shared LLC, the new reuse distance $\hat{rd}_i$ of the interleaved stream for task t_i is,

$$\hat{rd}_i = \sum_{k=1}^{n} rd_k = rd_i \sum_{k=1}^{n} \frac{pmi_i}{pmi_k} \frac{cpi_k}{cpi_i} \tag{4.4}$$

4.2.2 Static Probabilistic Timing Analysis of Random Caches

A cache, with evict-on-miss random replacement policy, randomly selects a cache block to be replaced for every cache miss. As opposed to, for instance an LRU replacement policy, this random behavior avoids pathological cases with low occurrence probabilities which are hard to test and predict [38]. Disregarding extremely unlikely events, such as those with probabilities lower than the physical destruction of the device, helps in upper bounding the WCET estimation. Several formulae have been proposed for SPTA of single-level random caches based on program memory traces. In here, we adopt the formula proposed in [40] to compute the hit probability of each memory access for a single level cache. This formulation has been proved to offer a safe and optimal bound [39].

$$P(hit) = \begin{cases} (\frac{N-1}{N})^K & if K < N \\ 0 & if K \geq N \end{cases} \tag{4.5}$$

where N represents the cache associativity, and K the reuse distance. We define an Execution Time Profile (ETP) that represents timing information and their associated probabilities in the form of two vectors. We use the number of cache misses as a timekeeper (execution time may be used in other papers) and define $ETP = \{(m_1, m_2, ...), (p_1, p_2, ...)\}$, in which m_i is the number of cache misses, and p_i is its corresponding occurrence probability.

The hit probabilities of each memory access are conservatively obtained and used to build the ETPs. Since Equation 4.5 yields lower bound probabilities for cache hits, regardless of previous memory accesses, ETPs can be regarded as independent of each other. The convolution operator $\circledast$ is then proposed to put all ETPs together and obtain the ETP of a task. The reader is prompted to go through the work of Bernat et al. [86] for further proof and explanation of the use of ETPs and the convolution operator.

To combine the profiles ETP_x and ETP_y, their convolution can be calculated as

$$ETP_x \circledast ETP_y = \{(m_1, m_2, ...), (p_1, p_2, ...)\} \tag{4.6}$$

where

$$m_k = m_i + m_j$$

$$p_k = \sum_{\substack{i=1 \\ j=1}}^{x,y} p_{x,i} \cdot p_{y,j}$$

We assume two ETPs: $ETP_1 = \{(1, 2, 3), (0.1, 0.3, 0.6)\}$ and $ETP_2 = \{(1, 3), (0.2, 0.8)\}$. Equation 4.6 produces the convoluted ETP

$$ETP_1 \circledast ETP_2 = \{(2, 3, 4, 5, 6), (0.02, 0.06, 0.2, 0.24, 0.48)\}.$$

In here, we emphasize the fact that the convolution operation used to combine the ETPs introduces a pessimism in the estimation of the probabilistic WCET. Although this deteriorates the accuracy of the cache model proposed in here, it provides estimations that are safe in nature and upper bounded. Considering that one of the target platforms for the simulator are critical hard real-time systems with strict deadlines, the introduction of this pessimism ensures the design of reliable systems.

4.2.3 Random Cache Model

In this section, we introduce a random cache model for multi-level caches, which is based on reuse distance and single-level SPTA techniques and is easily integrable into real-time simulators. The architecture we analyze hereforth has a memory system of L levels that consists of $L - 1$ cache levels, in which the LLC is shared among the n cores, and a main memory. In what follows, we detail the operations that we aim to incorporate into the simulator.

Step 1: A memory trace is collected for every task t. These memory traces are used to obtain the reuse distance rd_t. The method described in Section 4.2.1 is invoked to estimate the interleaved reuse distances when different tasks are executing on an L_x level memory. The hit probabilities of memory accesses are calculated using interleaved reuse distance from Equation 4.5. These are in turn used to construct ETPs that are then convolved by Equation 4.6 to obtain the overall ETP.

Step 2: To get the local miss ratio for task t at L_x level memory without considering previous hit/miss impact, assuming that for an L_x level memory, $ETP = \{(m_1, m_2, ...), (p_1, p_2, ...)\}$, we have

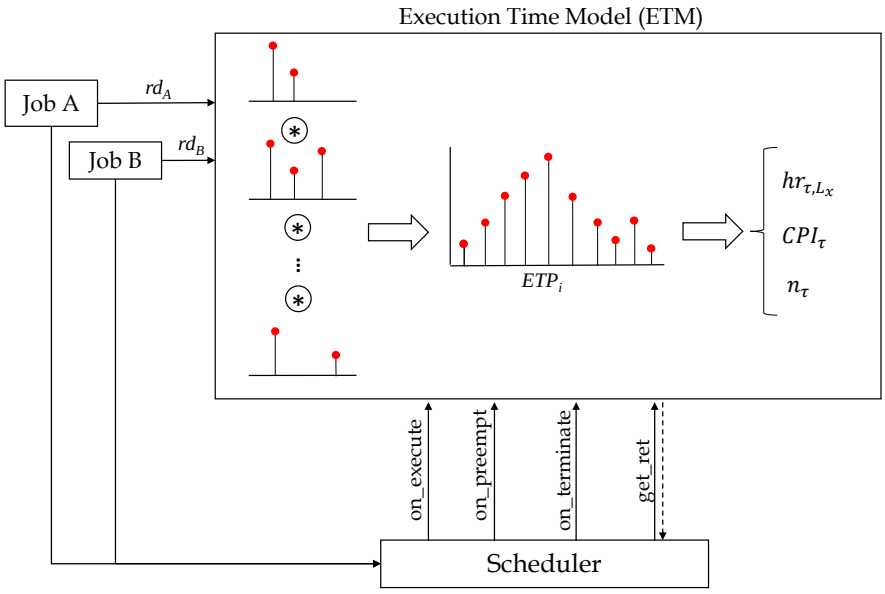


Figure 4.2: Integration of the cache model into the SimSo's ETM.

$$lmr_{t,Lx} = \frac{\sum_i m_i p_i}{max(m_i)} \tag{4.7}$$

where $\sum_i m_i p_i$ is the expected value for cache miss counts, $max(m_i)$ is the number of memory accesses.

With local miss ratio $lmr_{t,Lx}$, we add consideration of previous hit/miss impact in calculating the hit ratio at each level of memory. To have a cache hit at the L_x level memory, all lower level memory accesses $(L_1, ..., L_{x-1})$ must be cache misses. Thus, the hit ratio is computed as

$$hr_{t,Lx} = (1 - lmr_{t,Lx}) \prod_{i=1}^{x} \alpha_{L_i} \tag{4.8}$$

where $\alpha_{L_i} = \begin{cases} 1 & i = 1 \\ lmr_{t,L_{i-1}} & otherwise \end{cases}$

Step 3: The computed CPI for task t is denoted as cpi_t and can be used to describe system behavior. To make it contention-conscious, cpi_t is calculated as

$$cpi_t = \sum_{x=1}^{L} hr_{t,L_x} t_{L_x} \tag{4.9}$$

where hr_{t,L_x} is the hit ratio of L_x level memory access for job t, and t_{L_x} is the access time in

terms of cycles. Note that $\sum_{x=1}^{L} hr_{t,L_x} = 1$. We assume that the access time at each level of memory is constant.

4.2.4 Execution Time Model Integration

Through the modular design of SimSo, easing the integration of new modules, we aim to extend the behavior of the *Execution Time Model* (ETM) with our cache model. Figure. 4.2 shows an abstract representation of the involved modules and their interactions. The cache model is implemented within ETM, which handles the temporal behavior that is expected of a job τ of a task t at a particular point in time. We implement the random cache model in the Execution Time Model (ETM) of the SimSo simulator by incorporating the steps described in the previous section. As can be seen from Figure 4.2, the ETM is an event-triggered module. It updates the number of instructions executed every time an event occurs; whether it is an activation, pre-emption or a termination of job τ. With this, two modes of time modelling are implemented; the "no contention" mode and the "shared cache" mode.

- The "no contention" mode returns the execution time of jobs that are executed sequentially or on a single core. This covers the instances in which only a single job is running in one of the cores.
- The "shared cache" mode returns the execution time of jobs sharing then last level cache. This refers to the instances in which jobs are running in parallel on different cores and in which the memory contention might increase the miss ratio of tasks.

These concept are implemented for the cache model integration by calculating the number of instructions n_t as

$$n_\tau = \frac{\delta_\tau}{cpi_\tau} \tag{4.10}$$

where δ_t is the time interval between two events for a job τ and cpi_τ is the CPI expected for job τ as calculated by Equation 4.9.

Since scheduling policies affect which jobs can run together, the cache model implementation will adjust its calculations to match the timing activations sent by the `Scheduler`. When an `on_activate` event is generated, the Scheduler sends an event to the ETM with the ID of the activated job. In here, the ETM uses the reuse distance rd_t if the job is running by itself. In case a second job is activated on a different core, the ETM will switch to the "shared cache" mode in which the interleaved reuse distances are calculated for the running jobs. In this

manner, the ETM will switch between the “no contention” and the “shared cache” modes depending on the current state of the scheduler and the events sent to the ETM that describe the state of jobs.

4.3 Experimental Evaluation

4.3.1 Experimental Setup

To study random cache effects, we choose Mälardalen benchmarks – a popular suite for WCET and random cache analysis – for evaluation. We use gem5 [87], an instruction set simulator, to generate memory access traces for an ARM processor architecture with a Floating Point Unit (FPU) and statically link all libraries. Each benchmark is regarded as a task with a memory trace obtained separately from other benchmarks. We also only consider instructions relevant to the benchmarks while we ignore the instructions of system calls.

In the experiments, we adopt the simple architecture of Figure 4.3 that contains two cores with a 4 Kb L1 private cache each and share a 32 Kb L2 as a LLC and a 2 Mb main memory in which the access times are defined within the arrows.

In each of these experiments, the memory access trace of a benchmark is used to obtain the corresponding reuse distance. The reuse distance of the benchmarks to be tested is supplied as an input to the simulator’s ETM alongside the number of instructions as was previously stated in Section 4.2.4.

4.3.2 Experimental Results

The purpose of the following experiments is to show the importance of considering memory contention when scheduling is concerned, as well as, demonstrate the utility of the proposed

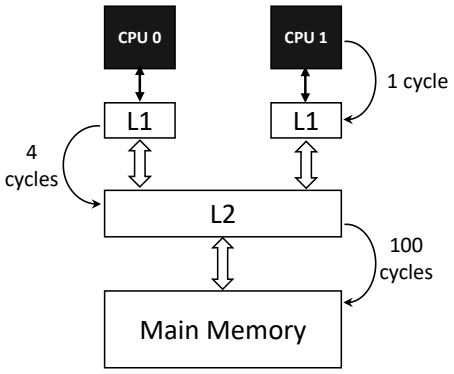


Figure 4.3: A two-level cache hierarchy.

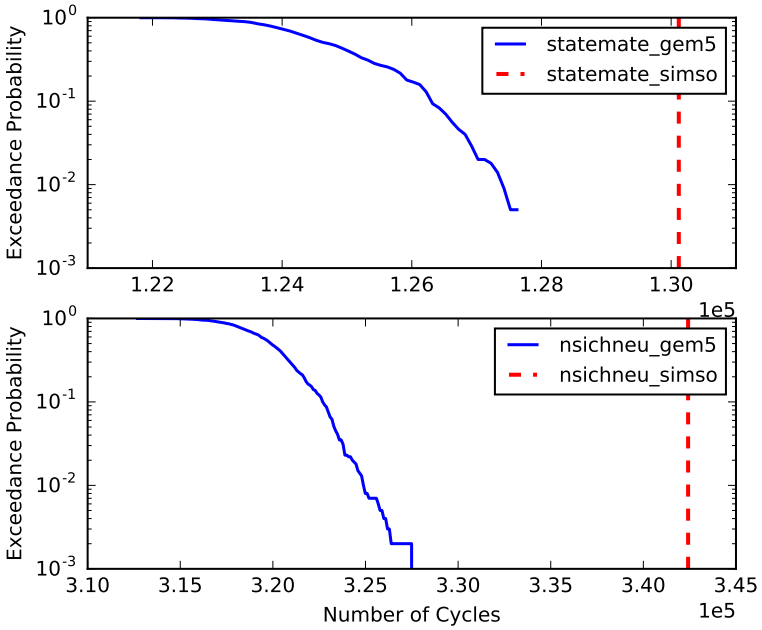


Figure 4.4: Executed cycles obtained from gem5 and SimSo.

tool in optimizing the hardware/software configuration by tuning the parameters of the cache and both existing and emerging scheduling policies.

Cache Boundedness

To validate the proposed cache model, Gem5 was used to simulate the execution of Malardalen benchmarks on the architecture of Figure 4.3. Figure 4.4 plots the exceedance probability which defines the probability to exceed a certain number of executed cycles. Since scheduling events affect the choice of reuse distance (single or interleaved) used for the timing analysis, it is difficult to obtain the same exceedance curve for the cache model. For such, we compare the curves obtained from Gem5 with the bounded values used by SimSo. For space limitations, we only show the results for two benchmarks, `nsichneu` and `statemate`, which are characterized with a higher sensitivity to changes in cache occupancy, however similar observations were obtained for the rest of the benchmark suite. This shows that the cache model offers a safe bound that can only be reached at very small probabilities.

To validate the proposed cache model, gem5 was used to simulate the execution of Malardalen benchmarks on the architecture of Figure 4.3. Figure 4.4 plots the exceedance probability which defines the probability to exceed a certain number of executed cycles. Since the knowledge of scheduling events are required, it is difficult to obtain the same exceedance curve for the cache model. For such, we compare the curves obtained from gem5 with the

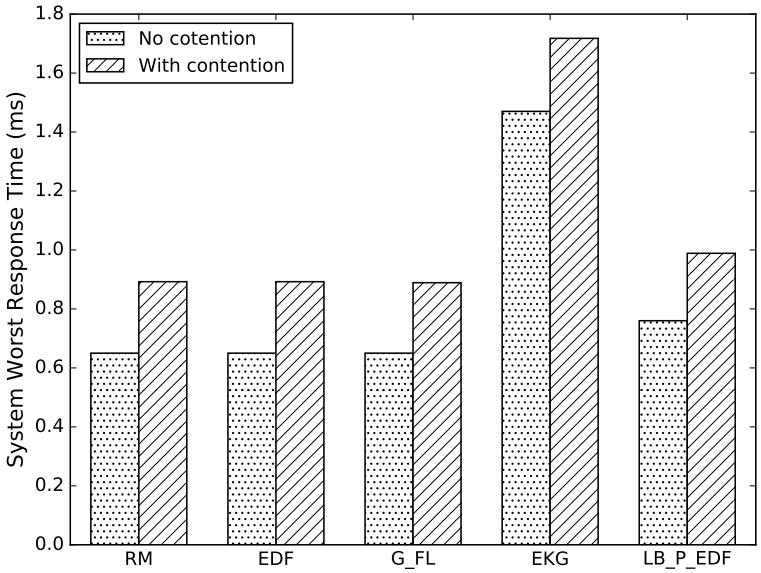


Figure 4.5: The worst response time with and without contention consideration.

Table 4.1: Task-set Γ specifications with every task's release time at $A_i = 0$ and $t = \{C_t, D_t, T_t, cpu_t\}$ defined as the worst case execution time (used for no contention), deadline, period and allocation (when applicable) respectively.

Task	$C_t(ms)$	D_t(ms)	T_t(ms)	cpu_t
`fdct`	0.20	2.0	2.0	CPU 2
`fft1`	0.09	1.0	1.0	CPU 1
`minver`	0.19	3.0	2.0	CPU 2
`nsichneu`	0.54	2.0	2.0	CPU 1
`qsort-exam`	0.07	1.0	1.0	CPU 2
`select`	0.07	1.0	1.0	CPU 2
`statemate`	0.12	2.0	2.0	CPU 2

bounded values obtained from SimSo. The cache model offers a safe bound that can only be reached at very small probabilities. For space limitations, we only show the results for two benchmarks, `nsichneu` and `statemate`, which are characterized with a higher sensitivity to changes in cache occupancy.

Cache Contention

The experiment involves executing a set of benchmarks Γ with the specifications of Table 4.1 under five different schedulers, (G_FL: global fair lateness scheduling, EKG: multiprocessor scheduling with few pre-emptions, EDF: partitioned earliest deadline first scheduling, RM: partitioned rate monotonic scheduling, LB_P_EDF: a load-balancing partitioned earliest

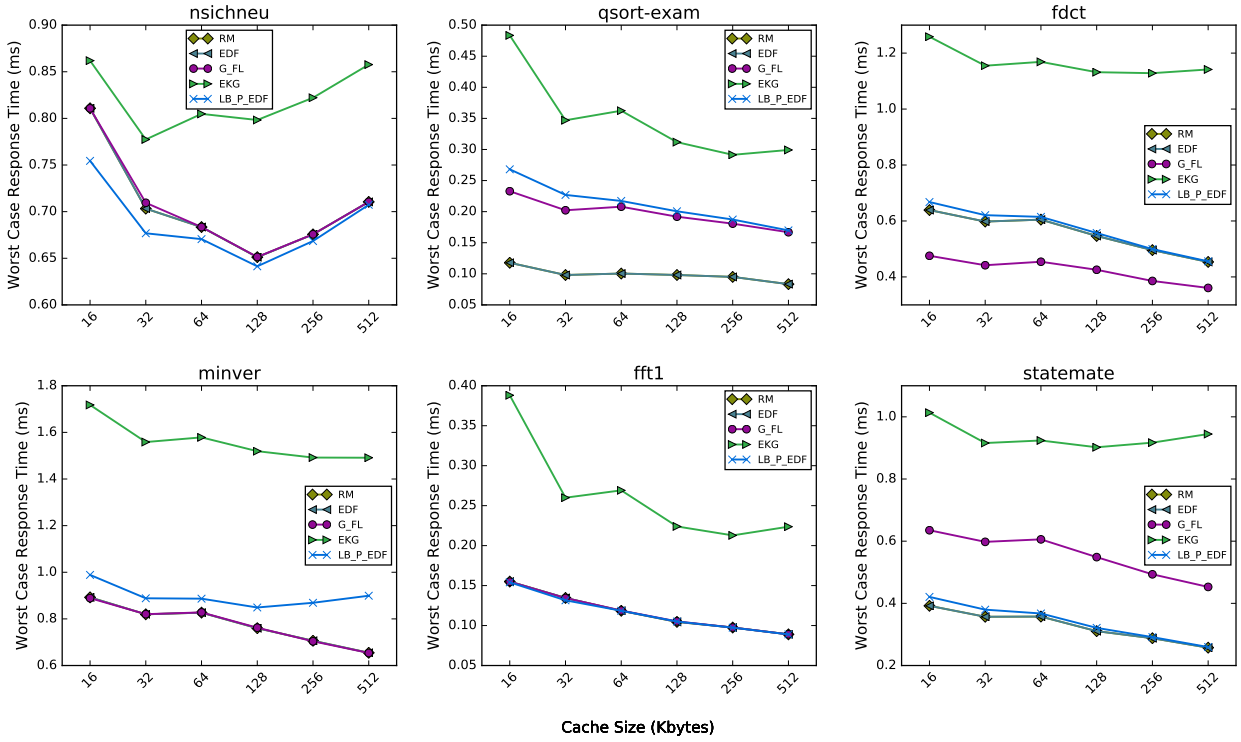


Figure 4.6: Worst Case Response Time for a subset of Γ tasks under different schedulers for a random cache.

deadline first scheduling.). The objective is to observe the system's worst response time $SWRT$ defined as the worst response time at which all tasks in the task-set have finished executing at least once.

$$SWRT = \max_{t \in \Gamma} (WCRT_t^{(1)})$$

where

$$WCRT_t = \max_{\tau \in t} (c_\tau - a_\tau) \tag{4.11}$$

with a_τ and c_τ as the activation and completion times of job τ respectively.

Figure 4.5 plots the SWRT observed when the set is scheduled without considering the cache contention (*no contention*) and when cache contention is considered (*with contention*). We can clearly see that the response time of the system is higher when the scheduling is conscious of the cache contention regardless of the scheduling policy applied at the time. This is due to the fact that the execution time of tasks increases since they incorporate the effect of sharing the cache among other tasks in different cores.

These observations might not have a huge impact when soft real-time task-sets are studies since, by definition, these types of tasks can withstand a certain level of degradation in performance if they miss their deadlines. The same can not be said when studying hard real-time systems in which the timing specifications are very strict and a missed deadline can lead to unforeseen damage.

Impact on Cache Design

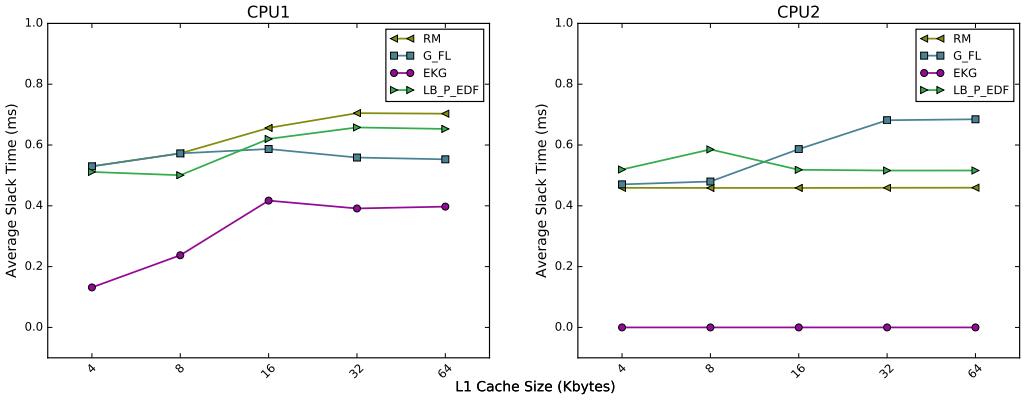


Figure 4.7: System's Minimum Slack reached under different scheduling policies while varying the L1 cache size of core 1.

Cache design plays an important role in system performance and for such, we investigate its effect on the temporal behavior of the system. We apply different cache configurations and demonstrate its effect on the performance of scheduling algorithms and the system as a whole. In this experiment, the L1 cache size is fixed to a relatively small size (4 Kb), and the L2 cache size is varied from 16 Kb to 512 Kb. The access time is changed as well to match the change in cache size and follows the configuration Φ =[*(16 Kb, 2 cycles), (32 Kb, 4 cycles), (64 Kb, 10 cycles), (128 Kb, 12 cycles), (256 Kb, 13 cycles), (512 Kb, 15 cycles)*].

Since the worst case response time (WCRT) of a task is one of the characteristics often studied when designing and optimizing real-time systems and scheduling considerations are taken, we adopt this metric in evaluating our cache integration. The WCRT of task t is as was defined previously with Equation 4.11.

Figure 4.6 shows the worst case response time of a subset of tasks as L2 cache size increases under different schedulers. Comparing the response time of tasks in terms of scheduling policy, we notice that although (RM, EDF, LB_P_EDF and G_FL) seem to result in lower response times, EKG performs poorly with most tasks in term of WCRT. This is due to the nature of the scheduling policy which allocates tasks from a global point of view. In

this case, the whole set was allocated to core 1 which increased the WCRT of most tasks, especially those with lower priorities (more pre-emptions). In most cases, the WCRT of tasks is observed to decrease as we expend the size of the LLC as can be seen for (`qsort-exam, fdct, minver`). This is a result of a faster execution time due to a lower miss rate at the L2 level. We note however that `nsichneu` displays an increase in WCRT with a cache size of 256 Kb and 512 Kb. Investigating this case revealed that, as opposed to the schedule with smaller cache sizes in which `nsichneu` was mostly co-running with a single task that had a higher execution time, with bigger cache size and due to the lower time execution of the other tasks on core 2, `nsichneu` was co-running alongside multiple tasks that had different bahviors. This affected the allocation of cache space to `nsichneu` and hence increased its execution time. We are able to observe this kind of behavior thanks to the inclusion of contention awareness into the simulator that was otherwise not detectable.

Heterogeneous Multi-core Systems

Real-time simulators that are able to model heterogeneous platforms are widely desirable. Being able to incorporate the same sense of heterogeneity in cache models is more advantageous. In here, we study the effect of having different private cache designs, namely cache memory size, on the choice of scheduling policy. The same task-set Γ of Table 4.1 is scheduled under the policies [RM, G_FL, EKG, LB_P_EDF] while fixing the size of the last-level cache to 128 Kb. We fix the size of the private cache for core 2 as well to 4 Kb and vary the size of the L1 of core 1 with $\hat{\Phi}$ = [*(4 Kb, 1 cycles), (8 Kb, 1 cycles),(16 Kb, 2 cycles), (32 Kb, 4 cycles), (64 Kb, 10 cycles)*].

To study the effect of heterogeneity by varying the size of one core's L1 cache, we adopt a metric that is sought extensively when optimizing real-time systems; *System slack time*. The slack time is defined as the time in which the CPU is idle after the execution of the task-set. With critical hard real-time systems that tend to enforce predictability in system scheduling, such as those encountered in automotive and avionic domains, having a large slack time is desirable since it gives a safe margin in case an unexpected event occurs that largely increases the execution time of the task-set.

Having this in mind, we formally define an upper bound as a metric *System Minimum Slack* (SMS) defined as the minimum slack time reached when the execution of a task is completed and after which no task is running.

$$SMS = \min_{t \in \Gamma} (\min_{\tau \in t} (T_\tau - C_\tau))$$

Figure 4.7 reports the system's minimum slack time observed for core 1 and core 2 under different schedulers. As expected, we observe that, as the size of the private cache for core 1 is increased, the execution time of tasks decreases which results in more slack time as can be seen for [RM, EKG, LB_P_EDF]. This is due to the lowered miss ratio at the L1 level which decreases the time to fetch instructions from the L2 cache. We can see that EKG exhibits smaller slack times and that is for the same reasons mentioned in the previous section in which EKG allocates all tasks on core 1 (which is the reason we see a 'zero' slack time on core 2). We notice however that G_FL, as opposed to its counterparts, shows a decrease in SMS as the cache size is increased and a reverse bahavior on core 2. This is due to the fact that the G_FL scheduling policy is a similar implementation of partitioned EDF that allows migration of tasks to other cores. For the purpose of balancing the load on both cores, some tasks were allowed to migrate from core 2 to core 1 which had the effect shown in Figure 4.7.

Due to the small size of the benchmarks studied here, some behaviors were not able to be investigated in these experiments. A larger benchmark might show additional behaviors such as an increased slack time for core 2 as well since increasing the size of L1 in core 1 results in less shared space in L2 between the two cores which we expect would decrease the miss ratio at the LLC level for tasks in core 2.

4.4 Conclusions

We proposed a random cache model for multi-level caches, compact enough to be easily integrable into modern simulators. For that endeavor, we integrated our random cache model into a real-time scheduling simulator, SimSo. We presented a framework which allows for the optimization of architecture and software alike and helps system engineers make better design choices when it comes to the desired temporal characteristics of the system. The proposed model offers safe bounds to the estimated cache contention which allows for the implementation of more reliable systems especially when hard real-time systems are of concern. In the future, we plan to study cache coherence effects on scheduling algorithms and investigate other timing analysis techniques such as Measurement Based Probabilistic Analysis (MBPTA).

CHAPTER 5 ARTICLE 3 – SCHEDULING REAL-TIME SYSTEMS WITH CYCLIC DEPENDENCE USING DATA CRITICALITY

Preface

When it comes to scheduling and integrating large software systems, a clear description of the system and the inter-dependencies among its different parts, such as a DAG, must be established in the very early stages of development. Many approaches proposed through literature are also based on this type of graph representation. The enability to achieve this description, which alas is more common in industrial application than not, aggravates the smooth opertation of system integration and limits accessibility to powerful techniques. In this work, we borrow the interpretation of information as a property that is characterized with flow and present an approache to attribute criticality levels to the data shared among the modules of the system to identify which relationships can be discarded in order to achieve a DAG representation. The approach becomes valuable when the modules are viewed as black-boxes and offer starting configurations to accelerate the integration phase.

Full Citation I. Hafnaoui, R. Ayari, G. Nicolescu and G. Beltrame, "Scheduling real-time systems with cyclic dependence using data criticality," *Design Automation for Embedded Systems*, vol. 21, no. 2, pp. 117–136, 2017.
DOI: https://doi.org/10.1007/s10617-017-9185-9

ABSTRACT

The increase of interdependent components in avionic and automotive software rises new challenges for real-time system integration. For instance, most scheduling and mapping techniques proposed in the literature rely on the availability of the system's DAG representation. However, at the initial stage of system design, a dataflow graph (DFG) is generally used to represent the dependence between software components. Due to limited software knowledge, legacy components might not have fully-specified dependencies, leading to cycles in the DFG and making it difficult to determine the overall scheduling of the system as well as restrict access to DAG-based techniques. In this paper, we propose an approach that breaks cycles based on the assignment of a degree of importance and that with no inherent knowledge of

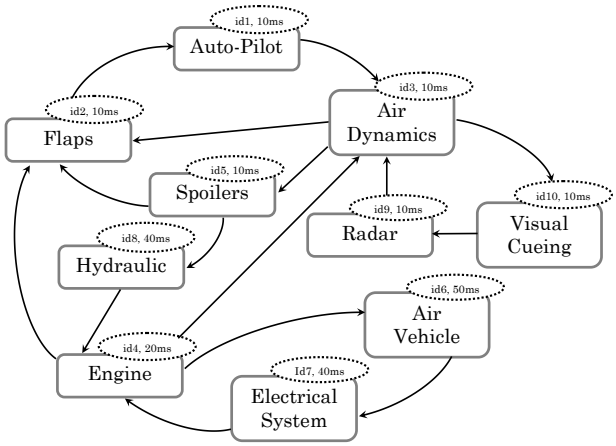
the functional or temporal behaviour of the components. We define a "criticality" metric that quantifies the effect of removing edges on the system by tracking the propagation of error in the graph. The approach was reported to produce systems $(56 \pm 14)\%$ less critical than other methods. It was also validated on two case studies; a data modem and an industrial full-mission simulator, while maintaining the correctness of the system.

5.1 Introduction

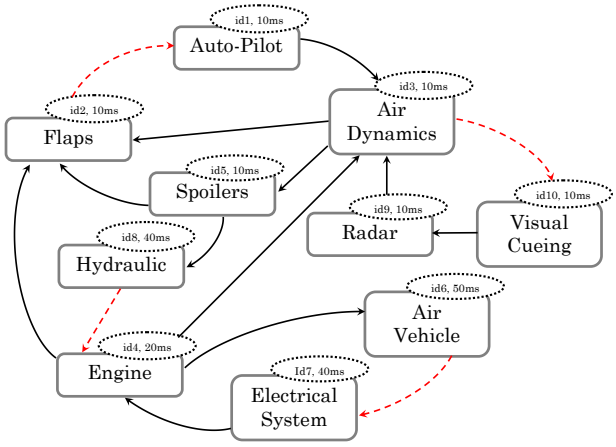
When scheduling complex real-time systems, such as those encountered in the avionic and automotive industries, engineers widely rely on a representation of the system in the form of a directed acyclic graph (DAG). This is usually referred to as a task graph in which nodes represent system components and edges, the communication between them. A system component is an encapsulated entity treated as a simple task and mapped and scheduled in the same manner. A DAG representation of a system is the basic input for various methodologies proposed in literature. The motivation of these works vary from the analysis of system performance and feasibility [88] to the optimization of the mapping and scheduling of real-time systems [89, 90], to involving artificial intelligence to solve the issue of real-time scheduling [91]. Some tools such as YARTISS [92] and STORM [26] use a DAG as an input model to simulate the real-time behaviour of the system. In large-scale system development, it is generally assumed that the DAG representation is available at integration time. However, the reality on the industrial level says otherwise. Legacy components and architectures are oftentimes not fully specified which limits the availability of a DAG. Instead, the system representation is limited to a generic model such as the example in Figure 5.1a.

A generic avionic subsystem is shown in Figure 5.1a with the edges representing the flow of data between the components and the bubble stating the id. and rate at which they are executed. These are among other characteristics specific to every component that we formally define in Section 5.3. Scheduling this system becomes tedious even under a simple application of Rate Monotonic scheduling (RMS) with dependent tasks. However, with the DAG representation of Figure 5.1b, in which the cycles are removed, one possible schedule for the system under a pre-emptive RMS can be achieved as in Figure 5.1c and further methods to optimize certain objective metrics such as the throughput or the energy of the system become possible; an option that was very limited with the graph of Figure 5.1a.

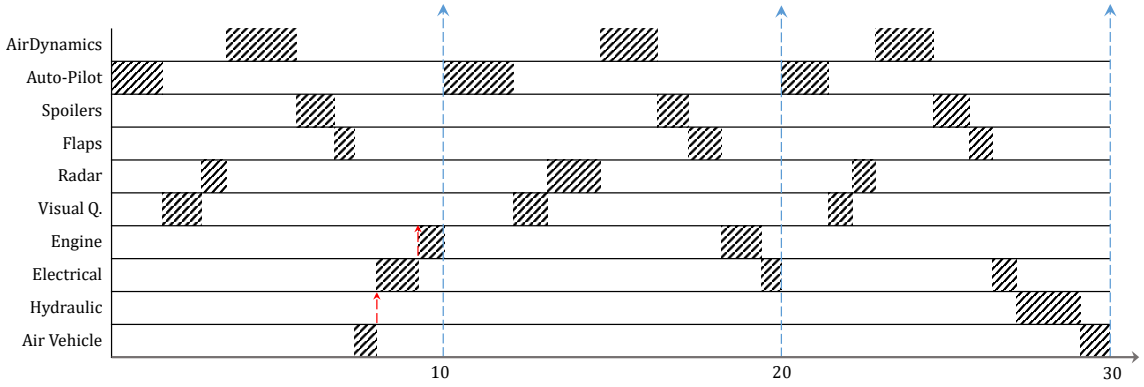
The inability to identify the execution order of the system components at integration time is a consequence of many scenarios, one of which is expecting engineers from different domains, such as mechanics, electrical engineering, etc., to be experts in their fields as well as software engineering. Moreover, due to the complexity of these systems, code is repeatedly reused



(a) DFG representation of the system with cyclic dependences.



(b) DAG representation of the system after cycle breaking.



(c) System scheduled under pre-emptive RMS.

Figure 5.1: Analysis and scheduling of a simple avionic system.

to decrease time-to-market and cost of software testing. On top of this, the system design sometimes requires the use of an Original Equipment Manufacturer (OEM)'s product which generally comes in the form of a binary component accompanied with a file to describe the component's inputs and outputs, often without specifying the inter-dependence nor the latency of a component. In all of these cases, the system components are viewed as black boxes which makes it difficult to gather concrete information about the components, unless specified by the designer. This limits the integration experts' accessibility to the system's task graph.

To model such systems, a dataflow graph (DFG) is employed to represent the dependence between components as well as the amount of data exchanged. However, different components are interconnected in such a way that their computations depend on the data generated by other components. This naturally creates cycles within the graph which complicates the scheduling process, as well as precedence assignments that schedulers will have to take into consideration. Furthermore, the existence of cycles in the system puts a hindrance to the application of many analysis tools and optimization techniques that are available for acyclic directed graphs.

Generally, integration engineers rely on the feedback of legacy component developers and their own expertise and knowledge of the components individual functionality acquired through the years to split the cycles. Alongside integration problems, this creates issues in other disciplines such as co-simulation. For the purpose of system verification and validation, different components are simulated using varied technologies. If we look at a simulation of an avionic system as an example, the tools used to simulate the electrical system are different from those that simulate the aerodynamics. In some cases, parts of the system are physically available, however presented as black boxes and engineers are required to co-simulate the rest of the system with limited knowledge about the components. The presence of cycles in the overall mapping of the system drives engineers to rely on recursive techniques to reach sound simulation results. Having a good initial configuration to the recursive process could reduce the design time extensively.

One popular approach to model systems that involve cyclic dependences is Synchronous dataflow graphs (SDFs). The authors of [93] presented a modular approach to analyze system performance that was applied directly to a cyclic SDF. The scheduling of an SDF was optimized in [94] using evolutionary techniques by considering the limitation of the size of the scratchpad memory. Although these approaches yielded good results, this model decreases the scope of approaches that study real-time systems since it excludes approaches based on DAGs. To this endeavor, other works [95, 31, 32] have been proposed to unfold an SDF

into a DAG by simply discarding the edges that contained delays. This makes DAG-based approaches to analyze and optimize the system accessible. For all that, these methodologies cannot be adapted to the problem that we presented so far for the sole reason that the edge property, delay, encountered in SDFs is among the metadata that are not available at integration time as discussed in the previous scenarios.

In this paper, we put forth an approach to transform a DFG with cyclic data dependencies into a DAG, with no inherent knowledge of the function and behaviour of system components for the purpose of opening access to DAG-based techniques. In here, we focus on simulations of real-time systems and systems such the informatics in automotive systems which are comprised of components with soft deadlines and scheduled under static schedulers. This is the case since certain standards are enforced when scheduling these types of real-time systems. Especially with the prevalence of the Integrated Modular Avionics (IMA) [96] to design avionic systems, static scheduling policies are preferred and sometimes imposed to enforce predictability in the system.

The approach we propose here is based on the idea of error propagation to eliminate cycles. We introduce a concept that describes the importance of data, which we label *criticality*, in which the effect of removing a certain edge is quantified as a characteristic of the data being carried by the edge. This is a key component in deciding which edges to discard and transform the DFG into a DAG.

The rest of the paper is structured as follows: Section 5.2 summarizes the work that has been proposed in literature to solve this issue; Section 5.3 gives a description of the system model. The approach to eliminate cycles based on criticality is detailed in Section 5.4 followed by a motivational example; The results obtained from a set of experiments and two case studies are reported in Section 5.5; Finally, conclusions are given in Section 5.7.

5.2 Related work

One of the important parameters that characterizes tasks in real-time system scheduling is their priority assignment. This, alongside task precedence constraints, could decide the schedule that drives the execution of the system. The assignment is generally attributed with a performance objective in mind decided by the designer when building the system such as schedule length, schedulability, etc.

Different approaches have been proposed to schedule the system and assign priorities with task dependences in mind. The authors in [97] and [98] tackled the issue from a mapping point of view in which a Genetic Algorithm (GA) was employed to allocate dependent tasks and

assign priorities on a multiprocessor system. The Distributed by Optimal Priority Assignment (DOPA) heuristic was proposed in [99] to address both the problem of finding a partitioning configuration and a priority assignment for tasks on a core that ensures the tasks don't miss their deadlines. The problem is extended to Network-On-Chip (NoC) systems in the work of Liu et al. [100] who proposed a dependency-graph based priority assignment algorithm (eGHSA) targeting NoCs with shared virtual-channels. On the other hand, the works in [101, 102, 103] relied on the system topology and focused on the basic idea that priorities should be chosen by the node's relative importance. In [104, 105], the concept of Global Critical Path is introduced which extends the top-level and bottom-level strategies by considering the critical path in the graph (the longest path from the source to the exiting node) and its branch paths. Sinnen et al. [106] proposed multiple extensions of the previous schemes to include communication contention and the number of successors. Be that as it may, the constant assumption seems to be that a graph representation of the system as a directed acyclic graph (DAG) is available. Other models that include cyclic dependences were proposed to bypass the issue of DAG inaccessibility. The work in [107] extended TTIG (Temporal Task Interaction Graph) [108]; a different model from a DAG that models cycles and bypasses some of the drawbacks of using a DAG. However, the cycles within this model were viewed as special nodes referred to as Composite Nodes to facilitate computation of path execution times and the presence of cycles was not dealt with head-on. A similar approach was proposed by Sardinha et al. [30] in which cycles were included in a special group called Strongly Connected Components (SCC) to facilitate mapping of tasks onto a number of processing elements. The authors in [109] proposed a modified Depth First Search (DFS) algorithm that splits the cycles. Yet, once again, the assumption was that all edges are of the same importance and the focus of the paper was on shortening the critical path of the resulting DAG to reach a better makespan for scheduling the DAG. The presence of cycles in attack graphs in the field of cyber security was addressed by Huang et al. [29] in which the authors identify two types of cycles; the ones that cannot be executed irrespectively of which edge was removed and hence discarded the cycle, and those that cannot be removed. The approach relies on the functionality of the nodes and characteristics specific to security modules to remove the cycles and hence cannot be generalized to other scenarios from different domains.

In this work, we set forth a methodology to break the cycles in a dataflow graph with no inherent knowledge of the components behaviour or execution times that relies on a definition of data criticality.

5.3 System Model

In this paper, we deal with systems similar to full-mission simulators (FMS) in which system components are viewed as tasks with execution times, deadlines and execution rates, and thereby mapped and scheduled as a generic taskset. To be able to visualize the system, better understand the dependencies and build our approaches on mathematical grounds, we define two types of graphs; the *dependence graph* and *task graph*.

Due to the partially specified components, the *dependence graph* represents the system at its raw state. The exchange of data between components is used to track the intra-component communication and build the *dependence graph*. Note that the components that depend on the data produced by other components do not wait for said data to be generated to start executing. Rather the data is accessed through a shared memory. This, in turn, implies that, depending on the schedule, the data read by a component at a point in time could be possibly out-dated. This is another reason synchronous dataflow graph (SDF) are not suitable to model our system since the nodes or actors in a SDF wait on certain tokens to start executing. For such, we choose to rely on a generic DFG to model the dependence graph.

On these grounds, we model our system's dependence graph with a dataflow graph $G = (V, E)$ where the nodes $V = \{t_1, t_2, ..., t_n\}$ represent the components that make up the system and $E = \{e_{11}, e_{12}, ..., e_{kp}\}$, the set of edges e_{ij} that link components t_i and t_j. It is worth noting that self-loops, defined as data generated by a component and read by the same component, are ignored since they have no effect on the execution order of the components.

The common representation of systems found in literature is that of a *task graph*, onto which we aim to transform the above defined DFG. A task graph is a directed acyclic graph, $\tilde{G} = (\tilde{V}, \tilde{E})$, in which the nodes $\tilde{V} = \{\tau_1, \tau_2, ..., \tau_n\}$ represent a job instance of the component V. A job τ_i is characterized as a tuple $\{Id_i, C_i, d_i, T_i\}$ defined as the identifier, execution time, absolute deadline, and period respectively. The edges $\tilde{E}$ represent the precedence between the jobs and describes the constraint on the execution order of the jobs.

As can be seen from Figure 5.1a, component communication does not imply identical periodicity. In other words, two components communicating with each other does not necessarily translate to them executing at the same rate. This characteristic is important to identify especially when certain schedulers are considered. With scheduler policies that rely on task deadlines and periods to assign priorities, multiple components are assigned the same priority in execution and the number of tasks with the same priority becomes large as the system grows in size. If we take as an example a taskset with three tasks (t_1, t_2, t_3) with periods

$\{16, 32, 16\}$ respectively, and we schedule them under a Rate Monotonic scheduler, the RMS algorithm will assign priorities $P = 2$ for task t_2 and $P = 1$ for both tasks t_1 and t_3. The need to assign execution orders for tasks with similar priorities becomes necessary when scheduling a dependent set of tasks in which precedence is a constraint.

Furthermore, we argue that analyzing the overall system to break all cycles is unnecessarily time-consuming since data produced by components with shorter periods can be consumed at a later time by the components with larger periods. Component `Spoilers` in Figure 5.1a for example might produce data every 10 (ms). However, `Hydraulic` only consumes the data once every 40 (ms) making it irrelevant whether `Hydraulic` is consuming the data generated in the first period of `Spoilers` or the last.

5.4 Graph Transformation Through Data Criticality

The graph transformation refers to breaking the cycles present within the dependence graph to be able to schedule the resulting task graph. In the context of scheduling soft real-time systems, cycle breaking involves the decision of which components can tolerate delay. Our approach relies on the definition of a new parameter that characterizes the data exchanged between the components that we label *data criticality*.

5.4.1 Error Propagation Approach (EPA)

Data criticality comes from the understanding that some data generated by some components are more important than other data. In here, the importance of data expresses how affected the system would be if the data were erroneously computed. Since our problem is a scheduling issue, in which the system components are scheduled in a certain order within one time period, breaking cycles by removing edges in the graph does not involve loss of data but rather results in one of the data generated to be "out-dated". This is to say that when edge e_{ij} is removed, the k^{th} job $\tau_j^{(k)}$ of the component at the tail t_j will use data generated by component t_i in the previous time period, $\tau_i^{(k-1)}$, to complete its inner computations. This can be observed in the example shown in Figure 5.2.

Since the system components are viewed as black boxes and the knowledge of the interactions of input and output variables within a component is unavailable, it is difficult to determine which components can tolerate out-of-date data. For such, we label the data generated by a component that is using outdated information as "faulty data". By modelling this behaviour as an error injection mechanism, we propose a method that computes the criticality of the data based on the propagation of the error within the system graph and that with no

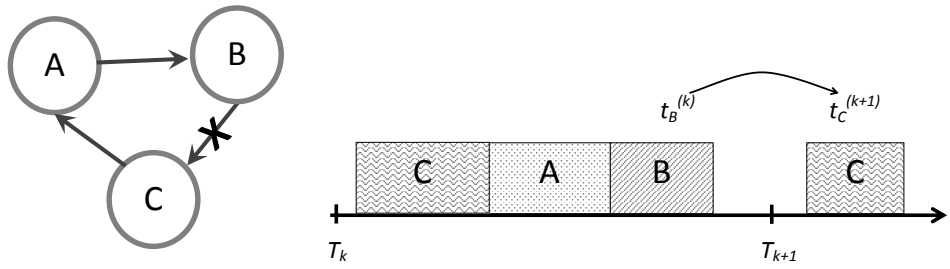


Figure 5.2: The effect of splitting a cycle in a real-time system.

knowledge of the components' functionality. This involves four steps:

1. Extraction of all graph cycles;
2. Calculation of the rate of error propagation within the system graph from every component in the cycles;
3. Assignment of criticality weights to the data flowing inside the cycles;
4. Removal of appropriate edges to break the cycles.

In the following, we detail every step and provide insight into their implementations.

Cycle Extraction

A simple cycle is defined as a path $\Gamma_c = \{t_1, t_2, ..., t_k\}$ in which the head node is the same as the tail node with no repetitive nodes or edges, except for the head and tail nodes. We implemented an algorithm based on the work presented in [110] to extract the cycles present in the graph G. The time complexity of this algorithm is reported as $O((n+e)(c+1))$ for a graph with n nodes, e number of edges and c number of simple cycles.

Error Propagation

By adopting the definition that removing an edge translates to a component using old data and hence generating faulty data, the algorithm follows the propagation of this erroneous data within the graph. Although we can easily follow the dependence between components given a dependence graph, there is no definitive way to track the dependence of a component's outputs to its input variables unless provided by the designer.

To avoid making assumptions about the components, an element of stochasticity is introduced. The behaviour of the erroneous data when consumed by a component can have two behaviours within a component.

– State A in which the error disappears if the faulty data gets overwritten by the component's inner calculations. A simple example of this is the case in which the faulty variable x is initialized if certain conditions are fulfilled.

```
[..]
if (Conditions == True) do:
        x = x_0
end if
z = x - 10
[..]
```

– State B in which the error propagates to other output variables that include the faulty variable x in their computations as a function $f(x)$, which allows the error to propagate from the component to the components of the system that directly depend on it.

```
[..]
y = x^2 + z
if (Conditions == True) do:
        x = x_0
end if
[..]
```

We define A_i as the event of component t_i producing an error in which:

$$A_i = \begin{cases} 1, & \text{if the error is propagated.} \\ 0, & \text{if the error is masked.} \end{cases} \tag{5.1}$$

The problem can be viewed as a set of Bernoulli trials in which every component has the probability of internally propagating an error defined as:

$$Pr(A_i) = \begin{cases} p_i, & \text{if } A_i = 1. \\ q_i, & \text{if } A_i = 0. \end{cases} \tag{5.2}$$

where p_i represents the probability of a component propagating an error from its inputs to its outputs, and $q_i = 1 - p_i$, the probability of masking the error. This is not to be confused with the probability of a component generating or containing a software bug since the components are assumed to be bug-free at integration time.

We define $Pred(t_i)$ and $Succ(t_i)$ as the list of predecessor and successor components of t_i respectively. Assuming that $Pred(t_i) = \{t_j\}$, the observation of an error at the output of t_i depends on t_j

$$Pr(A_i = 1|A_j = 1) = p_i,$$
$$Pr(A_i = 1|A_j = 0) = 0$$

We want to find the probability of a component propagating an error to its successor dependants with the probability of the error having propagated from its predecessor components. In other words, it is the probability of events A_i and A_j having occurred, where $t_j \in Pred(t_i)$ and $t_j \notin Succ(t_i)$.

The unconditional probability of t_i propagating an error is

$$\begin{aligned} Pr(A_i = 1) &= Pr(A_i = 1, A_j = 1) + Pr(A_i = 1, A_j = 0) \\ &= Pr(A_i = 1|A_j = 1)Pr(A_j = 1) + Pr(A_i = 1|A_j = 0)Pr(A_j = 0) \\ &= Pr(A_i = 1|A_j = 1)Pr(A_j = 1) \\ &= p_i Pr(A_j = 1) \end{aligned} \tag{5.3}$$

Events A_j are independent, however, they are not mutually exclusive since components t_j can contain and propagate an error at the same time. For more than one predecessor component, Equation 5.3 can be generalized as

$$Pr(A_i = 1) = Pr(A_i = 1, (\cup_{t_j \in Pred(t_i)} A_j = 1)) \tag{5.4}$$

In the case that t_i has no predecessors, we are dealing with the faulty component at which we are injecting the error and the probability of propagation is $Pr(A_i = 1) = 1$. A weighted graph $\Omega(t_i) = (V, \omega(t_i))$ is built by calculating the probability of error propagation from the faulty component to all its directly and indirectly connected components by assigning weights $\omega_{ij}(t_i)$ to the edges carrying the data. The algorithm is summed up in these steps:

Step 1. Given a cycle Γ, an edge e_{ij} is selected to study the effect of its removal on the system. The component at the tail of the edge is assumed to be the faulty component and the data it generates as erroneous. We calculate the direct error propagation by looking for the component directly connected to the faulty component and assign a

weight $\omega_{ij}(t_i) = 1$ to the edges connecting the faulty component t_i to its successor components $t_j \mid t_j \in Succ(t_i)$.

Step 2. Once the direct error propagation is assigned, we calculate the indirect error propagation by employing Equation 5.4 considering the components t_k that depend on the faulty component through other intermediate components and assign weights $\omega_{jk}(t_i) = Pr(A_j = 1)$ to the edges connecting these components to the intermediate components.

Step 3. We rely on a Breadth First Search (BFS) to look for lower level components that indirectly connect to the faulty component and repeat Step 2 until the last dependent component is reached and appropriate weights are assigned to the connecting edges. The result is a weighted graph $\Omega(t_i)$ with the probabilities of the error propagating from the designated faulty component t_i.

Figure 5.3 represents part of the system of Figure 5.1a and gives a visual example of how Equation 5.4 is used when we track the propagation of error from component t_9 to the rest of the subgraph. We can see that the directly dependent component t_3 will have a probability of $\hat{p}_9 = 1$ to receive faulty data at its input. To calculate the probability of indirect propagation of error from component t_3 to $[t_2, t_5, t_{10}]$, we use Equation 5.3 as

$$\begin{aligned} Pr(A_{t_3} = 1) &= Pr(A_{t_3} = 1 | A_{t_9} = 1) Pr(A_{t_9} = 1) \\ &= p_{t_3} Pr(A_{t_9} = 1) \\ &= p_{t_3} \end{aligned}$$

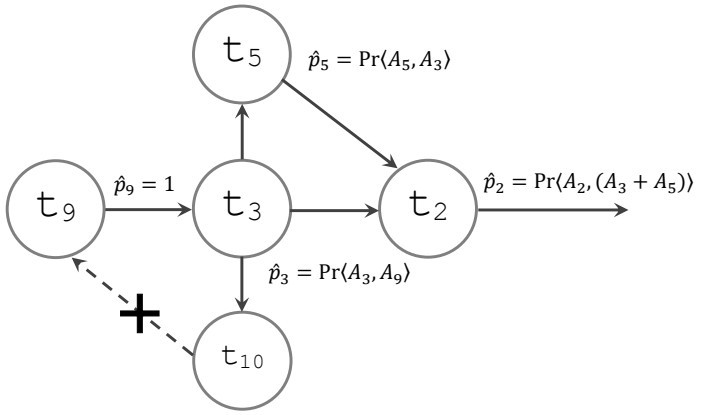


Figure 5.3: Direct and Indirect Error Propagation results in weighted graph $\Omega(t_9)$.

The same is done for the probability of the error propagating from component t_2. Considering that t_2 depends on both t_3 and t_5, the probability of the error propagation $Pr(A_{t_2} = 1)$ is the probability of event A_{t_2} occurring with the probabilities of events A_{t_3} and A_{t_5} having occurred respectively in which case Equation 5.4 will be employed.

Cumulated Error Propagation

Once the error propagation probabilities of a faulty component are calculated, the result is a weighted graph $\Omega(t_i)$ for every edge that belongs to a cycle as in the example graph of Figure 5.3.

The amount of propagation of an error in the system determines the global effect of removing an edge on the system. To quantify this effect, a *cumulated error propagation* (CEP), that expresses the criticality of data, is calculated as

$$CEP(t_x) = \sum_{i,j \in \Omega(t_x)} \omega_{ij}(t_x) \tag{5.5}$$

The example shown in Figure 5.4 illustrates how the error accumulates as it spreads in the system DFG when it transfers from one level to another for different faulty nodes of the same graph. A level is defined as a subset of components which have an equal number of hops (i.e. longest distance) to the root component; t_i. For the example given in figure 5.3, both components t_5 and t_{10} are at level 2 whereas component t_2 is at level 3.

Figure 5.4 shows that the CEP is not affected by how deeply the error spreads through the graph, but rather by the outdegree centrality of the studied component. This is the case since the probability of an error propagating is higher at the first levels and becomes lower

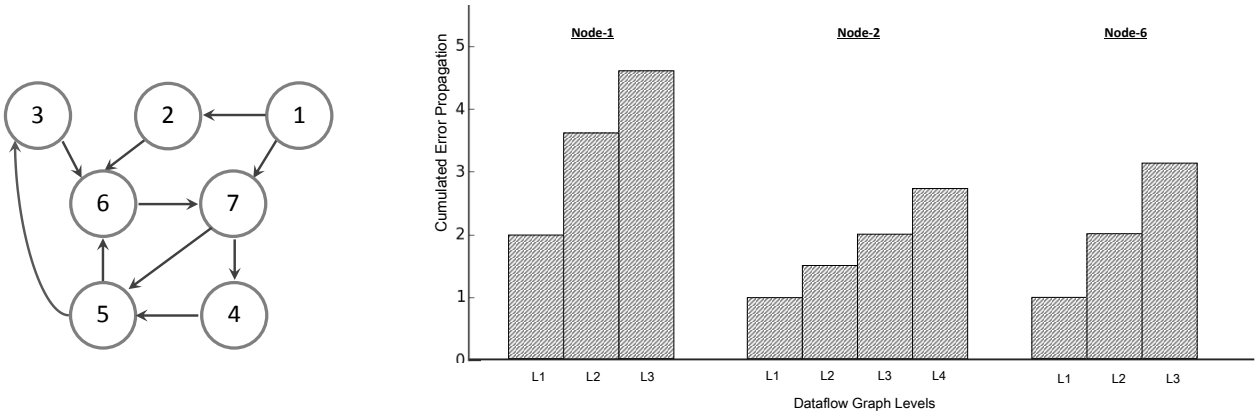


Figure 5.4: Example: CEP for three different nodes

as we go deeper and further from the root faulty component since the faulty data have more chances to be overwritten. That is to say that the denser the levels directly connected to the faulty component, the higher the CEP is going to be.

Given this definition, the component within the cycle with the minimum CEP has the smallest effect on the system, which will decide the edge to be remove to break the cycle.

Minimum Feedback Arc Set with Criticality

Our proposed approach described so far is only concerned with breaking the cycles by removing the edges that carry the least critical data within a cycle. However, in most cases, an edge belongs to more than one cycle. Hence, removing an edge from a cycle might also break other cycles. This means that the number of edges that are to be removed is:

$$\sum_{e_{ij} \in \Upsilon} e_{ij} \leq c \tag{5.6}$$

where c is the number of simple cycles in the graph.

To optimize our approach, we address the problem as a Minimum Feedback Arc (MFA) set problem. This refers to a set of NP-Hard problems that take a non-polynomial time to find the minimum number of edges to remove in order to break all cycles.

Since our main concern is removing edges based on the criticality of data they carry, solving the MFA problem is out of the scope of this paper. Nonetheless, we want to base the decision of removing edges on data criticality while reducing the number of edges that could break all cycles. For this purpose, we introduce the concept of "popularity" among edges. We define the "popularity" of an edge e_{ij} as the frequency of appearance $f(e_{ij})$ of an edge in the cycles Γ_c. Formally,

$$f(e_{ij}) = \sum_{e_{ij} \in \Gamma_c^f} e_{ij} \tag{5.7}$$

Accordingly, we extend the criticality based solution to lower the number of removed edges by first sorting the set of edges that EPA suggested to discard and then remove the most popular edge one by one until all cycles are broken $(12 - 17)$.

Algorithm 1 summarizes the methodology to break cycles within a dependence graph that relies on data criticality and consequently transforms the DFG into a DAG. The algorithm has a time complexity of $O((n + e) \cdot ce)$ for a graph with n components, e number of edges

Algorithm 1: Error Propagation Approach (EPA): Cycle breaking algorithm based on data criticality.

Input: A dependence graph $G = (V, E)$.
Output: A transformation of G to a task graph $\tilde{G}$.

```
1  Load the dependence graph G;
2  Cycles = simple cycles in G;
3  for cycle in Cycles do
4      for edge in cycle do
5          Calculate probability of error propagation Pr(A_i = 1) from component t_x;
6          Assign probability to edge weights ω_ij(t_x) ;
7          Build weighted graph Ω(t_x) with error propagation probabilities;
8          Attribute the Cumulated Error Propagation CEP(t_x) to edge;
9      end
10     Add edge with minimum(CEP) to Temporary Edges list;
11 end
12 Calculate "popularity" of edges in Temporary Edges;
13 Sort Temporary Edges according to popularity;
14 while Cycles do
15     Add Popular edge to the Removed Edges list ϒ;
16     Update Temporary Edges and Cycles lists ;
17 end
18 Remove edges ẽ_ij ∈ ϒ from G̃.
```

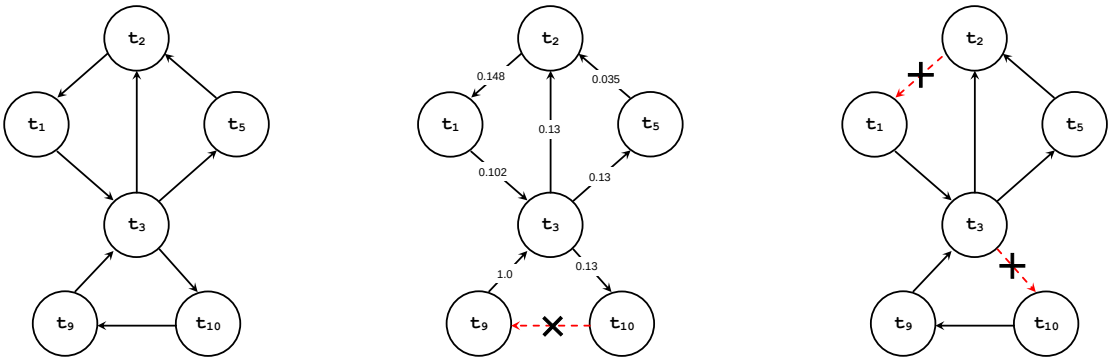
and ce number of total edges in c number of simple cycles.

5.4.2 Motivational Example

In order to demonstrate the EPA approach to break cycles and how the concept of data criticality is employed, we apply EPA to the example of a simple avionic system presented in Section 5.1. Keeping to the same example, we schedule the system under a Rate Monotonic scheduler and hence only consider the subgraph with components executing at a rate $T = 10(ms)$ to reduce the complexity (as explained in Section 5.3).

The dependence dataflow graph in Figure 5.5a represents this subgraph and consists of 6 interconnected components. We can see that the graph has three cycle; $\Gamma_1 = \{t_3, t_9, t_{10}\}$, $\Gamma_2 = \{t_3, t_2, t_1\}$, $\Gamma_3 = \{t_3, t_5, t_2, t_1\}$; that needs to be broken in order to transform the DFG into a DAG. Table 5.1 summarizes the probability $Pr(A_i)$ of every component to propagate an error from its inputs to its outputs.

Applying the (EPA) approach, Figure 5.5b illustrates the spread of the error through the graph when edge $e_{10,9}$ is removed. Considering the problem definition presented in Sec-



(a) Cyclic Dataflow Graph. (b) Error propagation after removing edge $e_{10,9}$. (c) Transformed Directed Acyclic Graph.

Figure 5.5: Dataflow graph transformation of an avionic subsystem with components executing at a rate of $T = 10ms$.

Table 5.1: Probability of a component propagating an error from its inputs to its outputs.

Component	t_1	t_2	t_3	t_5	t_9	t_{10}
$Pr(A_i)$	0.69	0.92	0.13	0.27	0.32	0.52

tion 5.4.1, removing edge $e_{10,9}$ translates to injecting a fault at component t_9. The weight of the edges represent the probability of the error propagating from a component to its dependent components. Thus, the probability of the error spreading from component t_9 to its directly dependent component $\{t_3\}$ is $Pr(A_9 = 1) = 1$. For the sake of illustration, we calculate here the probability of error propagation from component t_5:

$$\begin{aligned} Pr(A_5 = 1) &= Pr(A_5 = 1|A_3 = 1)Pr(A_3 = 1) \\ &= Pr(A_5 = 1|A_3 = 1)Pr(A_3 = 1|A_9 = 1)Pr(A_9 = 1) \\ &= p_5 \times p_3 \times 1 \\ &= 0.035 \end{aligned}$$

The general formula of Equation 5.4 can be illustrated by calculating the error propagation from component t_2 as follows:

$$
\begin{aligned}
Pr(A_2 = 1) &= Pr(A_2 = 1, (A_3 = 1) + (A_5 = 1)) \\
&= Pr(A_2 = 1 | (A_3 = 1) + (A_5 = 1)) Pr((A_3 = 1) + (A_5 = 1)) \\
&= p_2 \times (Pr(A_3 = 1) + Pr(A_5 = 1) - Pr(A_3 = 1) Pr(A_5 = 1)) \\
&= 0.148
\end{aligned}
$$

As can be seen in the weighted graph of Figure 5.5b, the probability of the error transferring to components in the lower levels decreases gradually as the error has more chances of being nullified. This, however, highly depends on the topology of the graph since the probability of a heavily connected component to propagate an error will be relatively greater than its less connected neighbours as is the case with t_2.

Table 5.2: CEP for edges in the overlapping cycles Γ_2 and Γ_3 and non-overlapping cycle Γ_1.

	Γ_1			Γ_2			Γ_3	
Edge	$e_{9,3}$	$e_{3,10}$	$e_{10,9}$	$e_{2,1}$	$e_{3,2}$	$e_{1,3}$	$e_{3,5}$	$e_{5,2}$
CEP	5.344	**1.495**	1.674	**1.514**	1.955	4.876	2.776	2.021

Given the resulting weighted graph of Figure 5.5b, Equation 5.5 is employed to calculate the CEP, which represents the criticality of the data carried by the removed edge. In the same manner, the algorithm will calculate the CEP of the edges constituting the cycles $[\Gamma_1, \Gamma_2, \Gamma_3]$ as summarized in Table 5.2. Considering this, edges $e_{3,10}$ and $e_{2,1}$ produce the smallest CEP which means that removing these edges will have the smallest effect on the overall system. This results in the Directed Acyclic Graph of Figure 5.5c and by extension, the graph of Figure 5.1b.

5.5 Error Propagation Approach: Experimental Evaluation

For the purpose of assessing the efficiency of the EPA methodology, we conducted a set of experiments to compare the approach with two other cycle breaking solutions in terms of system criticality. System criticality refers to the effect of removing a set of edges on the overall system. This is defined as the maximum cumulated error propagation throughout the system as a result of removing a set of edges Υ and is formally defined as,

$$
SysCrit = \max_{e_{ix} \in \Upsilon} CEP(t_x) \tag{5.8}
$$

An industrial avionic system was taken as a reference in the choice of parameters of these experiments. The average number of components that constitute the system as well as the graph structure of this case study were inspiration to generate the set of graphs and scenarios described below.

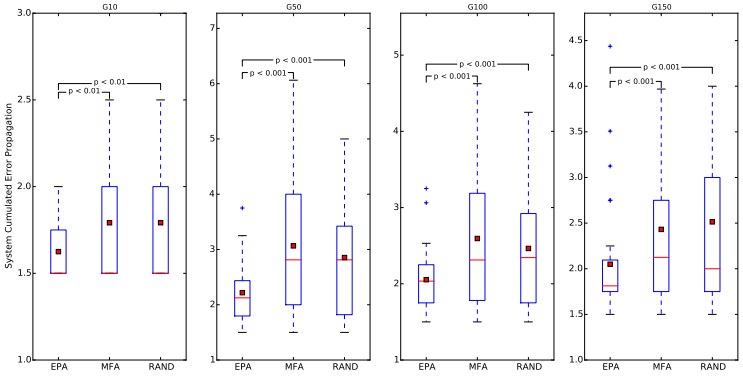
To ascertain that EPA does not operate haphazardly, the first approach is a random algorithm that removes a random edge from a cycle one step at a time until a DAG is obtained. The other approach is a minimum feedback arc set approximate solution. We mentioned before that finding the minimum number of edges to break all cycles in a graph is a NP-Hard problem. We implemented the approach proposed in [111], henceforth labelled MFA, that approximates the number of removed edges to break all simple cycles to the optimal minimum number.

For the purpose of these experiments, the graphs were generated using the `Networkx 1.11` package [112], by taking two characteristics into account: Density and connectivity degree. Density refers to the total number of the nodes making up the graph. Connectivity degree is the ratio between the number of nodes and the total number of edges. The graphs were generated by adding nodes one at a time with an edge in either directions to one previously added node, chosen with a uniform probability.

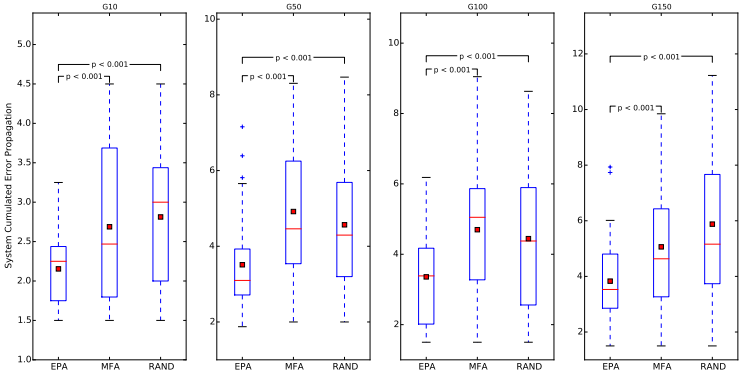
In here, 12 scenarios were considered in which 30 random graphs were generated for every scenario with densities ranging from 10, 50, 100 to 150 nodes and 3 different degrees of connectivity with ratios $[0.10, 0.25, 0.50]$. The generated graphs had c number of cycles in the range $c \in [1, 6000]$. These numbers were inspired by a real case study of a full-mission simulator (FMS) in which the average number of system components ranges from 50 to 120.

Figure 5.6a shows the system maximum CEP for graphs with different densities and connectivity ratio of 0.10. Although the performance of the algorithms are comparable when the density of the graphs are at 10 nodes, we notice that EPA outperforms the other methods in terms of system criticality as the density of the graphs grows. It can be seen that the Minimum Feedback Arc set solution decreases in performance and produces the most critical systems with an average $SysCrit_{MFA} \in [2.5, 3.5]$ as compared to the random and EPA approach with $SysCrit_{EPA} \in [1.8, 2.3]$.

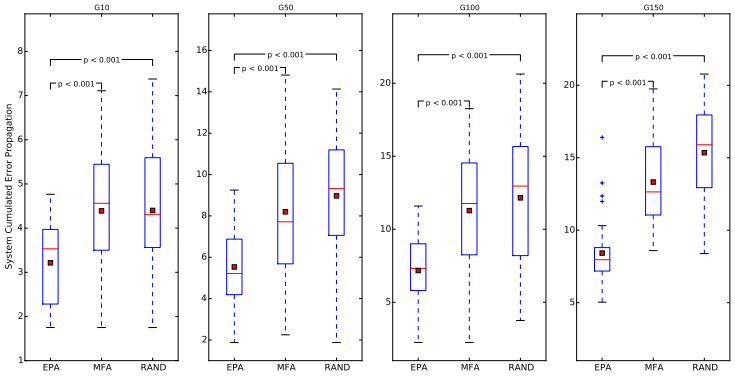
The same set of observations are noted when the connectivity between nodes is increased as shown in Figure 5.6b. We notice that the criticality of the system is much higher than the previous scenario, with an average $SysCrit_{EPA} \in [2.2, 3.8]$, which is expected since increasing connectivity results in a more connected network and the components being more dependent on each other, which increases the chances of an error propagating to a higher number of components.



(a) 1^{st} degree of node connectivity



(b) 2^{nd} degree of node connectivity



(c) 3^{rd} degree of node connectivity

Figure 5.6: System CEP for different node connectivity degrees

Increasing the connectivity of the nodes to a connectivity ratio of 0.50 in Figure 5.6c results in the MFA performing slightly better than the random algorithm with $SysCrit_{MFA} \in [4.3, 13.8]$. However, the same scenario results in a much superior performance from the EPA as compared to the previous scenarios with $SysCrit_{EPA} \in [2.3, 8.4]$.

We employed the Wilcoxon statistical test to confirm whether the results obtained for EPA and MFA had identical distributions. The test yielded a maximum $p\text{-}value = 2.9 \cdot 10^{-3}$ which allows us to reject the null hypothesis H_0 of identical means.

The observations reported so far could be attributed to the fact that the main objective of the MFA solution is to minimize the number of removed edges. It would make sense that the most connected edge will be chosen by MFA to be removed. Yet, EPA will avoid these type of edges since the criticality of central edges will increase because of component dependencies.

Table 5.3 summarizes the number of removed edges for EPA and MFA as the connectivity degree increases. We can see that for a connectivity ratio of 0.50 and high density graphs, EPA discarded double to triple the number of edges removed by MFA. However, the system criticality was still observed to be lower for EPA than MFA as shown in Figure 5.6c, which supports our previous explanation. MFA removes the least number of edges but the most critical as opposed to EPA, which removes a larger number of least critical edges. That being said, for less connected graphs, the number of edges removed for EPA and MFA does not differ significantly with less critical systems in the case of EPA as seen from 5.6b. This could be explained by the lower degree of connectivity as well as the fact that EPA includes a step to minimize the number of removed edges.

5.6 Case studies

5.6.1 Voice-band Data Modem

The first case study represents a voice-band data modem [113] as shown in Figure **??**. This application was selected since it contains cyclic dependencies and is often employed as a benchmark to validate emerging research scheduling techniques for soft real-time systems. The graph of the modem consists of 15 components and has 5 cycles in total. Although the potential schedule start and finish points (`IN`, `OUT`) are obvious, regardless of whether their functionality is known or not, the schedule encounters a cyclic execution once it reaches the component `(Eq)`. For such, we employ EPA to decide which edge(s) to disregard and thus, define the order in which the components will be executed. For the sake of brevity, we only show the results obtained by running the EPA algorithm on one of the cycles. Table 5.4 summarizes the date criticalities (CEP) when studying the edges in the cycle $\Gamma = \{$`Eq,`

Table 5.3: Number of removed edges as connectivity grows.

		G50			G100			G150		
Connectivity	Alg.	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max
1^{st} degree	EPA	1	1	3	1	1	2	1	1	1
	MFA	1	1	1	1	1	2	1	1	1
2^{nd} degree	EPA	1	2	4	1	2	4	1	2	7
	MFA	1	1	3	1	1	3	1	2	5
3^{rd} degree	EPA	1	5	14	1	8	21	3	14	27
	MFA	1	3	6	1	4	6	2	5	9

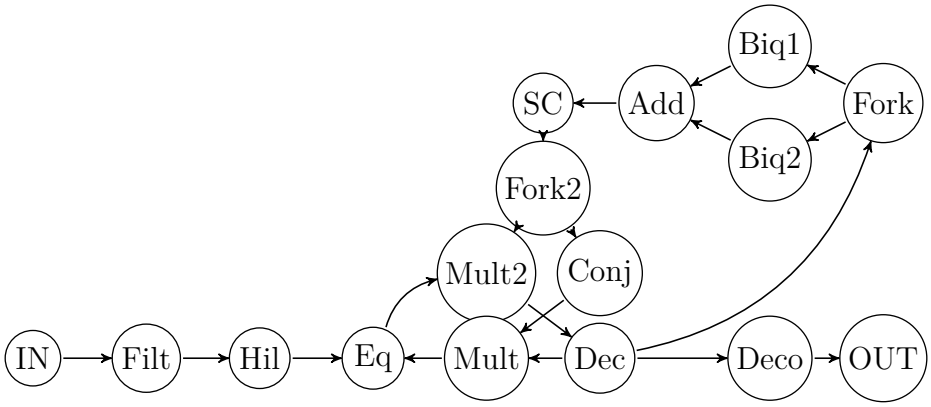


Figure 5.7: The directed graph of a voice-band data modem.

`Mult2, Dec, Mult`}.

From Table 5.4, we can see that in order to have the least critical system, EPA suggests the removal of edge $e_{(Dec,Mult)}$ to break the cycle Γ. Running EPA on the rest of the cycles yielded the set of edges $\{e_{(Mult,Eq)}, e_{(Dec,Fork)}\}$ as the candidate edges to remove to transform the graph in Figure **??** into a DAG. From a functional point of view, the choices made by EPA do not jeopardize the correctness of the system since the component `(Dec)` is the decision module and is supposed to execute last in the loop. This solution agrees with many scheduling solutions found in literature for this benchmark [113, 114].

5.6.2 Full-Mission Simulator

Full Mission simulators are mixed-critical systems comprised of components with hard and soft deadlines. Although our approach focuses on soft real-time systems, in this section, we want to validate the accuracy of the scheduling assignment when EPA is applied.

Our case study involves an industrial FMS scheduled under a modified version of RMS that

Table 5.4: CEP for edges in cycle Γ

Component	Edge	CEP
t_{Eq}	$e_{(Mult,Eq)}$	3.00
t_{Mult}	$e_{(Dec,Mult)}$	2.44
t_{Dec}	$e_{(Mult2,Dec)}$	6.34
t_{Mult2}	$e_{(Eq,Mult2)}$	4.12

consists of two subsystems: Sub_1 and Sub_2 consisting of 9 and 13 components respectively. The two subsystems are interconnected but the components within each subsystem execute with different rates from the components of the other subsystem. This means that the EPA algorithm operates on the two subsystems separately. It is worth noting here that the current schedule with which the industrial FMS is executed was implemented by integration engineers whom take advantage of their prior expertise in the different fields and the knowledge of the components functionality, as well as, trial and error to achieve the current working state of the simulator.

In order to compare the accuracy of the system generated by EPA, we observe the accuracy of the scheduling obtained from the DAG after the transformation of EPA and the current schedule of the FMS that we label the *real schedule*. We define $Accu_{sched}(t_i)$ as the degree to which the EPA scheduling matches the real schedule for every component t_i as follows:

$$Accu_{sched}(t_i) = \frac{Pred_{EPA}(t_i)}{Pred_{Real}(t_i)} \tag{5.9}$$

Where:

$Pred_{Real}(t_i)$ represents the number of components that are scheduled before component t_i and;

$Pred_{EPA}(t_i)$ represents the number of predecessor components that EPA scheduled before component t_i and were scheduled as predecessors in the real schedule as well.

Due to confidentiality agreements, we are unable to include the full details of the experiments conducted in this case study. However, we provide the outcome of the experiments henceforth.

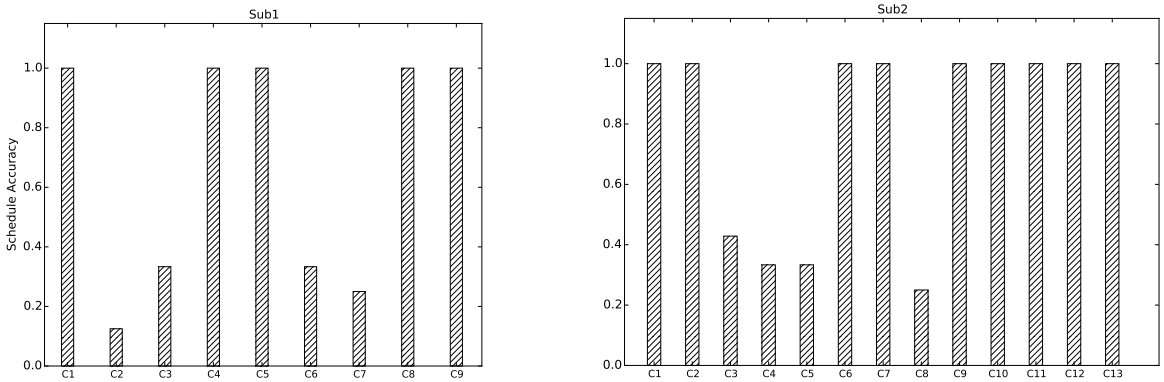
The results obtained are summarized in Figure 5.8 in which the schedule accuracy obtained for both Sub_1 and Sub_2 following the definition in Equation 5.9 are plotted. We can see that although the two schedules are not identical, the accuracy of the scheduled DAG obtained after EPA is relatively close to the real schedule for both subsystems. In this case study, we blindly trusted the schedule provided with the full-mission simulator (the real schedule) to

be the ideal schedule. However, the real schedule sometimes gives higher execution orders to certain components even when there is no precedence requirements to be met. This can occur if other objectives are in play such as load balancing. We argue that the values provided in Figure 5.8 are lower bounds and that the average accuracy of the EPA schedule would increase if these cases were overlooked.

It is a good place to remind the reader that the current schedule is obtained manually after a rigorous trial and error process that involves expertise from different fields. The fact that EPA could result in an average accuracy of 79% without requiring the knowledge of the functionalities of the components is very promising, especially when dealing with such complex systems. The solution becomes more appealing to co-simulation design and early stage integration of components coming from different sources and that is by providing starting configurations that could help accelerate the process and reduce time-to-market.

5.7 Conclusions

Our work was driven by the current state of complex software development in avionic and automotive industries. The lack of software and architecture specifications prompted us to propose an approach that offers access to the system's task graph with no inherent knowledge of the components functionality. In this paper, we presented a methodology that makes it possible to schedule soft real-time tasks after transforming the system's dataflow graph onto a task graph by assigning criticality levels to the data exchanged based on the idea of error propagation. This in turns opens access to DAG-based tools and techniques with limited knowledge about the target system. We demonstrated the efficiency of the algorithm to break cycles based on data criticality which produced the least effect on the system. As a matter of fact, the approach was able to deliver systems with criticality levels $(56 \pm 14)\%$ lower than other cycle breaking algorithms. Since adding new components affects the view of the system and thus increases integration cost, the approach proposed here offers the potential to test a set of configurations which will substantially reduce integration cost and offers automatic solutions to current integration issues.



(a) (b)

Figure 5.8: Schedule accuracy for the components in (a) subsystem Sub_1 and (b) subsystem Sub_2

CHAPTER 6 ARTICLE 4 – TIME IS OF THE ESSENCE: SPREADING INFORMATION AMONG INTERACTING GROUPS

Preface

In this last contribution, we moved our focus towards studying the flow of information as its potential in engineering better systems was becoming more evident. Following the same trend of having a narrow view of the system, we chose to limit the number of assumptions when modeling the propagation of information. The purpose was to explore the efficiency of the model in revealing the interesting patterns and explain emergence of behaviors. We did so through an empirical study of the spreading of a rumour case which revealed the main contributors to the propagation and the reason behind the occurrence of certain events. These observations lead to the development of a framework to strengthen data transfer in wireless sensor networks. The work here transfers the acquired knowledge to developing a probabilistic timing analysis technique based on the developed model to estimate a probabilistic worst time for the information to propagate in a network. This is great import with the surge of multi-agent systems such as IoT connected devices and robot swarms which are starting to be employed in critical situation calling for stricter timing requirements.

Authors I. Hafnaoui, G. Nicolescu and G. Beltrame
Submitted to: Nature Scientific Reports.

ABSTRACT

Animal behavior is greatly influenced by interaction between peers as well as with the environment. Understanding the flow of information between individuals can help decipher their behavior. This applies to both the microscopic and macroscopic levels, from cellular communication to coordinated actions by humans. The aim of this work is to provide a simple but sufficient model of information propagation to learn from natural coordinated behavior, and apply this knowledge to engineered systems. We develop a probabilistic model to infer the information propagation in a network of communicating agents with different degrees of interaction affinity. Another focus of the work is estimating the time needed to reach

an agreement between all agents. We experiment using swarms of robots to emulate the communication of biological and social media groups for which we are able to provide upper bounds for the time needed to reach a global consensus, as well as to identify individuals that are responsible for slow convergence.

6.1 Introduction

Behaviors in a large group of individuals that change the state of a system are referred to as collective behaviors. Although the term was first used by sociologists [115, 14] to refer to the emergence of new social structures as a reaction to certain events, it was later extended to cover behaviors observed in the animal kingdom such as in schools of fish [116], flocks of birds [117], and ant colonies [118]. There is a general agreement among sociologists and biologists as to the conditions that encourage the emergence of collective behavior. The most prominent ones are conflict, ambiguous policies [119], or change in the normative order [120, 121]. The detection of a new food source, for instance, is observed to trigger behaviors ranging from establishing optimal routes by ants [118] to nest migrations of bee swarms [122]. When an intruder is sensed, hyenas use unique whoops, specific to every individual, to reach a consensus on who belongs to the clan and then use the whoops to coordinate the hunt against the intruder [123].

Studying these intricate systems has taken one of two main directions: a macroscopic view, which focuses on the group-level behavior, like the study of the group morphology [124, 125]; or a microscopic view that aims at studying the interactions between individuals which give rise to the behaviors observed in aggregations [126]. Generally, a collective behavior does not emerge from the state of the individual entities in a group, whether that be emotions of uncertainty, imagery or strain in the natural order. It is rather the result of the information shared between the individuals in a communication network. A good example of this behavior is the spreading of rumors in social networks. A previous study [127] showed that social network platforms are increasingly becoming the go-to media to share information among directly and indirectly affected individuals in case of a crisis. Even officials, such as emergency responders, are becoming reliant on these media to gather and communicate information [128]. As such, the study of how information spreads, rumors in this case, becomes necessary to stave off potential emergence of chaotic social behaviors. At a microscopic level, the brain can be likened to the systems mentioned so far in that neurons are equipped with neurotransmitters that propagate signals through a neural network to give rise to a given function. Similarly, scientists have recorded collective behaviors in cancer cells similar to those observed in animal groups in which patterns of collective alignment is observed to generate collective cell migra-

tions, recognized to be at the crux of tumor invasions [129, 130, 131]. All of these systems can be abstracted and represented as networks of interacting agents (animals, users, cells, robots) propagating some kind of information (visual queues, pheromones, tweets, chemical signals, messages) for the purpose of changing the global behavior of the group.

Luckily, information can be quantified, its flow measured and its representation bounded. The limits as to the way information is described, processed or delivered is dictated by the physics of the system. One model to represent the communication in a network is the simple-to-define proximity network [132]. Here, the assumption is that individuals in close proximity interact with each other. However, in reality, the reception of information is hindered by various conditions such as a noisy environment, the affinity of an individual to cooperation, etc. On top of this, assuming that a dependence relationship exists only among the individuals in close proximity and is therefore at the epicentre of the the emergence of collective behaviors is rather naive. In an effort to move past this simple model, we introduce a stochastic element to the interactions between individuals in their range of communication. The next section defines the characteristics of this probabilistic variable and alludes at the physical elements in nature that can be modelled with it.

Our contribution is two-folds. First, we aim to probabilistically model the propagation of information in a network of interacting agents. Many works have proposed complex models to represent the propagation of information, especially infection spreading [17, 18, 133, 56] with a number of parameters and settings. These models, for instance, are built on the assumption that the infection rate, the state of the individual, time and age of infection, etc. are known which might actually be difficult to acquire in a real case-study. The work proposed in [18] goes as far as to require a pedestrian model to accurately estimate the infection transmission in air travel. In addition, most proposed propagation models rely on scenario-specific parameters such as an Susceptible-Infected-Recovered (SIR) model in studying infection spreading which cannot be used to study the propagation of signals in animal groups for instance. The strength of our model lays in abstracting the quantity being propagated to a piece of information (visual queues, chemical signals, messages) with the likelihood of transmission as an attribute (line of sight, infection probability, influence of users). This eliminates the dependence of the model on the scenario being studied and renders it applicable to multiple domains of study. In here, we model the information propagation by striping away these details down to a fewer number of assumptions ; namely (1) a static or slowly changing network, (2) the propagation of a single piece of information and (3) information transmission probability of a node. The purpose here is to show that this simple model is enough to explain the emergence and occurrence of certain events. Behaviors like synchronized flashing exhibited by *photonius carolinus* fireflies and the tendency of certain

fireflies to defect from the group is one interesting scenario to study [134]. In that regard, the flashing lights are construed as information being sent to the rest of the group. The model is then a tool to infer the influence of these defectors on others out of their line of sight by studying the propagation of information between individuals in different regions. As a sub-contribution, we explore the validity of centrality as a contributor to information propagation. Oftentimes, centrality has been a key parameter to techniques that dealt with detecting propagation sources and selecting influential nodes in the network [135, 136, 137]. In this work, we reveal the drawbacks of relying on centrality and propose a metric based on conviction and influence probability to boost the propagation of information.

Second, armed with this model, considering that one of the incentives to studying collective behavior in nature is to gather the knowledge to engineer new systems (e.g. optimized transportation routes, robot swarms as emergency responders), we analyse the timing characteristics of information propagation and the ways it could lead to new technologies. To illustrate this, we study leader-following consensus, common in decision-making problems within groups of interacting individuals, in which the purpose is for the group to reach and agree on the opinion of a leader. This can be modelled as an information propagation paradigm in which an opinion is an information for which the convergence means the agents in the network agreeing to that argument. This has been observed to be a feature that groups, both animal and human, strive for to make decisions and establish certainty over a choice of action [61]. When it comes to consensus, the research is focused on developing controllers that are more resilient and those that guarantee convergence. To our knowledge, little has been dedicated to explore the convergence times towards information propagation. The work of Başar et al. [54] defines the expected convergence rate of Quantized Metropolis consensus. The assumption though is that the graph remains connected in every sequence and that the transmission occurs to a single node per step with a uniform probability. A noisy environment increases the chances of information loss, and inter-individual conflicts might arise, especially within heterogeneous groups. All this increases the time needed to reach a consensus. This has been considered by Cheng et al. [62] in which the effect of noise-attenuation gain was explored in leader-following consensus to define a bound to the convergence rate. The results were limited to gains of a certain class of functions. From the field of evolutionary graph theory, the authors [55] define the exact fixation probability and time of a Moran process for arbitrary small graphs. Most of these works rely on Markov chains to model the information propagation which renders the states intractable as the network grows in size. Our aim is to provide a probabilistic estimate of information propagation time which is of practical use to real-time modern systems, especially those with hard timing constraints. In this work, we integrate the probabilistic model of information propagation

with a timing analysis technique to estimate a probabilistic worst case convergence time (pWCCT) for information propagation.

6.2 Information from an Observation to a Conviction

The communication among individuals that share feelings of uncertainty and are prone to share their state and the state of the environment around them to their neighbors have been observed to be at the root of the emergence of collective behaviors [14]. The time it takes for a behavior to form and spread in a group relies heavily on the efficiency of the communication medium as well as the affinity of the individuals to receive and share information; in other words, their affinity to cooperation. The intuition here is to model the direct interaction between an individual and its neighbors and the indirect interaction with the rest of the group in a probabilistic manner to project this affinity and information loss.

We want to define the probability of an individual to receive information if broadcast by a different individual in the network, and not necessarily by its neighbors. Suppose that individual A sends an information that will reach individual B with probability $p_{ab} = Pr(B|A)$. We define the *information conviction* as the probability of an individual to hold the information broadcast by another individual in the network. The conviction of B to have the information is $Pr(A) \cdot p_{ab}$. This is true provided B has a single source of information, A in this case. For such, we need to define what it means to share information, and how to condition the information propagation on other sources of information. The example in Figure 6.1 shows that X has multiple of these sources. We model this as a message passing system in which a message transmits the confidence of a node to pass on the information to a particular neighbor. It can be viewed as an individual sending a message to broadcast their ability to propagate the information; in other words, their conviction of holding the information and the confidence of which they are to send it to a neighbor given what its other neighbors are saying through the received messages. Hence, we recognize two types of messages:

- *Received messages* represent the messages π received by a node. They indicate the opinion of the source node of how likely the information is to reach a certain node from its neighboring nodes;
- *Shared messages* define the messages λ that a node share with its neighbors. Message λ_{XY_1} for instance describes the likelihood of Y_1 getting the information from a specific source X and no other.

The algorithm is local in that a node relies only on the opinion of its neighbors from the

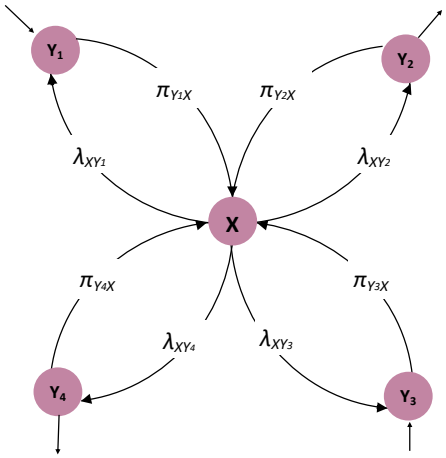


Figure 6.1: Information propagation from and to X through passing messages of type λ between X and its neighbors $Y_i \in \mathcal{N}(X)$ and π between $\mathcal{N}(X)$ and X.

received messages to build its conviction of an information reaching it given that the information was observed and broadcast from remote parts of the network. That being said, the interaction graph is defined as a directed graph that can contain cycles (See *Methods*). Initially, the only observations are the messages shared by the broadcasting source whereas all other messages are initialized to zero. To reach the information conviction of every individual considering direct and indirect interactions, messages are updated iteratively from previous observations of the information propagation in the network. We direct the reader to the *Methods* section for a detailed description of this process.

For illustrative purposes, a homogeneous network of 50 individuals is depicted in Figure 6.2.A in which the information has been set to have equal chances of reaching a neighboring node or getting lost in the process from any node in the network. Iterating over the procedure described by Algorithm 2 (See *Methods*) produces the results in Figure 6.2 in which we can observe the accumulation of the belief of the reception of an information broadcast at node ν_0 throughout different iterations. Once the algorithm converges, the conviction of every individual to hold the information is as exhibited in Figure 6.2.D. The neighbours of the broadcasting source are observed to have a high conviction that they hold the information, which is expected. Surprisingly though, we notice that individuals such as $\{\nu_{14}, \nu_{16}, \nu_{23}\}$ far from the source and with a degree of separation higher than two nodes, in different clusters altogether for that matter, have quite a high information conviction. This tells us that the propagation of information in a network of interacting individuals might not necessarily depend entirely on the distance to the source of the information and line of sight. This leads us to question whether centrality is the reason behind these observations. This notion is further studied in later sections where we show its validity and determine the circumstances

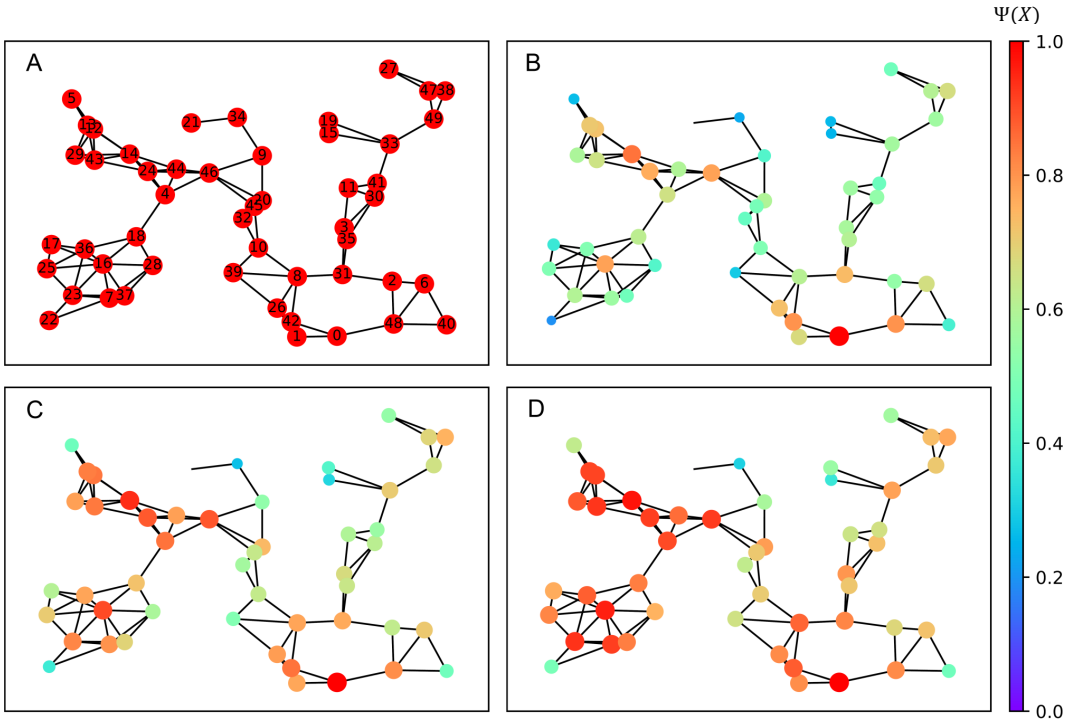


Figure 6.2: The accumulation of information conviction for the network in (A) for iterations (B) $t = 1$ (C) $t = 3$ (D) $t = 6$. We can see that as we observe new states, the conviction of certain nodes grow larger than others. This is observed to not be entirely dependent on the distance to the broadcasting source where some nodes in different regions of the network exhibit higher conviction than immediate neighbours.

under which it no longer holds.

6.3 Probabilistic Worst Case Convergence Time

The transfer of knowledge acquired from studying animal group behaviors to artificial systems has seen a surge of interest in the research community as of late. In light of this, there is a need to study certain characteristics that are inherent to these modern systems; which might, in turn, aid in understanding some behaviors in animals and humans alike that are still unpredictable. One of these characteristics is *timing performance*. In particular, this paper aims at providing a probabilistic estimate of the worst case time for a group to reach a consensus over a piece of information. We develop a methodology to obtain the exceedance probability curve (or Complementary Cumulative Distribution Function) which describes the probability that the convergence time will exceed a certain threshold. We refer to this as the probabilistic worst-case convergence time (pWCCT). This is of a particular interest to a

system designer whom might have strict timing restrictions: for example, the need to ensure that the time for a robot swarm to autonomously agree on a task assignment doesn't exceed a certain deadline.

For every individual ν_i, we define $I_{\nu_i}(t)$ as a probability mass distribution (PMD) which describes the probability of the individual ν_i to receive the information from its neighbors at time t. The intuition is that, for an individual to receive the information at time $t = T$, it suggests that its neighbors that hold the information have failed to transfer it at $t < T$. The PMDs I_{ν_i} represent the timing behaviour of corresponding individuals ν_i. To estimate the timing for a consensus to be reached, we need to look at the timing behavior of the network. Two elements are to be considered: (i) how to translate the individual timing behaviors into a group timing, namely a probabilistic worst case convergence time (pWCCT); and (ii) what possible structures to study to achieve a bound on the pWCCT.

In Figure 6.3, information is propagated to A and B with their corresponding PMDs. To determine the timing of this information flow, we need to enumerate the times at which both A and B have the information, and define the probability to reach every state. In the figure, every arrow represents one of these cases. Convolving the two distributions turns out to produce a probability distribution that describes these possible cases. To estimate the exact timing of the information propagation, one needs to consider all possible paths the information might take to propagate from an individual to the rest of the network. The complexity of this grows exponentially as the size of the network increases. However, since our main concern lays in estimating the worst case convergence time, it is sufficient to consider the longest path in the graph of the network. This is similar to picking one node at a time to

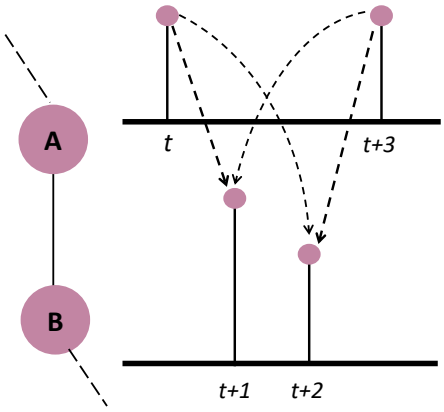


Figure 6.3: PMDs of A and B in which the arrows represent every possible time when both A and B have the information. The arrow linking times t and $t+1$ defines the case in which A recieves the information at time t but fails to propogate to B which recieves it at time $t+1$ instead.

receive the information from the rest of the network at every time-step. As one can expect, this consideration results in an upper bound of convergence time that is quite pessimistic. In reality, individuals in a group are more likely to transmit information to multiple individuals at the same time. In order to tighten this estimation of the probabilistic worst convergence time, instead of considering the longest path that spans all nodes, we look for a spanning tree with minimum branching. Among other assumptions (detailed in *Methods*), we have to assume that the information propagates sequentially to all the individuals, as the inclusion of branches hinders the use of convolution. The way to combine the PMDs of the nodes in the branches with the rest of the spanning tree is by defining a merge operator (described more thoroughly in *Methods*). This takes into account the possibility that certain individuals receive the information at the same time, akin to a parallel process.

6.4 Results

As has been mentioned previously, collective behavior is the product of sharing some kind of information among a group of individuals. Our work spans any system that can be represented as a network of agents propagating an information. In what follows, we illustrate the efficiency of our framework in a few examples of information propagation, namely rumour spreading and consensus reaching in a robot swarm. In this section, we showcase the model presented in this paper as (1) a way to study behaviors and understand the reasons certain events progress in the manner that they do; and (2) as a practical solution to issues encountered in engineered collective systems.

6.4.1 A Rumour and its Counter-Rumour

We study the case of rumour spreading as an example of information propagation in social media. We analysed 7 major news events from the PHEME rumour dataset [138] that spans 297 twitter conversations discussing rumours. We focus on one of the rumours that spread during the Sydney hostage situation of 2014. Figure 6.4 shows the progression of the rumour that claims that hostages were held by terrorists associated with ISIS. The rumour started by the following tweet:

> `@User_zero: SYDNEY SIEGE: Gunman forces hostages to hold up ISIS flag in window. [2014-12-14 23:27:39]`

We observe the tweets that counter-attacked the false rumour and their spreading throughout the network. We notice an explosion of tweets denying the rumour at around `23:45` after a tweet published by a user that we refer to as Bob (For confidentiality purposes):

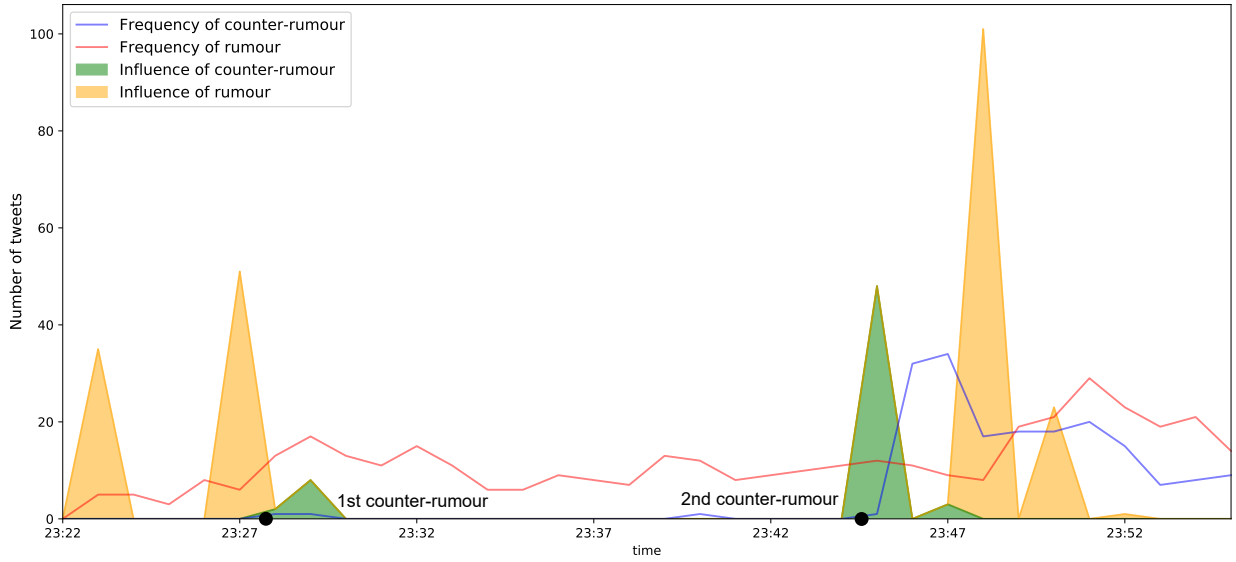


Figure 6.4: Progression of the rumour and the counter-rumours in terms of time which reveals two main attempts at correcting the false claim with differing reactions.

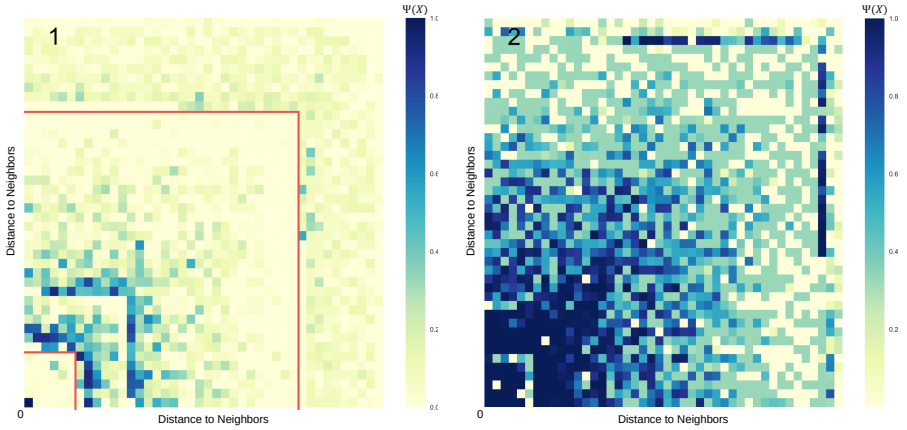
> `@Bob: Flag in window of Sydney Lindt cafe not an ISIS flag. Reads: 'There is no God but Allah and Muhammad is the messenger of God'. [23:45:51]`

However, this was not the first attempt at correcting the false rumour. A tweet previously published by a different user (Alice) revised the claim with visual evidence:

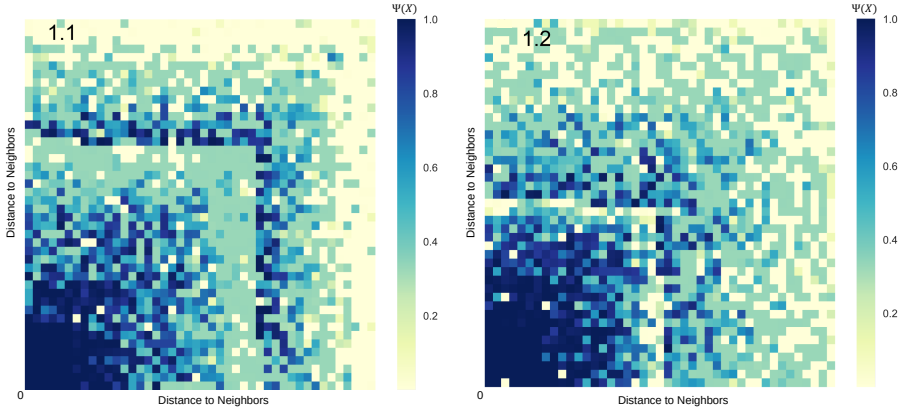
> `@Alice: These not the same. 1st Shahadah flag, 2nd is specifically claimed by IS(ISIS). [2014-12-14 23:29:26]`

In an endeavour to reach an understanding as to why the first counter-tweet by Alice didn't have much of an impact on correcting the rumour whereas the tweet by Bob did, we study the spreading of the information as modelled in previous sections and examine interesting patterns. The procedure to build a group interaction model is fully described in the *Methods* Section, which is then used to model the information propagation for two different scenarios to obtain the conviction of every user in the network to have the information. The first scenario represents the propagation of the information in the network from Alice and the second scenario sees the information spreading from Bob.

Figures 6.5a illustrate the conviction of receiving the information by every individual in the network if the information was propagated initially by Alice and Bob respectively, based on their distance from the source. The heat map generated for Bob shows a wider outreach and propagation of the information to the other users in the network as compared to Alice. This in part explains the reason behind the progression of information observed in Figure 6.4. We



(a)



(b)

Figure 6.5: (a) Heat maps illustrating the probability of the information spreading from (1) Alice and (2) Bob starting at the bottom-left corners to their neighbors. (b) Heat maps for critical users in the intermediate neighborhood of User Alice that exhibit higher probabilitties of information propagation than Alice.

also explore how this analysis could be exploited to prevent rumour spread. Figure 6.5a.1 can be divided in three sections: the first is Alice propagating the information but failing to reach its immediate followers; the second section can be observed to have a sudden darkening area which indicates a user with a great conviction of receiving the information from Alice and a higher influence on their immediate followers and indirect relationships; and a last section, two-thirds through the network, that encounters another user able to further propagate the information through the rest of the network.

Analyzing the spreading of information starting at these critical users results in the heat-maps of Figure 6.5b. Both maps demonstrate a wider outreach then that observed for Alice. These users exhibit high conviction as well as high influence on their direct and indirect

neighbours. This could be an incentive to rely on the framework to detect critical users that will spread the right information in times of crisis and quickly quench any rumours that might arise unforeseen chaotic behaviors.

6.4.2 Timing and Resilience

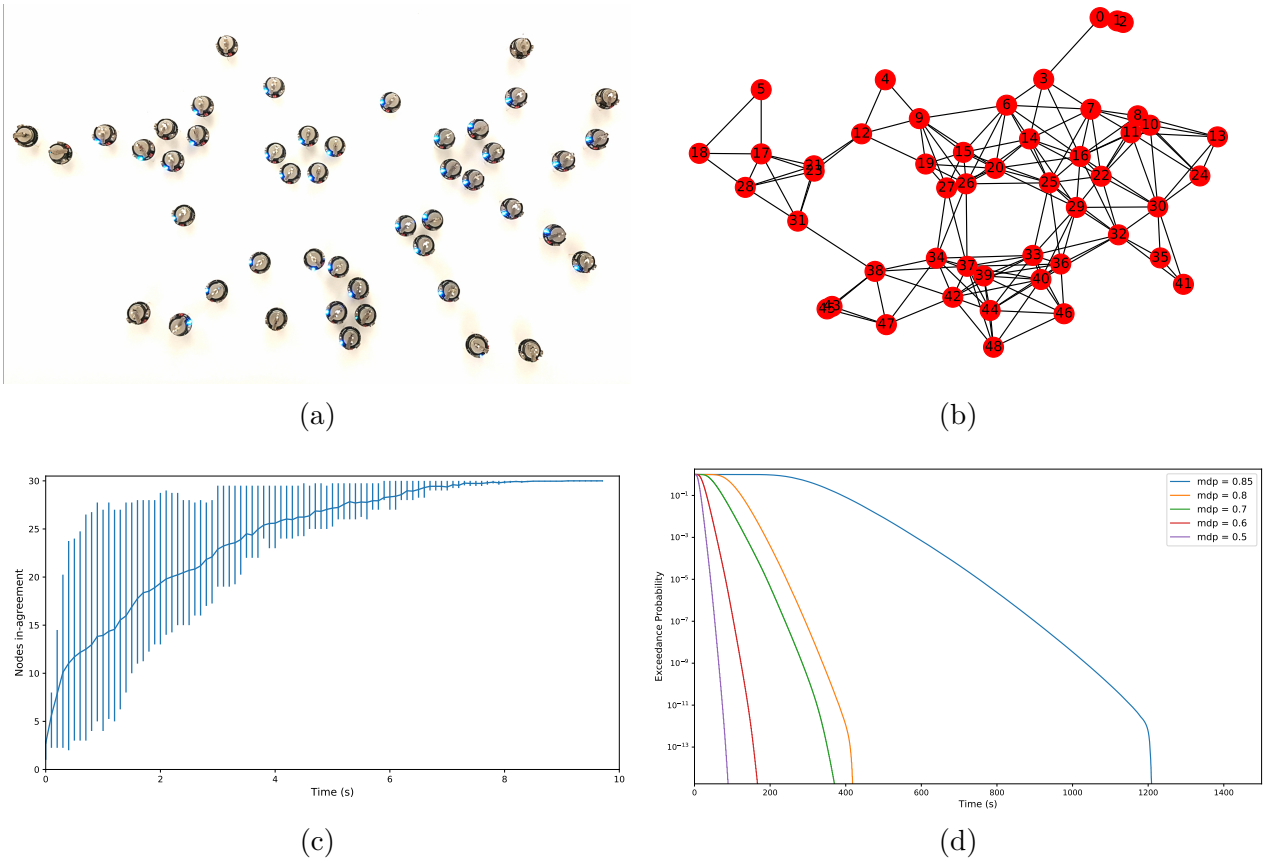
We experimented with our framework as an analysis tool where we estimate timing characteristics and study the resilience of a network to loss of information.

pWCCT and Kilobots

An interesting question to answer is how fast the spreading of infection could occur; either through the body, similar to cancerous cells contaminating neighbouring healthy cells [139] or at the population level, such as the spread of influenza. A recent work tackled this issue to determine the takeover time [58]. The propagation model by Ottino et al. [58] assumed an infected node transmits the infection to a single neighbor at random every time-step from any infected individual in a network. With our framework, we go beyond this simple model: whereas we don't claim to estimate the exact probabilistic distribution of the time for infection spreading, we model the probability of the infection spreading from any infected individual and upper bound the worst takeover time in a probabilistic manner under two assumption: (a) the infection can spread to more than one individual in a single time-step (b) as well as consider that every individual might have a different transmission affinity.

In here, we employed a swarm of small robots to emulate the propagation of information – infection in the example above– throughout a network. To validate the ability of our model to upper bound the worst case convergence time towards a consensus, a set of experiments on a real swarm of robots was performed in which a number of Kilobots [140] form a swarm of different topologies (Figure 6.6a shows the robots in a scale-free topology) and the aim is to study the time to converge to a consensus over a piece of information. Kilobots are simple robots that rely on infra-red communication which renders message transmissions very susceptible to noisy environments. Our motivation to experiment on a real robot swarm is to appraise the performance of our framework when it deals with physical characteristics of the swarm communication such as collision, interference, etc. that are difficult to simulate.

Figure 6.6c plots the results obtained for 30 runs of the same experiment on a scale-free topology. We observe that the distribution of the convergence times has a high variance and that is partially due to the fact that the scale-free topology contains a number of clusters connected by a small number of links. This renders the nodes at these links highly critical



(a) (b)

(c) (d)

Figure 6.6: For the scale-free topology in (a), we show the interaction graph in (b), the time for the swarm of Kilobots to reach complete agreement (c) and its corresponding estimation of pWCCT (d).

and whether they succeed to transmit the message or not highly affects the convergence time. The convergence time recorded for this particular topology was in the range $[3.8, 17.6](s)$.

Looking at what we estimated as the worst case convergence time expected for this topology, we examine Figure 6.6d. We plot the exceedance probability distributions for $mdp \in [0.5, 0.6, 0.7, 0.8, 0.85]$ varying the message drop probability (mdp). Our interest lays in the plots for $mdp_1 = 0.6$ and $mdp_2 = 0.7$ (Refer to *Methods*) which show the worst convergence time estimated at different exceedance probabilities. In other words, $WCCT = 170(s)$ and $WCCT = 360(s)$ for mdp_1 and mdp_2 respectively, exceeded with a very low probability of $Pr = 10e^{-13}$. This upper bounds the time to convergence for the swarm of Kilobots, including extreme cases. The pWCCT estimation based on both convolution and merge presented here defines a *safe* upper-bound to the information propagation in a network that might be pessimistic when only ordinary situations are expected. However, its use is promoted in hard real-time systems that call for strict requirements on their timing behaviors such as robot-aided space exploration or robot emergency responders. This also offers a flexible measure to upper bound this timing characteristic by varying the exceedance probability threshold.

The advantage of our model over what is proposed in literature is its ability to estimate a pWCCT for different topologies, and to be adaptable to different scenarios. In addition to the scale-free topology we presented here, results on other topologies such as a snake-like topology, and a topology where the robots are randomly distributed with obstacles can be observed in Figure 6.9.

A Chain is No Stronger than Its Weakest Link

From our study of rumour propagation, it is clear that preventive measures that rely on robustness analysis of the network are of utmost importance. This extends to non-biological systems, such as wireless sensor networks (WSNs). Designing a network that is entirely fault-tolerant can be too expensive, in terms of time, cost, and expertise. Our model can be a smart solution to select critical nodes in wireless sensor networks (WSNs) to be hardened against faults. Broadly speaking, our model is a tool to analyze a network and detect the weaker individuals that interfere with the propagation of information.

To demonstrate this, we simulate a robot swarm in a random geometric topology in which a selected number of robots have been hardened, i.e. they have been given message drop probabilities $mdp = 0$ to ensure message transfer. In addition, instead of working with a homogeneous swarm, we randomly assign different mdp for every node in the graph. The purpose behind this choice is to observe networks that are heterogeneous in terms of information propagation (e.g. due to individual preferences, different noise levels in the environment,

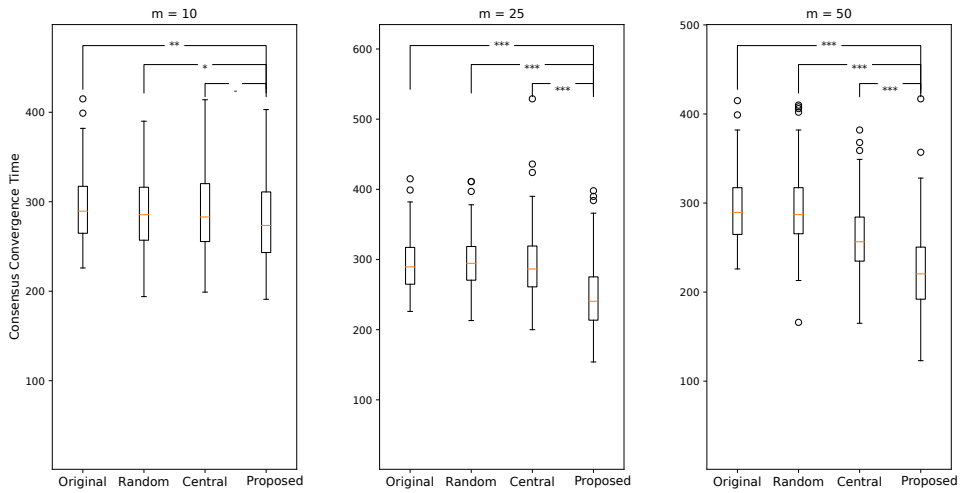


Figure 6.7: pWCCT for a random geometric network in its original form, and different selction methods as the number of selected elements m is varied to (a) $m = 10$, (b) $m = 25$, and (c) $m = 50$.

etc.). We compare against random selection, in which a set of nodes are picked at random. We also mentioned previously that our observations of information propagation might be related to centrality. We study these claims by comparing our method against a centrality-based selection which relies on the topology of the network and picks the nodes that have a high out-degree. We run every scenario 100 times, while we vary the number of selected elements m. This produces the results summarized in Figure 6.7.

While there is no major improvement in terms of shorter convergence time when the number of selected elements is small, we observe a major decrease in the time to reach consensus with $m = 50$ selected elements. The first observation shows that although our method reached global consensus in less time than the original scenario (with no hardened elements), the times reached were statistically similar to a centrality-based approach. In this case, since m is small, both techniques were choosing geometrically similar nodes.

The choice of working with a heterogeneous swarm, at least in terms of *mdp*, is highlighted in the results of Figures 6.7.b and 6.7.c. The idea is that a centrality-based technique selects nodes with high connectivity regardless of their information conviction I_{v_i}. Instead, we locate the weaker links in the network that might hinder the spreading of information throughout the swarm, especially in terms of convergence time. We do so by identifying the nodes that have a high conviction of obtaining the information, but are failing to transmit the information to their neighbours. In other words, we favor a node that we consider a weak link based on its ability to further propagate the information in the network, even if its

connectivity is low. For the sake of brevity, we did not include the results for a *homogeneous* network, but it is interesting to note that our model exhibits similar behavior as centrality-based methods when considering a homogeneous swarm of robots where the *mdp* is constant across the network. This leads us to conclude that centrality is a good measure to select influential nodes. However, by itself, it fails to promote the propagation of information especially in more complex configurations.

6.5 Discussion

Information propagation effects have been categorized by Arif et al. [127], in the case of rumours, into 4 patterns: Giant, Snowball, Fizzle and Babble. The giant and snowball effects, which both exhibit high derivative information propagation by high and low exposure individuals respectively, were of a particular interest since they present patterns that emergency responders look out for to maximize spreadability and stave off the emergence of chaotic behaviors. We observe both of these effects in Figure **??** from Bob and Alice respectively. Although the giant effect could be intuitively interpreted, the snowball effect is an observed fact that is not completely understood. In here, we introduced a simple but sufficient model of information propagation for the aim of studying emergence of behaviors. The purpose here was to explore the effect of stripping the propagation model from scenario-specific details, such as the volume of shared tweets or the rate at which the information is transmitted, etc. This showed that the few assumptions taken in modeling information propagation were enough to discern the patterns that lead to the emergence of the observed event. The study promotes the practice of bottom-up investigation when it comes to modeling information flow and to properly identify and isolate the originator of specific events and behaviors ; as is the case with giant and snowball effects in rumour propagation research. For instance, we show that, although the snowball effect starts with low exposure individuals, it is mainly due to highly influential individuals picking up the information from low exposure sources. Since the model quantifies the information conviction of individuals, it is able to detect easily-influenced individuals which are themselves apt to influence other individuals. Being able to identify the major players in information propagation is of utmost importance especially in crisis situations. We suggest that the framework can readily be used to spread the right information and maximize the likelihood of giant effects. In fact, we exploited this characteristic to analyze artificial networks in which the framework was used to detect the weaker links that hinder information spreading which, as a result, produced more resilient systems. The framework was extended with a timing analysis technique to study information consensus in a group. We showed that despite the few assumptions taken to model the information

propagation, which neither fully express the intricacies of the diffusion of information nor the environment, it is able to provide a probabilistic measure of the worst convergence time towards a consensus. Future work could see the framework extended to uncover the history of current events, such as determining the time and place for the origin of an infection. Admittedly, the framework presented could model a single information with multiple sources. However, future studies could see an extension of the framework to handle multiple, possibly conflicting, information in the network.

6.6 Methods

6.6.1 Group interaction model

The interaction between the group individuals is represented by a directed graph $G = (\mathcal{V}, E)$ in which the vertices $\mathcal{V} = \{\nu_i | i \in (0, n]\}$ depicts the n individuals in a group and $E = \{e_{ij} | (i, j) \in (0, n]\}$ the information flow from ν_i to ν_j. The graph can have cyclic interactions where the individuals exchange information in both directions, which makes the existence of edges e_{ij} and e_{ji} possible. This is observed in a pod of bottlenose dolphins where whistles are exchanged to identify whether a dolphin belongs to a certain group [141]. Since we want to study a group of individuals in noisy environments that are apt to not cooperate, a probability of the information propagating from ν_i to ν_j is defined as $p_{ij} = Pr(\nu_j | \nu_i)$ and is assigned to every pair (ν_i, ν_j).

6.6.2 Conviction Through Message Passing

The message-passing framework implements the idea that a node builds its conviction on an information reaching it, given the information was observed at one or multiple sources, by "listening" to the opinion of its neighbors about their own observations. This happens through an exchange of messages loosely based on the message passing algorithm described in [142]. In other words, the messages are a way to virtually strengthen a node's belief that it will hold a piece of information. This is done by observing the likelihood of its neighborhood to bring the information to it. The observations are modelled through the received messages. The node, by updating its conviction, shares a message to broadcast its conviction to hold and transmit the information. The existence of cycles in the graph imposes a recursive process for a node to iteratively reinforce its opinion until all messages converge.

The example in Figure 6.1 shows that X has multiple neighbors in which we recognize two types of messages; *Received messages* and *Shared messages*. The received messages are built on the direct relationship between a node X and its neighbors $\mathcal{N}(X)$ and the indirect influence

from its second degree neighbors $\mathcal{N}(\mathcal{N}(X))$. They are interpreted as a node receiving the opinion of its neighbors on the likelihood that they will transmit the information given the state of the rest of the network and are defined as:

$$\pi_{Y_iX}^{(k+1)} = \frac{1}{|\mathcal{N}(Y_i)|} Pr(X|Y_i) \sum_{Z \in \mathcal{N}(Y_i)} \lambda_{ZY_i}^{(k)} \tag{6.1}$$

Shared messages are of a particular interest since they can be decomposed to represent how much influence a node has on its individual neighbors. Message λ_{XY_1} for instance describes the level of conviction that Y_1 will get the information from a specific source X and no other. They are defined as:

$$\lambda_{XY_i}^{(k+1)} = \frac{1}{|\mathcal{N}(X)| - 1} \sum_{\substack{Y_j \in \mathcal{N}(X) \\ j \neq i}} \pi_{Y_jX}^{(k)} Pr(Y_i|X) \tag{6.2}$$

The summation excludes the message π_{Y_iX} received from Y_i since the shared message λ_{XY_i} represents the degree to which Y_i is convinced that the information is coming from X, which eliminates the possibility that π_{Y_iX} will be holding the information.

We define $\Psi(X)$ as the conviction of individual X that it holds the information observed to have spread from a particular node in the network and is defined as

$$\Psi(X) = \frac{1}{|\mathcal{N}(X)|} \sum_{Y_i \in \mathcal{N}(X)} \pi_{Y_iX} \tag{6.3}$$

Algorithm 2: Building conviction through message passing.

Input: An interaction graph $G = (\mathcal{V}, E)$. Nodes with observed information $\mathrm{O}(\mathcal{V})$.
Output: Information convictions $\Psi(\nu)$.

Instantiate messages ;
    $\pi_{ij}^{(0)} = 0$;
    $\lambda_{ij}^{(0)} = \begin{cases} 1 & \text{for } i \in \mathrm{O}(\mathcal{V}) \\ 0 & \text{for } i \notin \mathrm{O}(\mathcal{V}). \end{cases}$;
for *Node ν in traversal of G* **do**
    update received messages $\pi_{\eta\nu} \mid \eta \in \mathcal{N}(\nu)$ (Equ. 6.1);
    update shared messages $\lambda_{\nu\eta} \mid \eta \in \mathcal{N}(\nu)$ (Equ. 6.2);
end
Repeat for-loop until messages state converges;
Calculate information conviction $\Psi(\nu)$ (Equ. 6.3) for $\nu \in \mathcal{V}$.

Given the interaction graph defined above and an observation of the information at one or multiple broadcasting sources, the way to properly build the conviction of the other individuals in the network is by traversing the graph and gradually updating the state of the messages as information is observed. We do this in an iterative process that is summarized in Algorithm 2. Intuitively speaking, since the information is only observed at the sources, the messages are initialized to zero since they depict the conviction of an individual having the information, except for the messages shared by the broadcasting nodes. The state of the messages is then updated as dictated by Equations 6.1 and 6.2 by following the flow of the information and repeating the process until convergence. We are able then to define an information conviction for every node in the graph.

6.6.3 Probabilistic Worst Case Convergence Time

The multiple outbreaks of Spruce Budworms that ravaged north-American forests almost every decade of the first half of the 20th century [143] is a scenario of infection propagation that is still being studied extensively. Having a tool to estimate the time for an infection to spread in a community could lead to better prevention methods and open doors to understanding the propagation patterns. The probabilistic model defined so far is an essential part to reach this goal.

For every individual ν_i, we define a probability mass distribution (PMD) $I_{\nu_i}(t)$ that describes the probability of the individual to receive the information from its neighbors at time t.

$$I_{\nu_i}(t) = p(1-p)^t \tag{6.4}$$

where $p = \Psi_{\nu_i}$ represents the conviction that individual ν_i will receive the information from one of its neighbors.

The PMDs I_{ν_i} represent the timing behavior of corresponding individuals ν_i. To estimate a bound on the timing for a consensus to be reached, we need to look at the collective timing behavior in the form of a probabilistic worst case convergence time, which rises two concerns: (i) how to translate the individual timing behavior into a group timing and (ii) what possible structures to study in order to achieve a tighter bound on the pWCCT.

Looking at the first issue, we look at how to combine all possible ways in which the information could propagate in the network in terms of timing. We rely on the convolution operator to implement this;

$$I_{\nu_i} * I_{\nu_j}(t) = \sum_{s=-\infty}^{\infty} I_{\nu_i}(s) I_{\nu_j}(t-s) \tag{6.5}$$

Since the interest lays in estimating the worst time to convergence, and in order to estimate a tighter upper-bound on the pWCCT, instead of considering the longest path that spans all nodes, we look for a spanning tree with minimum branching.

For the convolution to work however, we have to assume that (i) the PMDs are independent and identically distributed (i.d.d) and (ii) the information propagates in a sequential manner to all the individuals (See Figure 6.8b). The inclusion of branches in the considered structure hinders the use of convolution solely. The way we go about combining the nodes in the branches with the main spanning tree is by defining a merge operator:

$$I_{\nu_i} \uplus I_{\nu_j}(t) = I_{\nu_j}(t) \sum_{s \leq t} I_{\nu_i}(s) + I_{\nu_i}(t) \sum_{s < t} I_{\nu_j}(s) \tag{6.6}$$

which looks into the probability of the information taking longer to reach the nodes in the branch than the main path and vice-versa. This takes into account the possibility that certain individuals receive the information at the same time, akin to a parallel process. In the structure of Figure 6.8c, the merging will look at the probability of the information reaching x at time t while the information has already spread to Λ at time $s \leq t$ and vice-versa. This reduces the pessimism of the convolution-based pWCCT estimation and offers a tighter bound.

Theorem 1. *Let Λ be a simple path and x be a node in a graph G such that $x \notin \Lambda$ then, merging $I_x(t)$ and $I_\Lambda(t)$ yields a tighter bound on the pWCCT than convolution.*

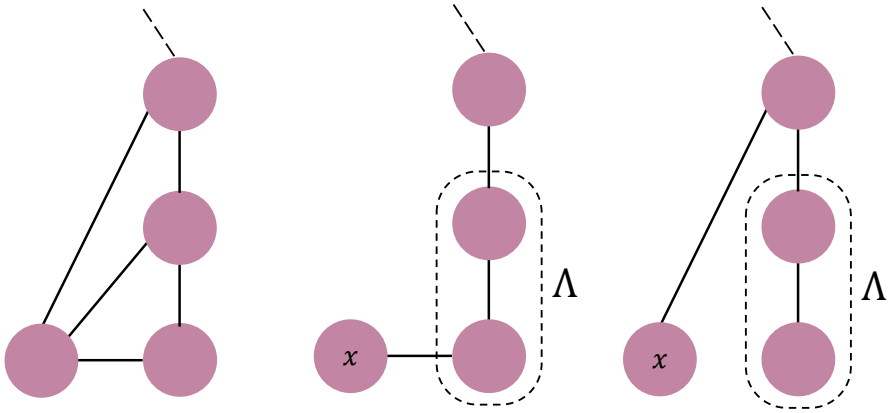


Figure 6.8: Example graph to demonstrate (b) sequential propagation of information from Λ to x that requires convolution of PMDs and (c) the less pessimistic structure of seeing x as part of a branch, akin to parallel process, which requires the merge operator.

We introduce a small lemma to prove this theorem.

Lemma. *Given a double summation, interchanging the order yields,*

$$\sum_{j=0}^{n}\sum_{i=1}^{j} f(i,j) = \sum_{i=0}^{n}\sum_{j=i+1}^{n} f(i,j)$$

This can be simply reached by looking at the double summation as one sum $\sum_{t\in\{(i,j)|j\leq i\leq n\}} f(t)$

Proof. Since the pWCCT is an exceedance probability distribution, to prove tightness of bound, it is enough to show that the CDF of merging at any given time t is larger than that of convolution. Formally, we want to prove that

$$\sum_{k=0}^{t} I_\Lambda \uplus I_x(k) \geq \sum_{k=0}^{t} I_\Lambda * I_x(k)$$

The convolution of two PMDs can be rewritten as a *Cauchy Product*,

$$\sum_{k=0}^{t} I_\Lambda * I_x(k) = \sum_{k=0}^{t} I_x(k) \sum_{s=0}^{t} I_\Lambda(s)$$

For the merge operator, given Equation 6.6, we study two cases;

For $k \geq t$:

$$\begin{aligned}\sum_{k=0}^{t} I_\Lambda \uplus I_x(k) &= \sum_{k=0}^{t}\left(I_\Lambda(k)\sum_{s=0}^{k} I_x(s) + I_x(k)\sum_{s=0}^{t} I_\Lambda(s) + I_x(k)\sum_{s=t+1}^{k} I_\Lambda(s)\right)\\ &> \sum_{k=0}^{t} I_\Lambda * I_x(k)\end{aligned}$$

For $k < t$:

$$\sum_{k=0}^{t} I_\Lambda \uplus I_x(k) = \sum_{k=0}^{t}\left(I_\Lambda(k)\sum_{s=0}^{k} I_x(s) + I_x(k)\sum_{s=0}^{t} I_\Lambda(s) - I_x(k)\sum_{s=k+1}^{t} I_\Lambda(s)\right)$$

Given the lemma, the last term can be rewritten as:

$$\begin{aligned}\sum_{k=0}^{t} I_\Lambda \uplus I_x(k) &= \sum_{k=0}^{t}\left(I_\Lambda(k)\sum_{s=0}^{k} I_x(s) + I_x(k)\sum_{s=0}^{t} I_\Lambda(s) - I_\Lambda(k)\sum_{k=1}^{t} I_x(s)\right)\\ &= \sum_{k=0}^{t}\left(I_\Lambda(k)I_x(0) + I_x(k)\sum_{s=0}^{t} I_\Lambda(s)\right)\\ &\geq \sum_{k=0}^{t} I_\Lambda * I_x(k)\end{aligned}$$

□

6.6.4 Rumour Data Collection

We relied on the Twitter API to collect the relevant information to build the network connecting users Alice and Bob to the users involved in the rumour and compile the data in the form of a group interaction model as described above. The parameter of most importance in the graph model is the probability of an individual transmitting an information to its neighbors p_{ij}. In the context of social media networks and interactions among humans, this refers to what is commonly addressed as *social influence*. On that account, metadata were extracted to represent the influence probability p_{ij} defined as the probability of user i to influence the opinion of user j. It is common in literature to rely on the rate of communication to quantify the influence [144, 145]. In here, we define this parameter as:

$$f_c(i \mapsto j) = \frac{\text{Number of tweets } i \mapsto j}{\text{Last 1000 tweets}}$$

In order to better represent the influence between individuals, we estimate the influence based on two other quantities; the level of trust between users f_t and the popularity of the user in the network f_p.

$$f_t(i \mapsto j) = \begin{cases}\texttt{True} & (i,j) \in \Phi(j) \cup \Phi(i)\\ \texttt{False} & otherwise\end{cases}$$

Where $\Phi(i)$ represents the set of individuals following user i.

$$f_p(i) = \frac{|\Phi(i)|}{\max_{k\in\mathcal{V}} |\Phi(k)|}$$

We then define the probability of i influencing j as a weighted combination of these quantities:

$$p_{ij} = \omega_c f_c + \omega_p f_p + \omega_t f_t \tag{6.7}$$

Where $\sum_k \omega_k = 1$.

We explored different values to the weights and we noted that giving a high weight to the rate of communication f_c, following the common trend in literature, didn't show any discernible patters. We observed the same result when giving a high weight to the trust factor f_t. The popularity factor, on the other hand, with a slightly higher weight, resulted in the patterns observed in Figure 6.5. This is highly pertinent to the fact that the case we study is the propagation of a rumour in a wide-scale event, such as a siege, in which the popularity of the propagator plays a bigger role in influencing their audience. We hypothesize that a study of a smaller scale such as among family and peers would require the assignment of higher weights to the communication rate and trust factors.

6.6.5 pWCCT Experiments

Experimental Setup

The communication protocol in the swarm follows the strategy proposed by Pinciroli et al.[146] labelled *Virtual Stigmergy*; which is inspired by communication among insects and is robust to sharing information in large swarms even under noisy conditions. The information is stored as a timestamped tuple `(key, value)` and transmitted in a message to a robot's neighborhood. We encourage interested readers to go over the paper to fully understand how Virtual Stigmergy works. In here, we limit the text to the elements necessary to understand the experiment setup. Virtual Stigmergy states that a robot updates its tuple space only if it receives a `(key, value)` pair that either does not exist in its table or a pair with a higher timestamp; which indicates that the value received is more up-to-date than the stored value. In these cases, the robot will update its table and broadcast a message to its neighbors in order to share the updated information. In this manner, the information propagates from a source node to the rest of the swarm even if the network is not strongly connected.

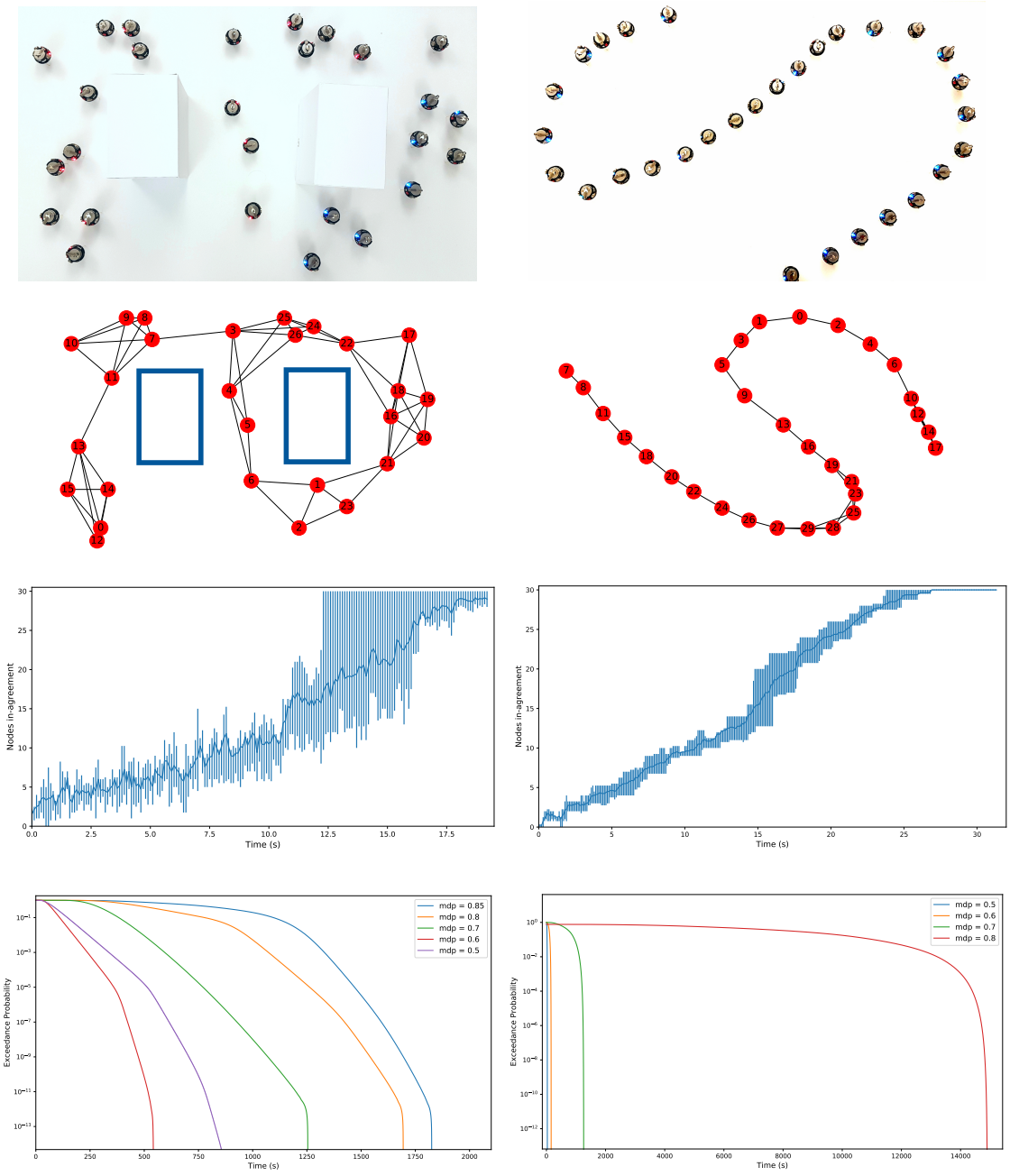
To ensure convergence, in these experiments, aside from the broadcasting procedure described above, the robots broadcast a message containing their state every period of time T_s. This time period models the time-step time that we mentioned in a previous section and is essential to build the PMDs (Equation 6.4) in which a robot fails to receive a message from its neighbors in the time $kT_s < t < (k+1)T_s$.

We recall that the Kilobots rely on infra-red communication which is unreliable and highly sensitive to the experiment's environment. For such, a set of separate experiments were performed to gauge the message drop probability of the Kilobots, labelled so forth *mdp*. Although the quantity was highly sensitive to the saturation level of the communication space and the environment such as the ambient light, the reflection of the communication medium, etc., the recorded probabilities were in the range $mdp \in [0.63, 0.78]$.

Additional Results

Figure 6.9 summarizes the results obtained for two different topologies: (a) a randomly distributed swarm with two obstacles to limit the communication between sections of the swarm and (b) a snake-like topology similar to a line topology which differs in the fact that the out-degree of every node is not forced to 1. The figures plot the times to convergence from 30 runs on Kilobots and their corresponding pWCCT estimations from our model.

The first thing that we observe is that the variance in estimations for different *mdp* differs from one topology to another which is expected since our model relies on the topology to estimate the pWCCT, more specifically, on the structure considered to combine the individual timing behaviors. Particularly, we observe that the estimations for the snake topology are more pessimistic as the *mdp* increases. This is due to fact that the structure of the spanning tree with minimum branching is closer to that of the longest path which implies that the convolution operator is mostly used. As proven before, the convolution introduces pessimism to the estimation which explains the results of Figure 6.9.b.



(a) Random Topology with Obstacles (b) Snake Topology

Figure 6.9: Time for a swarm of Kilobots to reach complete agreement and its corresponding estimation of pWCCT for (a) a random topology with obstacles and (b) a snake topology.

CHAPTER 7 GENERAL DISCUSSION

This chapter discusses the research work presented in the previous Chapters 3 – 6 and highlights the findings of the experiments conducted. Table 7.1 wraps up the chapter with a summary of the main contributions of this research.

7.1 Bridging the gap

There is a rift between the needs of industry and what is pursued by academic research [147]. Complex large software systems such as those found in the avionics and automotive industries tend to embrace a certain rigidity in the design of their systems. This stems from the criticality of such systems which favors the predictability of the system behavior before deployment. As such, there is a reluctance to move towards potentially better systems and explore more daring approaches proposed through both fundamental and applied research, despite their demonstrated efficiency. At the same time, because of confidentiality reasons, researchers have limited access to essential information to properly model the system. A lack that researchers fill by taking appropriate assumptions to approach issues and questions known to these fields. For similar reasons, the curiosity-driven nature of "Blue Sky research" tends to build approaches on assumptions that, on one side, certainly advance knowledge and might stumble upon scientific breakthroughs, but this makes them unsuitable and hence undesirable for the design of industrial applications.

> *"New technology always needs an additional effort for optimization and confirmation of its practical feasibility. If this extra step is not carried out, a new technology, regardless of the hard background work put into it, will most likely fade away and never develop into a technology that can make real benefits to society."*
>
> – Jong-Hyun Ahn,
> *From an idea to a technology*,
> Nature Nanotechnology, 2018. [22]

Chapter 1 calls attention to this issue by highlighting the "lack of relevant data to describe, optimize and integrate complex large real-time systems" as well as the "fluctuation of relevance of information from one phase to another during system design" which makes it challenging to employ certain techniques such as Design Space Exploration methodologies.

The work presented in Chapters 3, 4, and 5 in part attempts to bridge this gap and bring proposed academic research close to being implemented in real-life applications.

In particular, exploring the design space through search and optimization techniques the likes of those developed in Appendix B requires access to information that Subject Matter Experts oftentimes lack the expertise to provide. Although tools that could produce performance metrics about a particular system are available, they tend to be slow and their use becomes prohibitive in the context of Design Space Exploration of real-time embedded systems as discussed in Section 3.4.2. The work in Chapter 3, as a matter of fact, takes advantage of the accuracy provided by hardware simulators and the speed of an analytical approach such as regression to estimate the performance of the system (i) without having access to the end-target hardware platform on which a system is to be deployed and (ii) in a comparatively reasonable time. This last feature shines when methodologies that explore the design space, such as when deciding the optimal final hardware architecture, are demanded. These methodologies generally require the evaluation of a great number of possible configurations that can grow to hundreds of thousands. Clearly, relying on simulations solely becomes restrictive, even with the fastest simulators available at the time of writing. Having a framework, such as the one presented in [19], that can provide performance estimates of thousands of scenarios in a short time can be a game changing addition and a first step to using DSE technologies to enhance the integration of complex systems.

In the same effort to provide relevant information to the design of complex real-time systems, Chapter 4 extends the Real-Time Systems toolkit with more accurate scheduling tools that conform to today's technological advances. The work presented in 4 [21] develops a random cache model that is able to provide estimates of pWCET while being aware of memory sharing contention. The probabilistic nature of the timing analysis, by itself, provides tighter bounds on the execution time, an information that is valuable and is of practical use to experts designing systems under hard timing constraints. By virtue of its compact nature, as compared to already existing cache models, the proposed model is integrated in a multi-core scheduling simulator. Section **??** illustrates how it can be quite useful in uncovering risky configurations and deciding the scheduling policies that are more suited for the studied scenarios. This is becoming of relevance as critical system developers are turning more towards multi-core platforms to deploy their complex systems which generally involve the optimization of resources and exploring better allocations. The work cited in Appendix B falls under this category where the authors rely on the use of a scheduling simulator to find optimum allocation of tasks to a set of available resources.

7.2 A Hidden Simplicity

As highlighted by the problem statement on Chapter 1 and discussed in the previous section, exploratory research tends to make quite the number of assumptions that are difficult to translate to practical uses. This, as shown, might be prevalent in the complex real-time system and embedded systems but it stretches to other fields as well, especially in the natural sciences. This is partly due to the restrictions and limited access to live subjects, especially in the wilderness. Chapter 6 attempts to answer the question: *What knowledge can be acquired by studying a system through the simplest of information propagation models?*. The first half of the chapter explores the scenario of having limited access to data and attempts to model the propagation of information in a connected network with a minimal number of assumptions; namely (1) a static or slowly changing network, (2) the propagation of a single piece of information and (3) information transmission probability of a node. The latter is dependent on the scenario studied and can be modelled as the infection probability, influence of users, etc. The modelling approach applied to the rumour case study detailed in Section 6.4.1 was able to give a reason for the reported progression of events. It further exposed influential users in the studied network that have high potential to affect the spread or attenuation of the rumour. The purpose here was to explore the effect of stripping the propagation model from scenario-specific details, such as the volume of shared tweets or the rate at which the information is transmitted, etc. This showed that the few assumptions taken in modeling information propagation were enough to discern the patterns that lead to the emergence of the observed event. The study, although narrow in terms of scope and possibly biased by the choice of scenario, promotes the practice of bottom-up investigation when it comes to modeling information flow and to properly identify and isolate the originator of specific events and behaviors; as is the case with giant and snowball effects in rumour propagation research.

7.3 Information Propagation in Artificial Systems

As it is common with information propagation models proposed in literature (some of which have been addressed in Section 2.2.1), there is a certain passiveness to approaching the matter of propagation of information, especially when biological systems are concerned. These models are developed to uncover the underlying causal dependence, if any, between an observed behavior and a given property or are satisfied with the study of the distributions that inhibit the propagation itself. The rumour study discussed above falls under the same category since the aim was to explain why certain events occurred in the way they did. Having a more active

approach to studying information flow might not be of import to biological systems yet, but the knowledge gained so far could be transferred to benefit artificial systems.

As a matter of fact, the study of the proposed information flow on the siege rumour presented in the previous chapter revealed the potential of studying infromation flow. It was indeed able to identifying the strong elements that lead to a stronger circulation of the rumour. Section 6.4.2 exploits this finding to engineer more efficient wireless sensor networks that ensure a faster spread of information by locating the weakest links in the network that are more prone to hinder the propagation of information. The results shown in Figure 6.7 demonstrate the performance of the proposed approach, especially when analyzing heterogeneous networks.

Transferability of information flow models and findings is not only limited to artificial systems that, in one way or another, imitate biological systems; either in structure (e.g. wireless sensor networks) or in behavior (e.g. robot swarms). It extends to any system that can be abstracted as a network of nodes linked with some kind of causal dependency relationship similar to the study of [59] conducted to detect the root of anomalies by studying the spread of fault in large complex systems. This is the subject of the work presented in Chapter 5 which addressed the description of large complex systems. These systems can be viewed as a network of connected modules which, differently from multi-agent systems, can not function as separate agents. Due to the black-box view of the system modules, and in order to break a cycle, we virtually inject an error at these nodes and study the spread of the error in the system to assign a criticality attribute to the data being transmitted by the different modules. This has the potential to represent raw data of a system into a practical description such as a DAG. Even with the apprehension that some industries have towards this kind of fundamental work, this approach offers a transformed graph description of the system that can provide starting configurations to integration specialists that can have a huge impact on time-to-market as well as cost (This is backed-up by the findings of the Full-Mission Simulator case study of Section 5.6.2).

Finally, Section 6.3 capitalizes on merging the lessons learned from biology – information propagation models – with techniques built for real-time software/hardware systems – probabilistic timing analysis – which lead to the emergence of new techniques for systems of connected artificial entities. The static probabilistic timing analysis (SPTA) approach, proposed for a single random cache presented in Chapter 4, was extended with the proposed information propagation model. The new approach establishes a probabilistic upper-bound for the time it takes for an information to propagate in the system. Figure 6.6 shows that, despite the few assumptions taken to model the information propagation, which neither fully

expresses the intricacies of the diffusion of information nor the environment, it is able to provide a probabilistic measure of the worst convergence time towards a consensus.

With the growing interest that both research and industry are giving robot swarms, to respond in emergency cases or to explore the far-reaches of space, knowing what and when an event is to occur becomes crucial. This is applicable on a small scale such as Amazon's fleet of transporting drones or larger scale the likes of Facebook's ambitious plan to provide internet access through a linked network of drones [148]. The work in Chapter 6, by providing a timing estimate of flexible and practical use to the design of software intended for these types of systems, along the tools developed in Chapters 3 and 4, are an essential first step to ensuring a proper timing behavior and advancing the development of these technologies.

Table 7.1: Summary of the main contribution of the disseration.

Ch.	Ref.	Contributions	Impact & Influence
3	[19]	A framework for performance estimation with zero knowledge of software functionality and no access to the hardware platform.	Advance integration of large systems and ease the adoption of DSE techniques.
4	[21]	A compact random cache model for the estimation of (pWCET) of software executing on different cores and sharing cache space.	The ease of integration into modern simulation tools and reinforcement of shared cache modelling literature.
		An extension to an existing scheduling simulator with a cache contention-aware execution of applications scheduled on multi-core systems.	An addition to the real-time systems design and validation toolkit and contributor to more accurate design space exploration.
5	[20]	A methodology to transform raw and scarce data into a pertinent system description in the form of a DAG.	Connecting the benefits of existing techniques with the needs of industry like by access to DAG-based techniques abd easing the integration and optimization of systems.
		A strategy to provide starting configurations for the assessment of proper functional and timing behavior of complex systems.	A flexibility and acceleration of software system integration and a more-desirable time-to-market.
6	–	A probabilistic modelling of information propagation among interacting entities under minimal assumptions.	Re-evaluattion of the complexity in information flow modelling and the benefit of bottom-up exploration strategies.
		A timing analysis approach to estimate a probabilistic measure of the worst convergence time towards an information consensus.	Enabling the design of hard real-time multi-agent systems by providing a flexible and practical timing property.
		strategy to strength the information spreading in connected networks such as robot swarms	Incentive to further explore information flow in the engineering of better artificial systems.

CHAPTER 8 CONCLUSIONS

"One is a scholar as long as he keeps seeking knowledge, the moment he thinks that he has learned it all, he is ignorant."

– Ibn Al-Mubarak

"Now it is established in the sciences that no knowledge is acquired save through the study of its causes and beginnings, if it has had causes and beginnings; nor completed except by knowledge of its accidents and accompanying essentials."

– Ibn Sina

With the data overload experienced thanks to today's technology, the main questions that our research attempted to answer are: *Is relevant information truly abundant? and if not, how to grasp its essence in a manner that is meaningful to the advancement of knowledge?* The research journey undertaken here started with the intention to bridge the gap left by the mix of expertise in approaching a problem and implementing the existing and proposed solutions in the design of complex real-time systems. These have been outlined in objectives 1 and 2 – providing relevant information and accurate real-time tools for a better exploration of the design space, which was covered throughout Chapters 3 and 4. The role of information then became palpable introducing the work in Chapter 5, which tackled objective 3 – determining the criticality of data for the purpose of providing convenient descriptions of the system by studying the flow of information. The outcomes of that work was an incentive to engage objective 4 – modeling the information flow under fewer assumptions. The research done in Chapter 6 was driven by curiosity to appraise the potential behind the simplicity of the model to explain and uncover existing and likely new patterns. In undertaking objectives 5 and 6, Chapters 5 and 6 promote the study of information flow as a bedrock to investigate and engineer new and more efficient solutions both in large software systems and multi-agent systems.

The discussion performed in the previous chapter highlighted the findings of these works and their impact on biological systems in part but especially on artificial systems. The significance of this research endeavor shines in these outcomes: (1) strategies to contribute and specify relevant information for the bettering of the relationship between academic achievements and the design of industrial systems; (2) A better understanding of the importance of the

information flow and the underlying simplicity that might govern certain behaviors; (3) Frameworks to engineer better connected networks and boost the information propagation among interacting agents.

This chapter brings this document to an end with a summary of the main findings of this research and the lessons learned both scientifically and personally and closes up with a couple questions to be addressed in future research and some topics that could be explored and built on top of the currently presented findings.

8.1 Knowledge Transfer and Self-Reflection

As ingrained in any journey, the research path undertaken during these past years yielded worthwhile lessons, some more obvious than others. We outline the main ones here:

1. As Gary Johnson's book title goes: "Living with Less in the Land of More", the trend seems to extend to our understanding of life as a whole and technological design in particular. Through this research, Chapter 6 in particular, a simplicity to modeling information was uncovered to be in some cases enough to explain certain behaviors;

2. Although it is starting to be established that information is inherent to biological systems, information flow is a potent property to be explored to a great extent in the context of artificial systems as has been demonstrated through [20] and 6;

3. Probabilistic models are powerful tools to fill the void of uncertainty and the unknown due to ambiguous descriptions or faulty devices (e.g. unreliable communication media, unpredictable faults, erratic behavior, etc.);

4. Recognized techniques built on certain assumptions that have long been established in both academia and industry should be re-evaluated as technology advances and their premises are no longer negligible as is the case of memory contention introduced in [21].

For the purpose of sharing experiences and lessons learned through the process of performing research, these are some self-reflective suggestions for the interested reader:

1. In the process of designing one's research, it is fruitful to have a clear idea of what the research questions that those specific experiments are going to answer;

2. Hand in hand with the former point, it is better to establish the kind of research one wants to pursue; problem-driven versus curiosity-driven;

3. Explore side-projects and get involved with works in other research laboratories, if possible. It will build your knowledge of other topics as well as develop new attitudes to approaching your own research;

4. Embrace failure. Every researcher have had some of their works rejected at some point. The best learn to cry about it then pick themselves up and progress.

8.2 Future Ventures

As the epigraph at the beginning of this chapter states, scientists are in a constant quest for uncovering the unknown and improving existing knowledge. Predictably, the last developed research presented in Chapter 6 is comparatively young which makes it closer to today's issues and more in need of growing. This passage expands on few limitations of the presented research that could be addressed in future work:

- The probabilistic model of information propagation was developed under the assumption that a single piece of information is propagated, conceivably from one or more sources. Further study would extend the model to handle multiple, possibly conflicting, information spreading;

- A fresher look at information flow has been introduced here which views information as an emergent property instead of a stored entity. A question that this opens though is whether information and its flow are an inherent property to artificial multi-agent systems as it is deep-rooted in the essence of biological networks?

A compelling future venture that further promotes the duality of biology and technology discussed at the beginning of this document has been proposed. It aspires to untangle the factors that contribute to the emergence of certain behaviors, and more importantly those that are responsible for transitional behaviors, from a robotic swarm point of view. The first stage to achieve this endeavor has already been undertaken under the work performed during the National Institute of Informatics Internship program spent during the first half of 2018 in Tokyo, Japan, which deals with identifying relevant subspaces – and in turn relevant factors – within high-dimensional data (The summary of which can be found in Appendix A). The next stages will have to call on knowledge from the machine learning and meta-heuristics fields to develop new approaches in order to reach a further meaningful understanding of information.

REFERENCES

[1] S. Ozawa, T. Maeda, T. Matsumura, and K. Uchida, "Micro-pressure waves radiating from exits of shinkansen tunnels," *Railway Technical Research Institute, Quarterly Reports*, vol. 34, no. 2, 1993.

[2] D. W. Haldane, M. Plecnik, J. K. Yim, and R. S. Fearing, "Robotic vertical jumping agility via series-elastic power modulation," *Science Robotics*, vol. 1, no. 1, 2016.

[3] O. Bolmin, C. Duan, L. Urrutia, A. M. Abdulla, A. M. Hazel, M. Alleyne, A. C. Dunn, and A. Wissa, "Pop! observing and modeling the legless self-righting jumping mechanism of click beetles," in *Biomimetic and Biohybrid Systems*. Cham: Springer International Publishing, 2017, pp. 35–47.

[4] A. Witze, "High-jumping beetle inspires agile robots," *Nature News*.

[5] J. W. Jolles, N. J. Boogert, V. H. Sridhar, I. D. Couzin, and A. Manica, "Consistent individual differences drive collective behavior and group functioning of schooling fish," *Current Biology*, vol. 27, no. 18, pp. 2862 – 2868.e7, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0960982217310138

[6] S. B. Rosenthal, C. R. Twomey, A. T. Hartnett, H. S. Wu, and I. D. Couzin, "Revealing the hidden networks of interaction in mobile animal groups allows prediction of complex behavioral contagion," *Proceedings of the National Academy of Sciences*, vol. 112, no. 15, pp. 4690–4695, 2015.

[7] D. S. Brown, R. Turner, O. Hennigh, and S. Loscalzo, "Discovery and exploration of novel swarm behaviors given limited robot capabilities," in *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 447–460.

[8] M. Moses, G. Bezerra, B. Edwards, J. Brown, and S. Forrest, "Energy and time determine scaling in biological and computer designs," *Phil. Trans. R. Soc. B*, vol. 371, no. 1701, p. 20150446, 2016.

[9] J. H. Brown, W. R. Burnside, A. D. Davidson, J. P. DeLong, W. C. Dunn, M. J. Hamilton, N. Mercado-Silva, J. C. Nekola, J. G. Okie, W. H. Woodruff, and W. Zuo, "Energetic limits to economic growth," *BioScience*, vol. 61, no. 1, pp. 19–26, 2011. [Online]. Available: http://dx.doi.org/10.1525/bio.2011.61.1.7

[10] P. Balister, J. Balogh, E. Bertuzzo, B. Bollobás, G. Caldarelli, A. Maritan, R. Mastrandrea, R. Morris, and A. Rinaldo, "River landscapes and optimal channel networks," *Proceedings of the National Academy of Sciences*, 2018.

[11] M. Gell-Mann, *The Quark and the Jaguar: Adventures in the Simple and the Complex*. Macmillan, 1995.

[12] P. M. Binder and A. Danchin, "Life's demons: information and order in biology," *EMBO reports*, vol. 12, no. 6, pp. 495–499, 2011. [Online]. Available: http://embor.embopress.org/content/12/6/495

[13] P. C. Davies and S. I. Walker, "The hidden simplicity of biology," *Reports on Progress in Physics*, vol. 79, no. 10, p. 102601, 2016.

[14] R. H. Turner, L. M. Killian *et al.*, *Collective behavior*. Prentice-Hall Englewood Cliffs, NJ, 1957.

[15] R. Landauer *et al.*, "Information is physical," *Physics Today*, vol. 44, no. 5, pp. 23–29, 1991.

[16] V. de Lorenzo, "From the selfish gene to selfish metabolism: revisiting the central dogma," *Bioessays*, vol. 36, no. 3, pp. 226–235, 2014.

[17] Q.-l. Chen, L. Chen, Z.-Q. Sun, and Z.-j. Jia, "An epidemic propagation model with saturated infection rate on a small world network," in *International Conference on Intelligent Computing*. Springer, 2015, pp. 34–42.

[18] S. Namilae, P. Derjany, A. Mubayi, M. Scotch, and A. Srinivasan, "Multiscale model for pedestrian and infection dynamics during air travel," *Physical review E*, vol. 95, no. 5, p. 052320, 2017.

[19] I. Hafnaoui, R. Ayari, G. Nicolescu, and G. Beltrame, "A simulation-based model generator for software performance estimation," in *Proceedings of the Summer Computer Simulation Conference*, ser. SCSC '16, 2016, pp. 20:1–20:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=3015574.3015594

[20] ——, "Scheduling real-time systems with cyclic dependence using data criticality," *Design Automation for Embedded Systems*, vol. 21, no. 2, pp. 117–136, 2017.

[21] I. Hafnaoui, C. Chen, R. Ayari, G. Nicolescu, and G. Beltrame, "An analysis of random cache effects on real-time multi-core scheduling algorithms," in *Proceedings of the*

28th International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype. ACM, 2017, pp. 64–70.

[22] A. Moscatelli, "From an idea to a technology," *Nature nanotechnology*, vol. 13, no. 7, p. 528, 2018.

[23] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM computing surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[24] M. Niemeier, A. Wiese, and S. Baruah, "Partitioned real-time scheduling on heterogeneous shared-memory multiprocessors," in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*. IEEE, 2011, pp. 115–124.

[25] A. Patel, F. Afram, S. Chen, and K. Ghose, "Marss: a full system simulator for multicore x86 cpus," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. IEEE, 2011, pp. 1050–1055.

[26] R. Urunuela, A.-M. Déplanche, and Y. Trinquet, "Storm a simulation tool for real-time multiprocessor scheduling evaluation," in *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*. IEEE, 2010, pp. 1–8.

[27] M. Chéramy, A.-M. Déplanche, P.-E. Hladik, and Others, "Simulation of Real-Time Multiprocessor Scheduling with Overheads," *International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, 2013.

[28] M. Jones, "What really happened on mars rover pathfinder," *The Risks Digest*, vol. 19, no. 49, pp. 1–2, 1997.

[29] Z. Huang, C. C. Shen, S. Doshiy, N. Thomasy, and H. Duong, "Difficulty-level metric for cyber security training," *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2015 IEEE International Inter-Disciplinary Conference on*, pp. 172–178, 2015.

[30] A. Sardinha, T. A. Alves, L. A. J. Marzulo, F. M. G. França, V. C. Barbosa, and V. S. Costa, "Scheduling cyclic task graphs with SCC-map," *Proceedings - 3rd Workshop on Applications for Multi-Core Architecture, WAMCA 2012*, pp. 54–59, 2012.

[31] G. F. Zaki, W. Plishker, S. S. Bhattacharyya, and F. Fruth, "Implementation, scheduling, and adaptation of partial expansion graphs on multicore platforms," *Journal of Signal Processing Systems*, pp. 1–19, 2016.

[32] Q. Tang, T. Basten, M. Geilen, S. Stuijk, and J.-B. Wei, "Mapping of synchronous dataflow graphs on mpsocs based on parallelism enhancement," *Journal of Parallel and Distributed Computing*, vol. 101, pp. 79–91, 2017.

[33] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[34] J. Gaisler, "The leon/erc32 gnu cross-compiler system," 2001.

[35] P. Joseph, K. Vaswani, and M. J. Thazhuthaveetil, "Construction and use of linear regression models for processor performance analysis," in *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on.* IEEE, 2006, pp. 99–108.

[36] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based gpu design space exploration," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on.* IEEE, 2012, pp. 2–13.

[37] A. Verma, L. Cherkasova, and R. H. Campbell, "Profiling and evaluating hardware choices for mapreduce environments: An application-aware approach," *Performance Evaluation*, vol. 79, pp. 328–344, 2014.

[38] E. Quinones, E. Berger, G. Bernat, and F. Cazorla, "Using randomized caches in probabilistic real-time systems," in *Real-Time Systems, 2009. ECRTS '09. 21st Euromicro Conference on*, July 2009, pp. 129–138.

[39] S. Altmeyer and R. Davis, "On the correctness, optimality and precision of static probabilistic timing analysis," in *Design, Automation and Test in Europe Conference and Exhibition*, March 2014, pp. 1–6.

[40] R. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean, "Analysis of probabilistic cache related pre-emption delays," in *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, July 2013, pp. 168–179.

[41] S. Altmeyer, L. Cucu-Grosjean, and R. Davis, "Static probabilistic timing analysis for real-time systems using random replacement caches," *Real-Time Systems*, vol. 51, no. 1, pp. 77–123, 2015.

[42] B. Lesage, D. Griffin, S. Altmeyer, and R. Davis, "Static probabilistic timing analysis for multi-path programs," in *Real-Time Systems Symposium, 2015 IEEE*, Dec 2015, pp. 361–372.

[43] J. Mars, L. Tang, and M. L. Soffa, "Directly characterizing cross core interference through contention synthesis," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*. ACM, 2011, pp. 167–176.

[44] C. Ding, X. Xiang, B. Bao, H. Luo, Y.-W. Luo, and X.-L. Wang, "Performance metrics and models for shared cache," *Journal of Computer Science and Technology*, vol. 29, no. 4, pp. 692–712, 2014.

[45] X. Pan and B. Jonsson, "A modeling framework for reuse distance-based estimation of cache performance," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 62–71.

[46] N. Beckmann and D. Sanchez, "Modeling cache performance beyond lru," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 225–236.

[47] M. Chéramy, P.-E. Hladik, A.-M. Déplanche, and S. Dal Zilio, "Simulation of real-time scheduling algorithms with cache effects," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2015.

[48] D. Eklov, D. Black-Schaffer, and E. Hagersten, "Fast modeling of shared caches in multicore systems," in *Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers*, ser. HiPEAC '11. New York, NY, USA: ACM, 2011, pp. 147–157.

[49] R. B. Laughlin, *A different universe: Reinventing physics from the bottom down*. Basic books, 2008.

[50] J. de Vreeze and C. Matschke, "Keeping up appearances: Strategic information exchange by disidentified group members," *PloS one*, vol. 12, no. 4, p. e0175155, 2017.

[51] S. A. Hale, P. John, H. Margetts, and T. Yasseri, "How digital design shapes political participation: A natural experiment with social information," *PloS one*, vol. 13, no. 4, p. e0196068, 2018.

[52] A. Strandburg-Peshkin, C. R. Twomey, N. W. Bode, A. B. Kao, Y. Katz, C. C. Ioannou, S. B. Rosenthal, C. J. Torney, H. S. Wu, S. A. Levin *et al.*, "Visual sensory networks and effective information transfer in animal groups," *Current Biology*, vol. 23, no. 17, pp. R709–R711, 2013.

[53] E. R. Brush, N. E. Leonard, and S. A. Levin, "The content and availability of information affects the evolution of social-information gathering strategies," *Theoretical Ecology*, vol. 9, no. 4, pp. 455–476, 2016.

[54] T. Başar, S. R. Etesami, and A. Olshevsky, "Convergence time of quantized metropolis consensus over time-varying networks," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 4048–4054, 2016.

[55] L. Hindersin, M. Möller, A. Traulsen, and B. Bauer, "Exact numerical calculation of fixation probability and time on graphs," *Biosystems*, vol. 150, pp. 87–91, 2016.

[56] P. Jia, J. Liu, Y. Fang, L. Liu, and L. Liu, "Modeling and analyzing malware propagation in social networks with heterogeneous infection rates," *Physica A: Statistical Mechanics and its Applications*, vol. 507, pp. 240–254, 2018.

[57] J. Wang, X. Wang, and J. Wu, "Inferring metapopulation propagation network for intra-city epidemic control and prevention," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 830–838.

[58] B. Ottino-Löffler, J. G. Scott, and S. H. Strogatz, "Takeover times for a simple model of network infection," *arXiv preprint arXiv:1702.00881*, 2017.

[59] J. Ni, W. Cheng, K. Zhang, D. Song, T. Yan, H. Chen, and X. Zhang, "Ranking causal anomalies by modeling local propagations on networked systems," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017, pp. 1003–1008.

[60] C. P. Sankar, S. Asharaf, and K. S. Kumar, "Learning from bees: An approach for influence maximization on viral campaigns," *PloS one*, vol. 11, no. 12, p. e0168125, 2016.

[61] S. Gavrilets, J. Auerbach, and M. Van Vugt, "Convergence to consensus in heterogeneous groups and the emergence of informal leadership," *Scientific reports*, vol. 6, p. 29704, 2016.

[62] L. Cheng, Y. Wang, W. Ren, Z.-G. Hou, and M. Tan, "On convergence rate of leader-following consensus of linear multi-agent systems with communication noises," *IEEE Transactions on Automatic Control*, vol. 61, no. 11, pp. 3586–3592, 2016.

[63] Z. Wang and J. Henkel, "Accurate source-level simulation of embedded software with respect to compiler optimizations," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 382–387.

[64] Z. Wang, K. Lu, and A. Herkersdorf, "An approach to improve accuracy of source-level tlms of embedded software," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*. IEEE, 2011, pp. 1–6.

[65] Z. Wang, A. Sanchez, A. Herkersdorf, and W. Stechele, "Fast and accurate software performance estimation during high-level embedded system design," 2008.

[66] Z. Wang and A. Herkersdorf, "An efficient approach for system-level timing simulation of compiler-optimized embedded software," in *Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009, pp. 220–225.

[67] ——, "Software performance simulation strategies for high-level embedded system design," *Performance Evaluation*, vol. 67, no. 8, pp. 717–739, 2010.

[68] C. Mills, S. C. Ahalt, and J. Fowler, "Compiled instruction set simulation," *Software: Practice and Experience*, vol. 21, no. 8, pp. 877–889, 1991.

[69] T. Nakada, T. Tsumura, and H. Nakashima, "Design and implementation of aworkload specific simulator," in *Proceedings of the 39th annual Symposium on Simulation*. IEEE Computer Society, 2006, pp. 230–243.

[70] M. Zhang, R. Qiao, N. Hasabnis, and R. Sekar, "A platform for secure static binary instrumentation," *ACM SIGPLAN Notices*, vol. 49, no. 7, pp. 129–140, 2014.

[71] M. Lattuada and F. Ferrandi, "Performance modeling of embedded applications with zero architectural knowledge," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 277–286.

[72] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ACM SIGOPS Operating Systems Review*, vol. 40, no. 5. ACM, 2006, pp. 185–194.

[73] H. Wang, Z. Zhu, J. Shi, and Y. Su, "Sensitivity analysis based predictive modeling for mpsoc performance and energy estimation," in *VLSI Design (VLSID), 2015 28th International Conference on*. IEEE, 2015, pp. 511–516.

[74] V. Zyuban, D. Brooks, V. Srinivasan, M. Gschwind, P. Bose, P. N. Strenski, and P. G. Emma, "Integrated analysis of power and performance for pipelined microprocessors," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 1004–1016, 2004.

[75] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The mälardalen wcet benchmarks: Past, present and future," in *OASIcs-OpenAccess Series in Informatics*, vol. 15. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2010.

[76] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on.* IEEE, 2001, pp. 3–14.

[77] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques.* ACM, 2008, pp. 72–81.

[78] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," in *ACM SIGARCH Computer architecture news*, vol. 41, no. 3. ACM, 2013, pp. 475–486.

[79] ——, "Zsim tutorial: Validation." in *Proceedings of the 48th International Symposium on Microarchitecture.* ACM, 2015.

[80] G. Martin, "Overview of the mpsoc design challenge," in *Proceedings of the 43rd Annual Design Automation Conference*, ser. DAC '06. New York, NY, USA: ACM, 2006, pp. 274–279. [Online]. Available: http://doi.acm.org/10.1145/1146909.1146980

[81] A. Hangan and G. Sebestyen, "Rtmultisim: A versatile simulator for multiprocessor real-time systems," *Proceedings of The 3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems, Pisa, Italy*, p. 15, 2012.

[82] B. Nikolic, M. A. Awan, and S. M. Petters, "Sparts: Simulator for power aware and real-time systems," in *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, Nov 2011, pp. 999–1004.

[83] D. Sanchez and C. Kozyrakis, "Zsim: fast and accurate microarchitectural simulation of thousand-core systems," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 475–486.

[84] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *11th International Symposium on High-Performance Computer Architecture*, Feb 2005, pp. 340–351.

[85] A. Sandberg, D. Black-Schaffer, and E. Hagersten, "Efficient techniques for predicting cache sharing and throughput," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12, 2012, pp. 305–314.

[86] G. Bernat, A. Colin, and S. Petters, "Wcet analysis of probabilistic hard real-time systems," in *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, 2002, pp. 279–288.

[87] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[88] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic dag task model," in *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*. IEEE, 2013, pp. 225–233.

[89] R. Ayari, I. Hafnaoui, A. Aguiar, P. Gilbert, M. Galibois, J.-P. Rousseau, G. Beltrame, and G. Nicolescu, "Multi-objective mapping of full-mission simulators on heterogeneous distributed multi-processor systems," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*, p. 1548512916657907, 2016.

[90] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[91] Y. Laalaoui and N. Bouguila, "Pre-run-time scheduling in real-time systems: Current researches and artificial intelligence perspectives," *Expert Systems with Applications*, vol. 41, no. 5, pp. 2196–2210, 2014.

[92] Y. Chandarli, F. Fauberteau, D. Masson, S. Midonnet, and M. Qamhieh, "Yartiss: A tool to visualize, test, compare and evaluate real-time scheduling algorithms," in *WATERS 2012*. UPE LIGM ESIEE, 2012, pp. 21–26.

[93] L. Thiele and N. Stoimenov, "Modular performance analysis of cyclic dataflow graphs," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009, pp. 127–136.

[94] J. Choi, H. Oh, S. Kim, and S. Ha, "Executing synchronous dataflow graphs on a spm-based multicore architecture," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 664–671.

[95] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*. CRC press, 2009.

[96] M. J. Morgan, "Integrated Modular Avionics for Next-Generation Commercial Airplanes," *IEEE Aerospace and Electronic Systems Magazine*, vol. 6, no. 8, pp. 9–12, 1991.

[97] G. Sebestyen and A. Hangan, "Genetic approach for real-time scheduling on multiprocessor systems," *Intelligent Computer Communication and Processing (ICCP), 2012 IEEE International Conference on*, pp. 267–272, 2012.

[98] M. Rathna Devi and A. Anju, "Multiprocessor Scheduling of Dependent Tasks to Minimize Makespan and Reliability Cost using NSGA-II," *International Journal in Foundations of Computer Science & Technology*, vol. 4, no. 2, pp. 27–39, 2014.

[99] R. Garibay-Martinez, G. Nelissen, L. L. Ferreira, and L. M. Pinho, "Task Partitioning and Priority Assignment for Hard Real-time Distributed Systems," *Journal of Computer and System Sciences*, vol. 81, no. 8, pp. 1542–1555, 2013.

[100] M. Liu, M. Becker, M. Behnam, and T. Nolte, "A dependency-graph based priority assignment algorithm for real-time traffic over nocs with shared virtual-channels," in *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, May 2016, pp. 1–8.

[101] Y. Liu, I. Taniguchi, H. Tomiyama, and L. Meng, "List Scheduling Strategies for Task Graphs with Data Parallelism," *2013 First International Symposium on Computing and Networking*, pp. 168–172, 2013.

[102] J. Decker and J. Schneider, "Heuristic Scheduling of Grid Workflows Supporting Co-Allocation and Advance Reservation," *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pp. 335–342, 2007.

[103] S. W. Kim and H. S. Park, "Periodic task scheduling algorithm under precedence constraint based on topological sort," in *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2011.

[104] Y. Kwok and I. Ahmad, "Bubble scheduling: A quasi dynamic algorithm for static allocation of tasks to parallel architectures," *Proceedings.Seventh IEEE Symposium on Parallel and Distributed Processing*, pp. 36–43, 1995.

[105] ——, "Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors," *Cluster Computing*, 2000.

[106] O. Sinnen and L. Sousa, "List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures," *Parallel Computing*, vol. 30, no. 1, pp. 81–101, 2004.

[107] J. J. Han and Q. H. Li, *Grid and Cooperative Computing: Second International Workshop, GCC 2003, Shanghai, China, December 7-10, 2003, Revised Papers, Part II*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, ch. A New Task Scheduling Algorithm in Distributed Computing Environments, pp. 141–144.

[108] C. Roig, A. Ripoll, M. A. Senar, F. Guirado, and E. Luque, "A new model for static mapping of parallel applications with task and data parallelism," in *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*. IEEE, 2001, pp. 8–pp.

[109] F. Sandnes and O. Sinnen, "Stochastic DFS for multiprocessor scheduling of cyclic taskgraphs," *Lecture Notes in Computer Science*, vol. 3320, pp. 354–362, 2004.

[110] D. B. Johnson, "Finding all the elementary circuits of a directed graph," *SIAM Journal on Computing*, vol. 4, no. 1, pp. 77–84, 1975.

[111] P. Eades, X. Lin, and W. F. Smyth, "A fast and effective heuristic for the feedback arc set problem," *Information Processing Letters*, vol. 47, no. 6, pp. 319–323, 1993.

[112] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[113] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of embedded software from synchronous dataflow specifications," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 21, no. 2, pp. 151–166, 1999.

[114] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *Proceedings of the 43rd annual Design Automation Conference*. ACM, 2006, pp. 899–904.

[115] H. Blumer, "Collective behavior," *New outline of the principles of sociology*, pp. 166–222, 1951.

[116] D. M. Gordon, "The ecology of collective behavior," *PLoS biology*, vol. 12, no. 3, p. e1001805, 2014.

[117] M. Yomosa, T. Mizuguchi, G. Vásárhelyi, and M. Nagy, "Coordinated behaviour in pigeon flocks," *PloS one*, vol. 10, no. 10, p. e0140558, 2015.

[118] S. Garnier, M. Combe, C. Jost, and G. Theraulaz, "Do ants need to estimate the geometrical properties of trail bifurcations to find an efficient route? a swarm robotics test bed," *PLoS computational biology*, vol. 9, no. 3, p. e1002903, 2013.

[119] N. J. Smelser, *Theory of collective behavior*. Quid Pro Books, 2011.

[120] D. A. Locher, *Collective behavior*. Prentice Hall Upper Saddle River, NJ, 2002.

[121] D. L. Miller, *Introduction to collective behavior and collective action*. Waveland Press, 2013.

[122] S. Schneider and L. McNally, "Seasonal patterns of foraging activity in colonies of the african honey bee, apis mellifera scutellata, in africa," *Insectes Sociaux*, vol. 39, no. 2, pp. 181–193, 1992.

[123] E. E. Boydston, T. L. Morelli, and K. E. Holekamp, "Sex differences in territorial behavior exhibited by the spotted hyena (hyaenidae, crocuta crocuta)," *Ethology*, vol. 107, no. 5, pp. 369–385, 2001.

[124] J. K. Parrish and L. Edelstein-Keshet, "Complexity, pattern, and evolutionary trade-offs in animal aggregation," *Science*, vol. 284, no. 5411, pp. 99–101, 1999.

[125] R. Ni, J. G. Puckett, E. R. Dufresne, and N. T. Ouellette, "Intrinsic fluctuations and driven response of insect swarms," *Physical review letters*, vol. 115, no. 11, p. 118104, 2015.

[126] J. E. Herbert-Read, A. Perna, R. P. Mann, T. M. Schaerf, D. J. Sumpter, and A. J. Ward, "Inferring the rules of interaction of shoaling fish," *Proceedings of the National Academy of Sciences*, vol. 108, no. 46, pp. 18 726–18 731, 2011.

[127] A. Arif, K. Shanahan, F.-J. Chou, Y. Dosouto, K. Starbird, and E. S. Spiro, "How information snowballs: Exploring the role of exposure in online rumor propagation," in *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 2016, pp. 466–477.

[128] A. L. Hughes, L. A. St Denis, L. Palen, and K. M. Anderson, "Online public communications by police & fire services during the 2012 hurricane sandy," in *Proceedings of the 32nd annual ACM conference on Human factors in computing systems*. ACM, 2014, pp. 1505–1514.

[129] P. A. Kenny, C. M. Nelson, and M. J. Bissell, "The ecology of tumors: By perturbing the microenvironment, wounds and infection may be key to tumor development," *Scientist (Philadelphia, Pa.)*, vol. 20, no. 4, p. 30, 2006.

[130] A. J. Ewald, A. Brenot, M. Duong, B. S. Chan, and Z. Werb, "Collective epithelial migration and cell rearrangements drive mammary branching morphogenesis," *Developmental cell*, vol. 14, no. 4, pp. 570–581, 2008.

[131] F.-C. Bidard, J.-Y. Pierga, A. Vincent-Salomon, and M.-F. Poupon, "A class action against the microenvironment: do cancer cells cooperate in metastasis?" *Cancer and Metastasis Reviews*, vol. 27, no. 1, pp. 5–10, 2008.

[132] N. W. Bode, A. J. Wood, and D. W. Franks, "Social networks and models for collective motion in animals," *Behavioral ecology and sociobiology*, vol. 65, no. 2, pp. 117–130, 2011.

[133] Y. Yao, Q. Fu, W. Yang, Y. Wang, and C. Sheng, "An epidemic model of computer worms with time delay and variable infection rate," *Security and Communication Networks*, vol. 2018, 2018.

[134] A. Antonioni and A. Cardillo, "Coevolution of synchronization and cooperation in costly networked interactions," *Phys. Rev. Lett.*, vol. 118, p. 238301, Jun 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.118.238301

[135] F. Yang, R. Zhang, Y. Yao, and Y. Yuan, "Locating the propagation source on complex networks with propagation centrality algorithm," *Knowledge-Based Systems*, vol. 100, pp. 112–123, 2016.

[136] H. Sun, J. Liu, J. Huang, G. Wang, Z. Yang, Q. Song, and X. Jia, "Cenlp: A centrality-based label propagation algorithm for community detection in networks," *Physica A: Statistical Mechanics and its Applications*, vol. 436, pp. 767–780, 2015.

[137] S.-k. Kown, B. Jang, B.-D. Lee, Y. Do, H. Baek, and Y.-H. Choi, "Influence evaluation of centrality-based random scanning strategy on early worm propagation rate," in *International Workshop on Information Security Applications*. Springer, 2016, pp. 90–101.

[138] A. Zubiaga, M. Liakata, R. Procter, G. W. S. Hoi, and P. Tolmie, "Analysing how people orient to and spread rumours in social media by looking at conversational threads," *PloS one*, vol. 11, no. 3, 2016. [Online]. Available: https://figshare.com/articles/PHEME_rumour_scheme_dataset_journalism_use_case/2068650

[139] S. A. Melo, H. Sugimoto, J. T. O'Connell, N. Kato, A. Villanueva, A. Vidal, L. Qiu, E. Vitkin, L. T. Perelman, C. A. Melo *et al.*, "Cancer exosomes perform cell-independent microrna biogenesis and promote tumorigenesis," *Cancer cell*, vol. 26, no. 5, pp. 707–721, 2014.

[140] M. Rubenstein, C. Ahler, and R. Nagpal, "Kilobot: A low cost scalable robot system for collective behaviors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3293–3298.

[141] B. D. López, "Whistle characteristics in free-ranging bottlenose dolphins (tursiops truncatus) in the mediterranean sea: Influence of behaviour," *Mammalian Biology-Zeitschrift für Säugetierkunde*, vol. 76, no. 2, pp. 180 – 189, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1616504710000960

[142] D. J. MacKay, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[143] F. Webb, J. Blais, and R. Nash, "A cartographic history of spruce budworm outbreaks and aerial forest spraying in the atlantic region of north america, 1949–1959," *The Canadian Entomologist*, vol. 93, no. 5, pp. 360–379, 1961.

[144] F. Barile, J. Masthoff, and S. Rossi, "A detailed analysis of the impact of tie strength and conflicts on social influence," in *UMAP 2017 - Adjunct Publication of the 25th Conference on User Modeling, Adaptation and Personalization*, Bratislava, Slovakia, 2017, pp. 227 – 230. [Online]. Available: http://dx.doi.org/10.1145/3099023.3099056

[145] V. Arnaboldi, M. Conti, A. Passarella, and R. I. Dunbar, "Online social networks and information diffusion: The role of ego networks," *Online Social Networks and Media*, vol. 1, pp. 44–55, 2017.

[146] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "A tuple space for data sharing in robot swarms," in *BICT 2015 - 9th EAI International Conference on Bio-Inspired Information and Communications Technologies*, New York City, NY, United states, 2015.

[147] B. Prideaux and C. Campus, "Bridging the gap between academic research and industry research needs," *James Cook University, Cairns Campus*, 2012.

[148] A. Hern, "Facebook launches aquila solar-powered drone for internet access," *The Guardian*, vol. 30, 2015.

APPENDIX A IDENTIFICATION OF RELEVANT SUBSPACES BASED ON LOCAL INTRINSIC DIMENSIONALITY

During the time spent in the laboratory of Pr. Micheal Houle in Tokyo, Japan, under the National Institute of Informatics (NII) Internship program, our research exploration took a deeper dive into an inherent property to data, its intrinsic dimensionality. More particularly, the research interest of Pr. Houle focused on local intrinsic dimensionality which recent findings triggered the work accomplished during this stay. The appendix provides a brief glimpse into the context/motivation of the work and the efforts given to achieve these goals.

Subspace Clustering and Intrinsic Dimensionality

Subspace clustering differs from classical clustering in that, in addition to cluster discovery, a step essential to subspace clustering is detecting the subspaces in which data points form the clusters; referred to as cluster/instances subspace preference. Previous work focused on either arbitrarily-oriented subspaces which detection requires some kind of heuristic; or axis-parallel subspaces which shrinks the search space to 2^D different subspaces of a space with D dimensions. Whilst techniques based on the former perform better in terms of finding subspaces that best fit the data, they may suffer from over-fitting and outputs transformations of the features that are difficult to interpret (this approach is least desirable in the fields of bio-informatics (gene annotation, micro-array analysis) and chemistry (mass spectra analysis, interpretability of molecular signatures), etc. which require results that are interpretable in terms of the original feature set). The real features can be projected to axis-parallel subspaces which makes them more appealing in these fields.

Given the recent advances in the characterization of data sets according to local intrinsic dimensionality (ID), our aim was to propose an approach to detect axis-parallel subspaces based on ID and develop a heuristic for subspace clustering. The developed approach show immense potential when it comes to accurately identifying the subspace with relevant information compared to existing methodologies both in terms of accuracy and the number of parameters to tune.

The research carried out here sparked and established the first step towards attaining the future endeavor of discovering the main influencers of emergent behaviors in swarms of robots and more importantly, the factors that drive a swarm to transition from one behavior to another.

APPENDIX B RESOURCE OPTIMIZATION OF LARGE COMPLEX REAL-TIME SYSTEMS

This appendix addresses a couple contributions to the optimization community which focused on the optimization of large complex system resources with timing constraints in terms of allocation and scheduling. The work developed under our collaboration with an industrial partner to optimize full-mission simulators which are complex software avionics systems. The main orchestrator of these works is Rabeh AYARI. My co-author contributions involved: (i) providing advice and recommendations on optimization approaches (ii) Handling the integration of simulator in the implementation and experiments (iii) writing and preparing some figures (iv) reviewing/editing.

SIMULATION-BASED SCHEDULABILITY ASSESSMENT FOR REAL-TIME SYSTEMS

Real-time systems not only require functional correctness, but also specific timing properties. Correct timing is especially challenging for hard real-time systems such as in medicine, avionics, and space industries, where missing a deadline can lead to catastrophic failure. A number of theories tackled this issue to determine whether a set of tasks running on a given architecture meets its timing constraints. One technique is schedulability analysis, which can provide guarantees for the timing behavior for a set of tasks. However, the use of schedulability tests involve an intrinsic amount of pessimism, which greatly reduces the number of configurations that can be considered as schedulable. This removes potentially promising system configurations from the task allocation optimization process, thereby reducing the quality of the final result. The aim of this paper is to overcome this limitation in the context of heterogeneous multiprocessor architectures. We propose a simulation-based approach to assess solutions discarded by a schedulability test, and include them in the optimization process. We tested our method on the optimization of the communication cost of a set of tasks scheduled on a quad core architecture, showing an improvement of up 11% when compared to the use of a schedulability test.

Authors: R. Ayari, **I. Hafnaoui**, G. Beltrame and G. Nicolescu
URL: http://dl.acm.org/citation.cfm?id=3015574.3015604

MULTI-OBJECTIVE MAPPING OF FULL-MISSION SIMULATORS ON HETEROGENEOUS DISTRIBUTED MULTI-PROCESSOR SYSTEMS

Full-mission simulators (FMSs) are considered the most critical simulation tool belonging to the flight simulator family. FMSs include a faithful reproduction of fighter aircraft. They are used by armed forces for design, training, and investigation purposes. Due to the criticality of their timing constraints and the high computation cost of the whole simulation, FMSs need to run in a high-performance computing system. Heterogeneous distributed systems are among the leading computing platforms and can guarantee a significant increase in performance by providing a large number of parallel powerful execution resources. One of the most persistent challenges raised by these platforms is the difficulty of finding an optimal mapping of n tasks on m processing elements. The mapping problem is considered a variant of the quadratic assignment problem, in which an exhaustive search cannot be performed. The mapping problem is an NP-hard problem and solving it requires the use of meta-heuristics, and it becomes more challenging when one has to optimize more than one objective with respect to the timing constraints. Multi-objective evolutionary algorithms have proven their efficiency when tackling this problem. Most of the existent works deal with the task mapping by considering either a single objective or homogeneous architectures. Therefore, the main contribution of this paper is a framework based on the model-driven design paradigm allowing us to map a set of intercommunicating real-time tasks making up the FMS model onto the heterogeneous distributed multi-processor system model. We propose a multi-objective approach based on the well-known optimization algorithm "Non-dominated Sorting Genetic Algorithm-II" satisfying the tight timing constraints of the simulation and minimizing makespan, communication cost, and memory consumption simultaneously.

Authors: R. Ayari, **I. Hafnaoui**, A. Aguiar, P. Gilbert, M. Galibois, J.P. Rousseau, G. Beltrame and G. Nicolescu

DOI: https://doi.org/10.1177/1548512916657907

APPENDIX C TIMING ANALYSIS WITH A PERMANENT FAULT DETECTION MECHANISM

This last appendix addresses the work done to develop an SPTA technique that considers the effects of permanent faults in the estimation of pWCET. The main contributor to this work is Choa CHEN. My co-author involvement covers: (i) writing the introduction (ii) reviewing and editing the paper.

STATIC PROBABILISTIC TIMING ANALYSIS WITH A PERMANENT FAULT DETECTION MECHANISM

In recent years, random caches have been proposed as a way to simplify the timing analysis of real-time systems. However, technology-scaling makes caches prone to faults. Fault detection mechanisms can detect permanent faults but they affect the timing analysis of a random cache. This paper introduces a Static Probabilistic Timing Analysis (SPTA) technique that accounts for a permanent fault detection mechanism. The permanent fault detection mechanism periodically checks caches for faults and disables faulty cache blocks to prevent future accesses. The SPTA method operates by periodically switching its runtime between the fault-detection and the no-fault-detection states. This is the first SPTA with a realistic permanent fault detection mechanism. Experiments show that the proposed method always provides safe timing estimations-even when few memory blocks are provided-and accurate results-when sufficient memory blocks are present.

Authors: C. Chen, J. Panerati, **I. Hafnaoui** and G. Beltrame
DOI: https://doi.org/10.1109/SIES.2017.7993373