THÈSE

Pour obtenir le grade de

Docteur

Délivré par **UNIVERSITÉ DE MONTPELLIER**

Préparée au sein de l'école doctorale
Information, Structures et Systèmes
Et de l'unité de recherche
LIRMM

Spécialité : **Systèmes Automatiques et Microélectronique**

Présentée par **Marcelo Grandi Mandelli**

EXPLORATION DE TECHNIQUES D'ALLOCATION DE TÂCHES DYNAMIQUES ET DISTRIBUÉES POUR MPSOCS DE LARGE ÉCHELLE

Soutenue le 13 juillet 2015 devant le jury composé de

Dr. Guy Gogniat	Pr., Université de Bretagne-Sud	Rapporteur
Dr. José Luís Güntzel	Pr., UFSC	Rapporteur
Dr. Michel Robert	Pr., Université de Montpellier	Président
Dr. Leandro Indrusiak	Pr., University of York	Examinateur
Dr. César Marcon	Pr., PUCRS	Examinateur
Dr. Luciano Ost	Pr., University of Leicester	CoDirecteur de Thèse
Dr. Fernando G. Moraes	Pr., PUCRS	Directeur de Thèse
Dr. Gilles Sassatelli	DR CNRS, Université de Montpellier	Directeur de Thèse

EXPLORATION OF RUNTIME DISTRIBUTED MAPPING TECHNIQUES FOR EMERGING LARGE SCALE MPSOCS

ABSTRACT

MPSoCs with hundreds of cores are already available in the market. According to the ITRS roadmap, such systems will integrate thousands of cores by the end of the decade. The definition of where each task will execute in the system is a major issue in the MPSoC design. In the literature, this issue is defined as task mapping. The growth in the number of cores increases the complexity of the task mapping. The main concerns in task mapping in large systems include: (i) scalability; (ii) dynamic workload; and (iii) reliability. It is necessary to distribute the mapping decision across the system to ensure scalability. The workload of emerging large MPSoCs may be dynamic, i.e., new applications may start at any moment, leading to different mapping scenarios. Therefore, it is necessary to execute the mapping process at runtime to support a dynamic workload. Reliability is tightly connected to the system workload distribution. Load imbalance may generate hotspots zones and consequently thermal implications, which may result in unreliable system operation. In large scale MPSoCs, reliability issues get worse since the growing number of cores on the same die increases power densities and, consequently, the system temperature. The literature presents different task mapping techniques to improve system reliability. However, such approaches use a centralized mapping approach, which are not scalable. To address these three challenges, the main goal of this Thesis is to propose and evaluate distributed mapping heuristics, executed at runtime, ensuring scalability and a fair workload distribution. Distributing the workload and the traffic inside the NoC increases the system reliability in long-term, due to the minimization of hotspot regions. To enable the design space exploration of large MPSoCs the first contribution of the Thesis lies in a multi-level modeling framework, which supports different models and debugging capabilities that enrich and facilitate the design of MPSoCs. The simulation of lower level models (e.g. RTL) generates performance parameters used to calibrate abstract models (e.g. untimed models). The abstract models pave the way to explore mapping heuristics in large systems. Most mapping techniques focus on optimizing communication volume in the NoC, which may compromise reliability due to overload processors. On the other hand, a heuristic optimizing only the workload distribution may overload NoC links, compromising its reliability. The second significant contribution of the Thesis is the proposition of dynamic and distributed mapping heuristics, making a tradeoff between communication volume (NoC links) and workload distribution (CPU usage). Results related to execution time, communication volume, energy consumption, power traces and temperature distribution in large MPSoCs (144 processors) confirm the tradeoff hypothesis. Trading off workload and communication volume improves system reliably through the reduction of hotspots regions, without compromising system performance.

Keywords: Modeling, System Management, Task Mapping, NoC, SoC, MPSoC.

EXPLORATION DE TECHNIQUES D'ALLOCATION DE TÂCHES DYNAMIQUES ET DISTRIBUÉES POUR MPSOCS DE LARGE ÉCHELLE

RÉSUMÉ

MPSoCs (systèmes multiprocesseurs sur puces) avec des centaines de cœurs sont déjà disponibles sur le marché. Selon le ITRS, ces systèmes intégreront des milliers de cœurs à la fin de la décennie. La définition du cœur, où chaque tâche sera exécutée dans le système, est une question majeure dans la conception de MPSoCs. Dans la littérature, cette question est définie comme allocation de tâches. La croissance du nombre de cœurs augmente la complexité de l'allocation de tâches. Les principales préoccupations en matière d'allocation de tâches dans des grands MPSoCs incluent: (i) l'évolutivité; (ii) la charge de travail dynamique; et (iii) la fiabilité. Il est nécessaire de distribuer la décision d'allocation de tâches à travers le système afin d'assurer l'évolutivité. La charge de travail de grands MPSoCs peut être dynamique, à savoir, de nouvelles applications peuvent commencer à tout moment, conduisant à différents scénarios d'allocation. Par conséquent, il est nécessaire d'exécuter le processus d'allocation à l'exécution pour soutenir une charge de travail dynamique. La fiabilité est étroitement liée à la distribution de la charge de travail du système. Un déséquilibre de charge peut générer des hotspots et autres implications thermiques, ce qui peut entraîner un fonctionnement peu fiable du système. Dans de grands MPSoCs, les problèmes de fiabilité empirent puisque l'augmentation du nombre de cœurs sur la même puce augmente la densité de puissance et, par conséquent, la température du système. La littérature présente différentes techniques d'allocation de tâches pour améliorer la fiabilité du système. Cependant, ces techniques utilisent des approches d'allocation centralisées, qui ne sont pas évolutives. Pour répondre à ces trois défis, l'objectif principal de cette Thèse est de proposer et évaluer des heuristiques d'allocation de tâches distribuées et dynamiques en assurant l'évolutivité et une distribution équitable de la charge de travail. Une distribution équitable de la charge de travail et du trafic du NoC (réseau sur puce) augmente la fiabilité du système dans le long terme, en raison de la minimisation des régions de hotspot. Pour permettre l'exploration de l'espace de conception de grands MPSoCs, la première contribution de cette Thèse se situe dans le cadre d'une modélisation multi-niveaux, qui prend en compte différents modèles et de capacités de débogage qui enrichissent et facilitent la conception des MPSoCs. La simulation de modèles de niveau inférieur (par exemple RTL) génère des paramètres de performance utilisés pour calibrer des modèles abstraits (sans précision d'horloge). Les modèles abstraits permettent d'explorer des heuristiques d'allocation de tâches dans de grands systèmes. La plupart des techniques d'allocation de tâches se focalisent sur l'optimisation du volume de communication, ce qui peut compromettre la fiabilité du système, en raison d'une surcharge des processeurs. D'autre part, une heuristique qui optimise seulement la distribution de la charge de travail peut surcharger le NoC et compromettre sa fiabilité. La deuxième contribution importante de cette Thèse est la proposition d'heuristiques d'allocation de tâches dynamiques et distribuées, qui réalisent un compromis entre le volume de communication (liens du NoC) et la distribution de la charge de travail (de l'utilisation des processeurs). Des résultats liés au temps d'exécution, au volume de la communication, à la consommation d'énergie, aux traces de puissance et à la distribution de la température dans les grands MPSoCs (144 processeurs) confirment l'hypothèse de compromis. Faire un compromis entre la réduction du volume de communication et une distribution équitable de la charge de travail améliore le système de manière fiable grâce à la réduction des régions de hotspots, sans compromettre la performance du système.

Mots clés : Modélisation, Gestion de MPSoCs, Allocation de Tâches, NoC, SoC, MPSoC.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API ..... Application programming interface
DTW ..... Digital Time Warping
DVFS..... Dynamic Voltage and Frequency Scaling
EC ..... Energy Consumption
EDK..... Embedded Development Kit
GALS..... Globally Asynchronous Locally Synchronous
GMP ..... Global Manager Processing Element
HeMPS..... Hermes Multiprocessor System
ISS ..... Instruction Set Simulator
L ..... Load task mapping heuristic
LC..... Load-Communication task mapping heuristic
LEC-DN ..... Low Energy Consumption - Dependences Neighborhood
LMP..... Local Manager Processing Element
MP..... Manager Processing Element
MPI..... Message Passing Interface
MPSoC ..... Multiprocessor System on Chip
MTTF..... Mean-Time To Failure
MWD ..... Multi-Window Display
NI ..... Network Interface
NN ..... Nearest Neighbor
NoC ..... Network-on-Chip
OS ..... Operating System
OVP..... Open Virtual Platform
PBD..... Platform-based Design
PE ..... Processing Element
PPM ..... OVP Peripherals models
PREMAP-DN ..... PREMAP - Dependences Neighborhood
QoS ..... Quality Of Service
SDK..... Software Development Kit
SoC ..... System-On-Chip
SP ..... Slave Processing Element
TE..... Total Energy consumed by a processing element
TLM ..... Transaction-Level Modeling
UART ..... Universal Asynchronous Receiver/Transmitter
VOPD ..... Video Object Plane Decoder
XY ..... XY Routing Algorithm

TABLE OF CONTENTS

1. INTRODUCTION

MPSoCs (Multiprocessor Systems-on-Chip) have been employed to provide the high demands of performance while maintaining energy efficiency during the execution of concurrent embedded applications (e.g. video compressing, wireless communication standards, gaming). Such systems increase performance by using multiple homogeneous or heterogeneous processors. MPSoCs also integrate memories, dedicated hardware cores, and a communication infrastructure to interconnect the system components, as NoCs (Networks-on-Chip) and buses. Despite the higher design complexity of NoCs, such communication infrastructure offers better scalability, performance and power capabilities when compared to buses [BEN02].

Applications designed to execute in MPSoCs may be partitioned into different tasks to execute in different cores, enabling their parallel execution [SIN13]. A task is a set of instructions and data, containing information and constraints for its correct execution in a given core. Additionally, tasks exchange data with other tasks during the execution of the application. The definition in which system core each task will execute is a major issue in the MPSoC design. In the literature, this issue is referred to *task mapping* [SIN13].

Task mapping decision have been executed at runtime in order to deal with time-varying workloads caused by the most part of embedded system applications [SIN10]. Such variations cannot be accurately predicted during design time, such as the scenarios when the system interacts with complex deployment environments or user-driven requests [CHO10]. Runtime approaches (also referred as online or dynamic mapping approaches) require simple and fast mapping solutions, since high time-consuming and high computational algorithms may compromise the system performance. Further, runtime mapping can better deal with other system changes during runtime, such as cores availability and defective cores [SIN13].

The increasing number of cores in MPSoCs also requires scalable and distributed mapping solutions. Novel large MPSoCs, with dozens of cores, are already present in market [INT12][TIL10] and the ITRS roadmap [ITR13] projects systems integrating thousands of cores by the end of the decade. In such systems, a centralized mapping decision compromises the system performance since a single core is responsible for processing and responding all mapping requests [FAR08]. Underlying solution contributes to increasing NoC congestion around the mapper leading to hotspot zones, which may result in system failures.

Reliability is an important concern related to task mapping, tightly connected to the workload distribution [CHA13][WAN14][HEN13]. Load imbalance decisions can generate hotspots zones (i.e. peaks of power dissipation) and thermal variations, which directly affects system reliability [CHA13][WAN14][MEY14]. This issue is worse in large MPSoCs,

which aggregate a growing number of cores on the same die, increasing power densities and, consequently, system temperature. Further, mapping communicating tasks far from each other results in more data transfer through the system, increasing communication latency and energy consumption. Higher data volume transferred through the system also induces link failures, which may produce unreachable zones (e.g. isolated and, consequently, unusable cores). Unusable cores induce mapping of applications onto other system cores, increasing their workload and, consequently, reducing their lifetime.

Dealing with dynamic and distributed mapping in large-scale systems requires efficient means to evaluate several scenarios considering different performance metrics. Simulation enables the evaluation of different task mapping solutions, as well as other MPSoCs challenges (e.g. quality of service management). However, the raising complexity of large MPSoCs restricts the adoption of RTL simulation due to its high simulation time, verification and debugging cost [ROT13]. For this purpose, high-level modeling techniques have been employed to boost system design and validation. High abstract models simplify system properties and characteristics using some formalism, preserving those characteristics and properties that are relevant for a given purpose (e.g. system energy consumption prediction) [JAN03]. In this context, a designer can employ different models to represent different system aspects, targeting specific goals.

Designers must use different models to accomplish both software and hardware design and validation process. While architectural-oriented design requires quasi-cycle accurate models [BIN11], software development demands high simulation speeds (e.g. 100 MIPS [OVP13]) [LEM12]. With such conflicting requirements, it is difficult to cover all modeling and simulation needs inherent to hardware and software design space exploration with one single model. Beyond that, to ensure correct functionality of emerging MPSoC embedded systems it is necessary to consider lower level design constraints, such as area, energy consumption, and temperature. To deal with such design constraints, an RTL description of the platform is necessary to acquire information through low-level models.

To overcome the challenges mentioned above, this Thesis proposes a multi-level platform framework, which combines different modeling techniques aiming to provide efficient means to explore large-scale MPSoCs, considering both hardware and software. The proposed framework includes high abstract models, providing great modeling and debuggability capabilities allied to high simulation speeds. Such abstract models are supported by a clock-cycle accurate implementation, which provides power and area figures. The proposed framework is used to develop and validate runtime distributed mapping solutions.

1.1 HYPOTHESIS TO BE DEMONSTRATED WITH THIS THESIS

This Thesis relies on two hypotheses: (*i*) untimed abstract models can be used to evaluate task mapping solutions; and (*ii*) task mapping solutions that focus on communication volume reduction can compromise system reliability; and task mapping solutions that focus on workload balancing can increase system communication volume.

1.2 GOALS

In order to address the hypotheses mentioned above, the strategic goal of this Thesis is first to combine different modeling techniques into a multi-level framework, targeting the investigation of task mapping strategies for MPSoCs. Such multi-level framework should support the design space exploration of large-scale MPSoCs (thousands of PEs, interconnected by a NoC executing dozens of applications), providing accurate performance evaluation in an acceptable time. Considering the ITRS perspective [ITR13], the demand for new modeling techniques and supporting tools (models definition, generation, and simulation) will continue growing, sustaining the importance and the relevance of this Thesis. The second strategic goal of this Thesis is to investigate different task mapping strategies, in order to define heuristics to reduce communication volume and provide a balanced workload distribution for large-scale MPSoCs. Further, in order to provide scalable task mapping decision, distributed system management strategies are investigated.

To accomplish these strategic goals, the following specific objectives should be fulfilled:

- adopt a stable MPSoC platform that will serve as reference in terms of clock-cycle accurate simulation and power and area results;
- propose higher level implementations of the reference platform, aiming at speeding up the design and the investigation of novel management strategies;
- propose an integrated design flow by providing semi-automated and easy to use toolset, considering concomitant hardware and software development;
- investigate distributed system management strategies in order to achieve scalable large-scale MPSoCs;
- Investigate different task mapping techniques, focusing on communication volume reduction and workload distribution balancing for large-scale MPSoCs.

1.3 ORIGINAL CONTRIBUTIONS

This Thesis has two main contributions: (*i*) proposition of a multi-level modeling framework, which supports different modeling and debugging capabilities that enrich and facilitate the design of large MPSoCs; *(ii*) proposal of scalable and distributed lightweight runtime mapping techniques for large-scale MPSoCs. As summary, such contributions can be detailed as follows:

- The multi-level platform framework uses the reference NoC-based MPSoC model presented in Section 3.1. This model is implemented in synthesizable RTL VHDL, which has main advantage at being synthesizable, allowing to captures accurate area, frequency and power performance figures. Debug facilities include waveforms and assertions, targeting hardware development, not software development. In this context, this Thesis contributes with two new models:
    - a SystemC RTL model, presented in Section 3.2, which enables the simulation of larger systems in a reasonable simulation time but still providing high accurate performance results. Some improvements in terms of debuggability are achieved, e.g., by inserting debug coded in the ISS.
    - an OVP [OVP13] (i.e. Open Virtual Platform) model, presented in Section 3.3. OVPSim is a virtual platform and modeling framework proposed by Imperas, aiming to accelerate the development of embedded software, specifically for SoCs and MPSoCs. The proposed model is used to develop and validate task mapping heuristics, presented in Chapter 5.
- Task mapping contributions of this Thesis are described as follows:
    - Proposal of a distributed management architecture that provides the necessary features to include scalable mapping solutions, described in Chapter 4;
    - Modification of formerly proposed task mapping heuristics [MAN11b] to support decentralized mapping decisions, presented in Chapter 5;
    - Proposal of profiler platform to generate performance information (inter-task communication volume, energy consumption of each task) to guide the runtime mapping process, presented in Section 4.1;
    - Proposal of a novel scalable runtime distributed energy-aware mapping technique, which focuses on workload balancing and communication volume reduction (presented in Chapter 5);
    - Validation of the proposed mapping techniques with different and large scenarios (Section 5.5), considering the number of executed instructions, power, energy, temperature, and execution time.

1.4 ORIGINALITY OF THIS THESIS

The originality of this Thesis relies on a novel a scalable and lightweight runtime distributed energy-aware mapping technique, aiming to reduce communication volume and balance workload distribution in large-scale systems. This mapping heuristic is validated in a multi-layer model approach, which provides flexibility in the system design by using different abstraction levels.

1.5 STRUCTURE OF THIS DOCUMENT

The remaining of this document is organized as follows: Chapter 2 presents the state-of-art on MPSoC platforms and task mapping solutions. Next, Chapter 3 presents the proposed multi-level framework. Chapter 4 describes the proposed distributed system management approach, comparing it with a centralized approach. Chapter 5 presents different distributed runtime task mapping heuristics, including a heuristic aiming to reduce communication volume and a balanced workload distribution. Finally, Chapter 6 provides conclusions and directions for future work.

2. STATE-OF-ART

This Chapter discusses the state-of-art related to the main contributions of this Thesis. Section 2.1 presents works underlying MPSoC platforms development, including different modeling approaches and specific design goals. Section 2.2 first introduces a task mapping taxonomy and, then, focuses on the state-of-art of runtime mapping approaches.

2.1 State-of-Art in MPSoC platforms

The literature contains examples of MPSoC platforms developed in different abstraction levels, differing in terms of accuracy, simulation cost and design flexibility.

Some works [CEN09][VEN10][LEM12][REK13][ZHA13][DUE14] use high abstraction models for MPSoC design exploration. Ceng et al. [CEN09] present a framework aiming at improving application development at early design stage, in which target platform details are not already determined. The proposed framework, called High-level Virtual Platform (HVP), includes a simulator that abstracts hardware (i.e. processor elements) and software (i.e. OS, communication API) details of the target MPSoC platform. Such simulator is built using SystemC, including processing elements, interconnection infrastructure and peripherals. Processing elements are modeled as Virtual Processing Elements (VPEs). A VPE comprises a high-level processor model and an operating system, which is responsible for task execution control, such as scheduling and execution speed. Both VPEs and tasks are implemented using SystemC, interacting through TLM channels between them. Communication interconnection between VPEs is done through generic shared memories, which can be controlled and accessed in applications by the proposed programming API. Further, each VPE includes a set of software tools for application programming that is completely platform independent, enabling reusable code for different platforms. Such tools include the previously mentioned programming APIs, which enable inter-task communication/synchronization and VPE interactions. Debugging facilities include connection with host debuggers, such as GDB.

Ventroux et al. [VEN10] present the SESAM, a NoC-based MPSoC framework targeting the design and the exploration of asymmetric multiprocessor systems. The platform in SESAM is described in SystemC TLM, while the programming model is based on the explicit separation of control and computation parts. SESAM has a library of components like NoCs (e.g. multibus, torus, ring and mesh) that are modeled in approximate-timed TLM, enabling fast and accurate simulation. A Hardware Abstraction Level (HAL) is provided to manage all memory accesses and dynamic memory allocation. To debug the platform it is possible to use a GNU GDB implementation. The Authors point out 90% accuracy compared to a fully cycle-accurate simulator.

Lemaire et al. [LEM12] present a flexible simulation environment integrating different modeling techniques for design and exploration of SoCs. The proposed environment uses the GENEPY MPSoC as base model, which contains high-performance DSP processors, general-purpose processors and dedicated hardware interconnected by a GALS NoC. The NoC is modeled in SystemC/TLM with 3 different modes: loosely-timed packet-level mode, ignoring NoC contention; approximately-timed packet-level mode with a contention model; and an accurate flit-level mode, which implementation is very close to the real hardware. Such NoC models can integrate different CPU core models, also modeled in three different abstraction levels: Host Code Execution (HCE) model, in which application code is compiled and directly linked to SystemC/TLM platform; an Instruction Set Simulator (ISS) model; and an RTL model. A SystemC power model including DVFS is integrated to the proposed environment. Authors claim the proposed environment enables software development and hardware implementation, providing from fast simulation to low-level hardware models.

In [REK13] OVP is used to model inter-processor communication in shared memory MPSoCs. Authors claim OVP only allows processors communication through shared memory. However, this Thesis presents distributed memory MPSoC platform modeled In OVP (see Section 3.3). In order to evaluate the benefits of OVP, Authors present different homogeneous and heterogeneous platform configurations and demonstrates the integration of operating systems (i.e. Linux). In this work, no comparisons with other simulation models (e.g. SystemC) were made.

Zhang et al. [ZHA13] present a modeling tool, MCVP-NoC (Many-Core Virtual Platform with Networks-on-Chip), which is designed to explore large-scale MPSoCs. MCVP-NoC is developed in TLM-SystemC with an OVP layer to provide fast processor models simulation, memory models and bus decoder. Orion2 software was integrated into MCVP-NoC to report power and area estimation values. TLM2.0 interface connects all component models. Authors report that the simulation using a virtual platform can speed up to 40 times the RTL simulation.

Duenha et al. [DUE14] propose a simulation toolset for development and evaluation of MPSoCs. The proposed toolset, called MPSoCBench, supports up to 64 processor cores of four different architectures (PowerPC, MIPS, SARC and ARM). ArchC [AZE05] is used to generate processors models in SystemC/C++. Further, processors have configurable cache models and may be interconnected by different communication infrastructure (crossbar, NoC) described in SystemC/TLM. MPSoCBench does not support operating system, using a POSIX PThread emulation library to handle thread management, barriers, mutual exclusion, semaphores, and conditional variables. Performance analysis include: power estimation based in different FPGAs from Xilinx and Altera; number of instructions per core; number of memories reads and writes; number of hops in NoC communication; simulation time; application correctness. Authors claim that

although the proposed model is not cycle-accurate, users can incorporate timing information to provide a degree of accuracy, such as time per instruction in processor models.

Indrusiak [IND14] proposes analytical methods to evaluate whether applications meet their timing constraints on homogeneous NoC-based multiprocessor architectures. The proposed work uses a system model covering different applications and NoC-based platform configurations. Experiments compare the proposed method with simulation methods, verifying figures for computation and communication response times.

Other works propose the use of EDKs (Embedded Development Kit) to automate FPGA-based MPSoC design and emulation. Lukovic et al. [LUK08] propose an automatic MPSoC generation using the Xilinx EDK, allowing fast hardware redesign. The work of Meier et al. [MEI10] presents a platform-based design model that reduces the MPSoC design complexity using the LavA framework. Tian et al. [TIA09] uses an FPGA implementation to evaluate the efficiency of data synchronization in a NoC. Benini et al. [BEN12] present a platform with an SDK (Software Development Kit) able to execute different programming models at an abstract level of hardware.

MPSoCs described in synthesizable RTL are also found in the literature. Paulin et al. [PAU06] propose a deadline evaluation in an RTL model MPSoC. Ngouanga et al. [NGO06] use a NoC-based platform developed in synthesizable VHDL for task mapping heuristic validation. Validations use up to a 6x6 mesh system, taking place by either VHDL simulation or by FPGA prototyping. Busseuil et al. [BUS11] present an RTL distributed memory platform for design space exploration of NoC-based MPSoCs. The proposed platform is totally validated in FGPA, providing a set of functions and services to enable different performance analysis.

Table 1 compares relevant platform characteristics of the reviewed works. As most proposals, this work models a NoC-based MPSoC. Our contribution distinguishes itself from all previous works mentioned in this section by combining fast and accurate modeling and simulation capabilities in one single software development flow, including: cycle-accurate model (VHDL and RTL-SystemC/ISS) and approximated-time model (OVP, which uses C language). The OVP model provides performance estimates, using data obtained from the RTL simulation. In this direction, complex software stacks can be successively refined in different platform models until an adequate development is achieved in terms of functionality and accuracy.

Table 1 – MPSoC platforms.

Proposal	Interconnection	Description Language or design method	Debugging	Accuracy and evaluation
Paulin et al. (2006)	NoC	RTL synthesizable	N/A	Clock-cycle accurate
Ngouanga et al. (2006)	NoC	RTL synthesizable	N/A	Clock-cycle accurate
Lukovic et al. (2008)	NoC	PBD (Xilinx EDK)	Xilinx EDK Tools	Untimed
Tian et al. (2009)	NoC	PBD (Xilinx-4 FX140)	EDK Tools	Untimed
Ceng et al. (2009)	Bus	SystemC/TLM	GNU GDB	Untimed
Ventroux et al. (2010)	NoC	TLM-SystemC	GNU GDB	Approximate-timed TLM
Meier et al. (2010)	Bus	PBD (Virtex-V LX 110T EDK)	EDK Tools	Untimed
Busseuil (2011)	NoC	RTL synthesizable	N/A	Clock-cycle accurate
Benini et al. (2012)	NoC (Cluster)/Bus	PBD (Xilinx Zynq)	EDK Tools	Untimed
Lemaire et al. (2012)	NoC	RTL-SystemC and TLM	N/A	7% compared to RTL
Rekik et al. (2013)	Bus	OVP	N/A	Untimed
Zhang et al. (2013)	NoC	TLM-SystemC with OVP and RTL	N/A	40 times faster than RTL
Duenha et al. (2014)	NoC	SystemC/TLM	N/A	Approximate-timed TLM
Indrusiak (2014)	NoC	analytical methods	N/A	Untimed
This proposal	**NoC**	**RTL-VHDL, RTL-SystemC/ISS, and OVP**	**OVP tools / performance reports**	**Clock-cycle accurate / approximate-timed**

2.2 State-of-Art in Dynamic Task Mapping

Task mapping literature is wide, requiring a taxonomy classification considering different mapping criteria. In this context, this Thesis classifies [MAN11b][OST13] the mapping process according to four criteria: (*i*) the target architecture; (*ii*) the number of tasks per PE; (*iii*) the moment in which it is executed; (*iv*) system management approach.

According to the target architecture, task mapping can be performed in homogeneous (identical PEs) or heterogeneous (e.g. DSPs, dedicated IPs, accelerators) systems. The complexity of task mapping is higher when heterogeneous MPSoC platforms

are employed, since PE type must be considered in mapping process. Regarding the number of tasks mapped per PEs, mapping approaches can be classified as single or multi-task. Single-task assumes only one task assignment per PE while multi-task allows mapping more than one task per PE according to some criteria (e.g. communication, execution time, task deadlines). A multi-task approach can better explore system resources, enabling the execution of an increasing number of applications in parallel.

The mapping process can be defined at design-time or runtime. When task mapping is defined at design-time (also referred as offline or static mapping approach), all applications that will be executed in the system must be known a priori. Such approach can explore high time-consuming and computational intensive algorithms to better evaluate the mapping solutions. These algorithms use a global view of the system state to define the mapping of all its tasks, which improves the mapping quality. However, design time approaches are not suitable for dynamic and unpredictable workloads imposed by the execution of different applications [CHO08][HÖL08][SCH10][SIN10]. Further, such approaches are not able to cope with system changes, such as cores failures incurred in the system during runtime [SIN13].

Runtime task mapping approaches (also referred as online or dynamic mapping approaches) have been used to improve system adaptability. Such approach enables different applications to be inserted into the system at runtime, enabling dynamic workloads. Runtime task mapping is incorporated in the system management scheme, which verifies system conditions and takes decisions to improve performance. In this context, whenever mapping an application task, the system conditions are verified to achieve better mapping decisions. The time to take mapping decisions must be taken into account since it can impact on applications execution time [SIN13].

As mentioned before, runtime task mapping is one of the system management functions, which also include monitoring, task migration, DVFS. System management can be classified in centralized or distributed. Centralized management uses a single core (called central manager) responsible for the overall management, which is suited for small MPSoCs due to scalability issues. For example, such single core can be overloaded very quickly, due to the computational effort to execute all management functions, such as mapping actions and treatment of monitoring events. In addition, the traffic around the central manager induces a communication hotspot, compromising system performance. A central manager also compromises system reliability, since it creates a single point of failure in the system. In a distributed approach, the system management is distributed in different cores, increasing system scalability and reliability.

This Thesis focuses on general-purpose MPSoC platforms, able to execute several applications that are unknown in advance. This Thesis also assumes that the underlying applications can be inserted in the system in a non-deterministic way, according to user

requirements. The resulting scenario points to the exploration of runtime mapping approaches. Different research on runtime mapping approaches have been proposed. Smit et al. [SMI05] propose an iterative hierarchical dynamic mapping approach, which aims to reduce energy consumption while providing the required quality-of-service (QoS). In such approach, tasks are firstly grouped by assigning to a system resource type (e.g. FPGA, DSP, ARM), according to their performance constraints. Then, each task inside a group is mapped, minimizing the distance between them and reducing communication cost. Finally, the resulting mapping is checked, and if it does not meet the application requirements, a new iteration is required. Ngouanga et al. [NGO06] present a force-directed mapping heuristic for homogeneous NoC-based MPSoCs. The heuristic selects the new position for a task according to a resulting force, which is proportional to the communication volume and distance between tasks.

Chou et al. [CHO07] introduce an incremental dynamic mapping process approach, where PEs connected to the NoC have multiple voltage levels while the network has its voltage–frequency domain. A global manager (OS-controlled mechanism) is responsible for finding a contiguous area to map an application, and for defining the position of the tasks within this area, as well. According to the Authors, the strategy avoids the fragmentation of the system and aims to minimize communication energy consumption, which is calculated according to the approach proposed by Ye et al. [YE02]. The proposed approach was extended in [CHO08], incorporating the user behavior information in the mapping process. The user behavior corresponds to the application profile data, including the application periodicity in the system and data volume transferred among tasks. For real applications considering the user behavior information, the approach achieved around 60% energy savings compared to a random allocation scenario.

Hölzenspies et al. [HÖL07][HÖL08] investigate a runtime spatial mapping technique with real-time requirements, considering streaming applications mapped onto heterogeneous MPSoCs. In the proposed work, the application remapping is determined according to a set of information (i.e. latency/throughput) collected at design time, aiming to satisfy QoS requirements, as well as to optimize the resources usage and to minimize the energy consumption. A similar approach is proposed by Schranzhofer et al. [SCH10], merging pre-computed template mappings (defined at design time) and online decisions that define newly arriving tasks to the PEs at runtime. Compared to static mapping approaches, the obtained results reveal that it is possible to achieve an average reduction on power dissipation of 40% - 45%, while keeping the introduced overhead to store the template mappings as low as 1kB. Wildermann et al. [WIL09] present another power-aware mapping approach. This approach employs a heuristic that includes a neighborhood metric inspired by rules from Cellular Automata, which allows decreasing the communication overhead and, consequently, the energy consumption imposed by dynamic applications.

Coskun et al. [COS07] propose a task mapping technique aiming to eliminate hotspots and to reduce spatial and temporal temperature variations. The thermal history of the cores obtained from HotSpot [WEI06] is used as basis for the mapping decisions. The proposed technique is extended in [COS09] considering 3D multicore architectures, aiming to reduce thermal problems with low overhead. The experimental setup uses a 3D multicore UltraSPARC TI system, with eight cores. Results show that the proposed technique reduces the frequency of hotspots, spatial gradients, and thermal cycles. The proposed technique can be also combined with DVFS, improving the reduction of hotspots by 20%-40% compared to a DVFS only approach.

In [HOS09] a stochastic dynamic task mapping and a routing algorithm are used to minimize the reconfiguration overhead. Lu et al. [LU10] propose a dynamic mapping algorithm, called Rotating Mapping Algorithm (RMA), which aims to reduce the overall traffic congestion (taking into account the buffer space) and communication energy consumption of applications (reduction of transmission hops between tasks).

Huang et al. [HUA09] present a lifetime-aware task mapping and scheduling strategies based on simulated annealing. The proposed approach uses an analytical model to estimate lifetime reliability, considering the processors' temperature variation aggregated to its voltage/frequency to obtain the system MTTF. Experiments are conducted based on a set of synthetic applications including from 20 to 260 tasks each, which are executed in an abstract MPSoC platform. Simulated scenarios use up to 8 processors. The authors relax deadline constraints by 10%, obtaining lifetime improvements ranging from 16.9% to 20.6%.

Carvalho et al. [CAR10] evaluate pros and cons of using dynamic mapping heuristics (path load and best neighbor heuristics), when compared to static ones (e.g. simulated annealing and Taboo search). The Authors adopted energy consumption and latency as performance metrics, which were evaluated in a heterogeneous NoC-based MPSoC model (RTL NoC and abstract PEs implemented in System-C), regarding different application scenarios. Singh et al. [SIN09a][SIN09b][SIN10] extended the dynamic heuristics proposed by Carvalho et al. [CAR10] to support multi-task mapping. A clustering approach is proposed, which tries to maximize the number of communicating tasks in the same PE. This technique verifies the previously mapped tasks in a given a PE to map a new ones on it: if the required task communicates with some previously mapped task, it is mapped; if not, then another PE is verified. The Authors mention that some PEs may receive only one task, underusing the system resources. The clustering approach, compared to a non-clustering approach, improves in average 15% the channel load and energy consumption, with some improvement in packet latency and execution time.

Khajekarimi et al. [KHA12] present a runtime task mapping approach aiming to reduce network communication and congestion in NoC-based heterogeneous MPSoCS.

First, the approach maps the initial tasks of an application. The initial task mapping aims to group the application tasks in adjacent nodes or as close as possible. This is done by choosing a PE that has a number of idle neighbors equal to the number of communicating tasks with the required initial task and distance among these neighbors is minimal. Then, the other application tasks are mapped using a heuristic based on BN (Best Neighbor heuristic) [CAR10]. As BN, this heuristic tries to approximate a pair of communicating tasks using the network path with the lowest traffic. It also evaluates if a PE has a number of idle neighbors equal to the number of tasks communicating with the required task. Experiments are evaluated with the Noxim Simulator [FAZ08] using a 4x4 mesh topology and real applications. Results show that the proposed approach achieves an energy reduction of 15% compared to BN and 23% compared to CE [HU10].

Hartman et al. [HAR12] present a runtime task mapping approach aiming to increase the system lifetime. The proposed approach assumes that each processor has a wear sensor that captures system data (e.g. processor usage). Based on captured information, the mapping is defined considering changes in wear patterns in the system. Real and synthetic applications are used to validate the proposed task mapping, which improves the lifetime of 7.1% on average when compared to temperature-based heuristics.

Chantem et al. [CHA13] propose both dynamic task mapping and scheduling to improve system reliability. This work adopts the LTF-based algorithm [CHE08] for dynamic task mapping, which tries to balance processors load assigning the larger tasks to least worn cores. Then, a scheduling technique is executed periodically, depending on system wear state conditions, trying to compensate uneven core wear states. Authors also analytically determine thermal profiles for a given workload, aiming to maximize system lifetime. Experimental results use a bus-based system with 4 or 9 homogeneous cores, based on the Alpha 21264 processor. Benchmarks varying from 100 to 1000 tasks are used for evaluation. The proposed approach improves in 97% the system MTTF when compared to a thermal-aware algorithm.

Das et al. [DAS13] propose a task mapping technique that generates mapping solutions at design-time, aiming to satisfy application deadlines and to maximize the system lifetime (measured in terms of the MTTF of cores due to internal wear-outs and aging of NoC links). These solutions are stored in a database, which is used at runtime by a system manager that selects the best mapping solution for a given application. When an application is required, the entire set of system cores is dedicated to it, and the optimum mapping solution is fetched and applied. If one or more cores fail, the system restarts and the best solution for the reduced number of cores is selected. Experiments are evaluated using a range from 4 to 32 synthetic and real applications executed on an MPSoC platform configured with up to 8 homogeneous cores. The proposed solution is improved in [DAS14] by including an offline DVFS technique defining voltage and frequency levels of all cores. A model based on HotSpot [WEI06] is proposed to estimate the temperature of a

core to determine its aging impact. Experimental results are conducted using synthetic and real applications executed on a 3x3 mesh MPSoC platform, which assumes five voltage-frequency pairs for each core. The proposed technique achieves 40% energy savings and 6% increases in the lifetime compared to existing approaches.

Bolchini et al. [BOL13] present a runtime task mapping technique aiming to improve system reliability (MTTF) under energy and performance constraints. For this purpose, this work combines the technique of Das et al. [DAS13] with a remap strategy. Therefore, mapping solutions are also generated at design time, and system conditions at runtime determine their selection. Then, a remap strategy periodically verifies the system and re-assigns a part of the tasks to other cores, in order to improve lifetime. The authors report some results based on a NoC-based SystemC TLM many-core model using 3x3 and 3x4 mesh configurations. Results show the proposed technique improves lifetime by 16% with less than 10% communication energy overhead compared to the static approach.

Differently from other works, [FAR08][ANA12][CUI12][KOB11] and [WEI11] propose distributed dynamic mapping approaches. Al Faruque et al. [FAR08], Anagnostopoulos et al. [ANA12] and Cui et al. [CUI12] divide the system into regions, named clusters. In Al Faruque et al. [FAR08] approach, clusters are controlled by an agent (manager) responsible for the task mapping within a cluster. The overall system is controlled by many synchronized global agents, responsible for storing information related to all clusters, deciding in which cluster a given application will be mapped and re-organizing the clusters (re-clustering) at runtime. Cui et al. [CUI12] claim that the work proposed by Al Faruque leads to a high communication traffic to collect resource information to make decisions. To solve this issue, they propose a cluster-based scheme for task mapping, aiming to reduce the communication traffic between a global agent and the cluster agents, removing some of local clusters information from the global agent and changing the cluster reorganization scheme.

Anagnostopoulos et al. [ANA12] propose a divide and conquer method to perform distributed runtime mapping onto homogeneous and heterogeneous many-core platforms. When a new application is required to execute in the system, the proposed scheme divides the network in regions, using as criterion the application size. The region that best fit the application is chosen. A regional controller is configured to execute the mapping algorithm within the chosen region.

Kobe et al. [KOB11] propose task mapping using agents. However, agents do not have a predefined region to control as in the previous works. The agent selects PEs to map a given application. An agent is assigned at runtime to a random PE when a new application is required. Then, the agent searches for PEs to map the tasks, starting with the closest to farthest ones. The disadvantage of the presented method is the possibility of a communication bottleneck since the agent may become distant from the application it

controls.

Weichslgartner et al. [WEI11] propose a decentralized mapping approach aiming to reduce the NoC congestion. The proposed mapping approach considers only a local view of adjacent nodes on the NoC to execute the mapping, where each task contains a list of its succeeding tasks to be mapped. The method is limited to applications with a tree topology. The root (initial task) is mapped first, and then each task executes the mapping heuristic to map its successor tasks.

Table 2 summarizes the reviewed works according to the proposed taxonomy for dynamic mapping. Only a few works related to multi-task mapping were found in the literature, all proposed by Singh et al. [SIN09a][SIN09b][SIN10]. Multi-task techniques include *clustering*, which groups tasks to be executed in the same PE. A non-optimized *clustering* approach may lead to hotspots, reducing system lifetime and accelerating system wear out. Heterogeneous MPSoCs may have better performance for specific applications, and homogeneous MPSoC are general-purpose platforms. As industrial examples [INT12][TIL10], the present work focuses the research on homogeneous architectures. Another important feature is the distributed system management approach, as proposed in [FAR08][ANA12][CUI12][KOB11] and [WEI11]. Such approach is scalable and can reduce the mapping algorithm computational effort, increasing system performance.

The literature presents different runtime task mapping approaches to improve system reliability. All reviewed works use a centralized system management approach [CHA13][COS07][COS09][DAS13][DAS14][BOL13][HUA09][HAR12]. Among them, some works [DAS13][DAS14][BOL13] produce mapping decisions at design time, which are stored in a database and used at runtime. This approach may reduce system performance due its incapability of dealing with unpredictable system variations. Task mapping approaches proposed in [CHA13][HAR12], employ physical sensors to capture thermal or wear-state condition of cores at runtime. Included sensors provide accurate information to the mapping decision at the cost of additional system area and energy consumption. Huang et al. [HUA09] use an abstract MPSoC to validate the proposed approach, which can produce inaccurate performance results.

The literature presents distributed approaches to improve system reliability. However, such approaches use other techniques rather than task mapping [GE10][WU11][LIU15]. Ge et al. [GE10] propose a task migration approach for system thermal balancing. This approach uses thermal sensors, which aggregates hardware costs. Liu et al. [LIU15] also present a thermal management task migration approach, which does not consider performance costs. Wu et al. [WU11] present a dynamic frequency scaling for thermal management, which may impose additional hardware costs.

Table 2 – State-of-the-art in dynamic mapping heuristics.

Author / Year	Multi/ Mono-task	Architecture model	Control management	Optimization Goal
Smit et al. [SMI05]	Mono-task	Heterogeneous	Centralized	Energy Consumption and QoS application requirements
Ngouanga et al. [NGO06]	Mono-task	Homogeneous	Centralized	Communication volume, computation load
Coskun et al. [COS07][COS09]	Mono-task	Homogeneous	Centralized	System Reliability
Chou et al. [CHO07][CHO08] [CHO10]	Mono-task	Homogeneous	Centralized	Energy Consumption, Internal and external network contention
Hölzenspies et al. [HÖL07][HÖL08]	Mono-task	Heterogeneous	Centralized	Energy Consumption and QoS application requirements
Al Faruque et al. [FAR08]	Mono-task	Heterogeneous	Distributed	Execution time, napping time and nonitoring traffic
Wildermann et al. [WIL09]	Mono-task	Homogeneous	Centralized	Communication latency, energy consumption
Hosseinabady et al. [HOS09]	Mono-task	Homogeneous	Distributed	Reconfiguration overhead
Huang et al. [HUA09]	Mono-task	Homogeneous and Heterogeneous	Centralized	System Reliability
Schranzhofer et al. [SCH10]	Mono-task	Homogeneous	Centralized	Energy consumption
Lu et al. [LU10]	Mono-task	Homogeneous	Centralized	Communication latency and energy consumption
Carvalho et al. [CAR10]	Mono-task	Heterogeneous	Centralized	Network contention, communication volume
Singh et al. [SIN09a][SIN09b] [SIN10]	Multi-task	Heterogeneous	Centralized	Network contention, communication volume and energy consumption
Weichslgartner et al. [WEI11]	Mono-task	Homogeneous	Distributed	Communication latency and network contention
Kobe et al. [KOB11]	Mono-task	Homogeneous	Distributed	Execution time, Communication traffic
Cui et al. [CUI12]	Mono-task	Homogeneous	Distributed	Communication traffic energy consumption
Anagnostopoulos et al. [ANA12]	Mono-task	Homogeneous and Heterogeneous	Distributed	Communication volume
Khajekarimi et al. [KHA12]	Mono-task	Heterogeneous	Centralized	Network communication and congestion
Hartman et al. [HAR12]	Mono-task	Homogeneous and Heterogeneous	Centralized	System reliability
Chantem et al. [CHA13]	Mono-task	Homogeneous	Centralized	System reliability
Bolchini et al. [BOL13]	Mono-task	Homogeneous	Centralized	Energy consumption and system reliability
Das et al. [DAS13][DAS14]	Mono-task	Homogeneous	Centralized	Application deadlines and system reliability
Proposed work	**Multi-task**	**Homogeneous**	**Distributed**	**Communication volume reduction and workload distribution balancing**

This Thesis proposes a task mapping approach that differs from literature since it includes all the following characteristics:

- *Executed at runtime.* The proposed approach can better manage time-varying workloads and system changes.
- *Distributed mapping approach*. The proposed approach is implemented in an MPSoC managed in a distributed way. Such distributed system management improves system scalability by dividing the system into regions, each one with a manager responsible for actions inside it. Further, it reduces mapping decision computational effort, not compromising the system performance.
- *Induces to a better system reliability.* The proposed approach aims to improve communication volume reduction and workload balancing, which are directly related to a better system reliability [CHA13][WAN14].
- *Does not employ physical sensors in the mapping decision*, which increases area and energy costs.
- *Validated in large cycle-accurate MPSoCs* (10x10 MPSoC size).

3. PLATFORM MODELS

This Chapter details the first contribution of the Thesis: a multi-level modeling framework. Such framework aims to explore the vast number of alternatives in the design space of MPSoCs by combining high-level and high-accurate models and tools. The proposed framework is divided into three layers, as illustrated in Figure 1, in which the abstraction of the models is increased from the bottom to the upper layer.

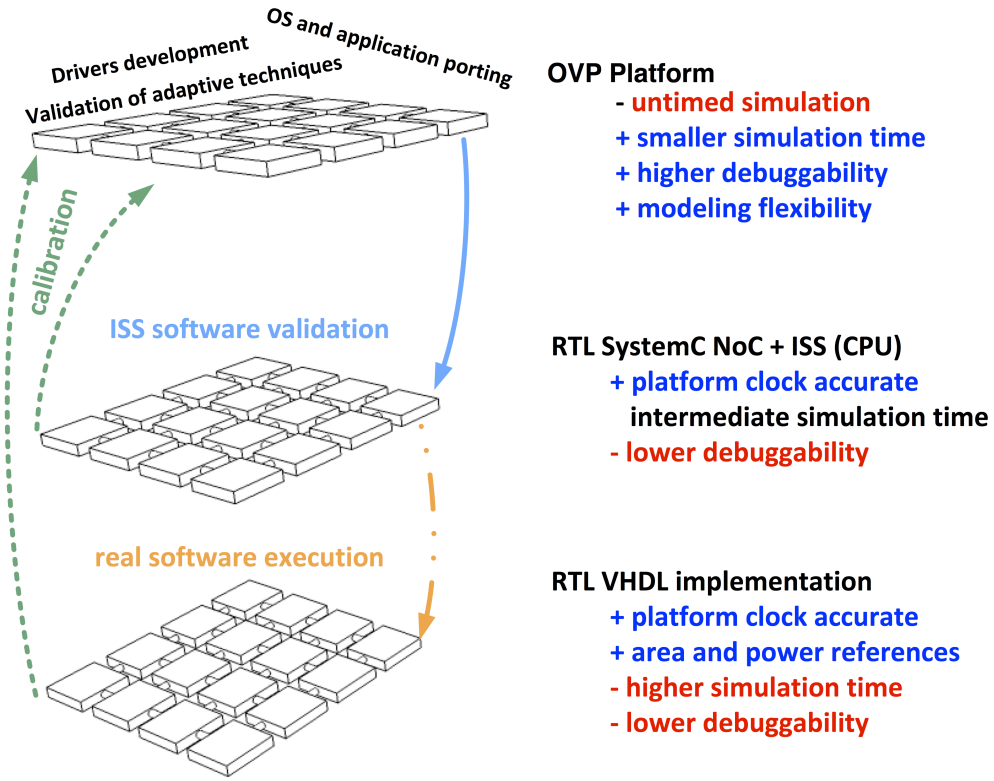
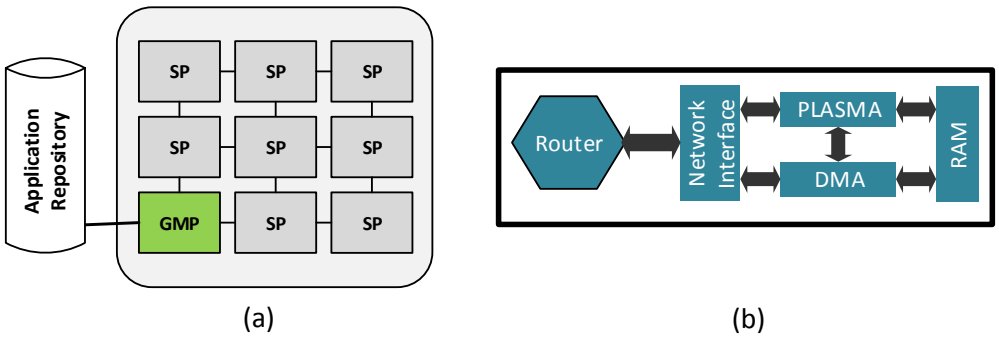


Figure 1 – Multi-Level modeling framework proposed by this Thesis.

The first layer concerns the reference NoC-based MPSoC VHDL RTL model presented in Section 3.1. Section 3.2 presents the second layer of the framework (Thesis contribution), where the reference platform is implemented in SystemC RTL. Section 3.3 presents an Open Virtual Platform (OVP) implementation of the reference platform (Thesis contribution). The interoperability between the three platform models is guaranteed through a well-defined hardware abstraction layer (HAL) and a unified software description (i.e. OSs, applications, communication model), also proposed in this Thesis. In this direction, target software stack can be modified and executed onto the OVP-based platform model until the point where its functionality is validated. The same code can then be executed in a still fast but clock-cycle accurate RTL SystemC-ISS model, which allows assessing lower performance figures (e.g. application execution time). Finally, RTL-VHDL model can receive the target software as input to profile the power figures, e.g., the average switching activity of adopted CPU architecture. Section 3.4 concludes this Chapter, summarizing the main results.

3.1 Reference Platform

The reference MPSoC model used in this Thesis is the HeMPS (Hermes Multiprocessor System-on-Chip) platform [WOS07][CAR09]. HeMPS is a general purpose homogeneous MPSoC in which processing elements (PEs) are interconnected through the Hermes NoC [MOR04]. An external memory, named application repository, contains the object code of the application tasks to execute in the system. The system uses distributed memory architecture, based on *scratchpad* memories rather than cache memory. HeMPS adopts scratchpad as local storage memories due to its power efficiency and management facilities when compared to cache memories. Further, scratchpad memory is more predictable in terms of access time, and it does not require any coherence protocol, as required by cache-based architectures [BAN02][VIL11]. In HeMPS, all communication occurs through message passing. Inter-task communication uses send and receive MPI-like primitives. Figure 2(a) illustrates a general view of a 3x3 instance of the HeMPS architecture.



(a) block diagram of 3x3 instance of the HeMPS platform, with SPs (slave PEs), and one GMP (Global Manager PE).

(b) Processing element of the HeMPS platform.

Figure 2 –HeMPS MPSoC block diagram.

The MPSoC architecture can be defined as a directed graph *GMPSoC = (PE, L)*. Each vertex $pe_i \in PE$ is a processing element, containing a MIPS-like processor (Plasma), a local memory (RAM), a DMA module, a network interface and a router, as shown in Figure 2(b). An *edge* $l_{ij} \in L$ is a NoC link interconnecting pe_i to pe_j.

Processing elements are divided into two different types: Slave Processing Element (SPs) and Manager Processing Elements (MPs). SPs are responsible for executing application tasks. Each SP runs a simple operating system, named microkernel, which supports communication between PEs, multitask execution and software interrupts (traps). The SPs local memory is organized into SP_PAGES pages. One page stores the operating system. Other pages, called *resources*, store the object-code of tasks that will be executed on this SP. Each SP can execute MAX_SP_TASKS tasks simultaneously, which corresponds to SP_PAGES -1. If a resource does not have a task mapped on it, it is considered free or available. The microkernel is a preemptive operating system where each task uses the CPU for a pre-defined period, named *timeslice*.

MPs are responsible for system management functions, including task mapping and system debug. MPs can be divided in different types, which are defined depending on the system management approach: centralized or distributed. Such system management approaches are better explained in Chapter 4. Both management approaches have a Global Manager Processing Element (GMP), which is a single PE responsible for receiving user requests demanding the execution of new applications on the system. GMP is also the only one that accesses the application repository.

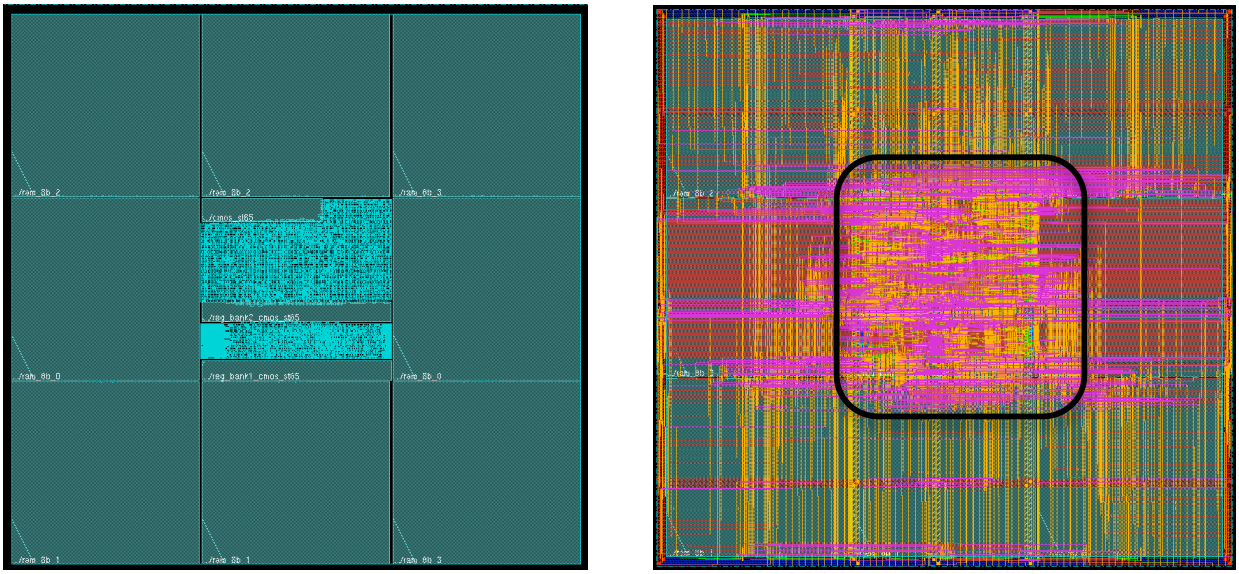
The Hermes NoC employs a 2D mesh topology. The communication mechanism is performed by wormhole packet switching, in which a packet is forwarded between routers divided by flits. Routers have input buffers, control logic shared by all router ports, an internal crossbar, and up to five bi-directional ports. These ports are east, west, north, south and local. The local port establishes communication between a router and a PE, and the remaining ports are used to connect a router to its neighbors. The arbitration algorithm used by the router is the round-robin. Each router has a unique network address. This address is expressed in XY coordinates, where X represents the horizontal position and Y the vertical position of the router in the network, being 00 the lower left corner. The XY routing algorithm is used, sending packets through the NoC first horizontally to reach the X destination router position, and then vertically to arrive at the destination router.

Originally, two HeMPS implementation models were available:

(i) a VHDL model: all components modeled in RTL VHDL.

(ii) a mixed model, called VHDL-ISS model: NoC, NI, and DMA are modeled in RTL VHDL; processors are modeled with an ISS (Instruction Set Simulator) with a SystemC wrapper, and memories are modeled in SystemC RTL. Such model has the same accuracy of the VHDL model since ISS processors and SystemC memories are clock-cycle accurate.

The VHDL is synthesizable, enabling to capture precise area, frequency and power performance figures. Such model was successfully implemented in a 3x3 instance FPGA prototype and a 65nm ASIC. The FPGA prototype contains the MPSoC and three additional modules: (*i*) MAC Ethernet communication interface with the host; (*ii*) control unit; (*iii*) DDR2 memory controller. The host sends the applications' codes to a DDR2 memory, which acts as the *application repository*. Next, the host may send commands to the MPSoC to start the execution of users' applications or to request debug information. The control unit is responsible for controlling the access to the external memory or the MPSoC. The 65nm ASIC implementation, using the memory generator of the design kit, required roughly one mm^2 for each PE, as illustrated in Figure 3. The MPSoC worked correctly after the back-end simulation.



(a) PE floorplanning, with the processor and the router in the center, and the memories in the periphery.

(b) PE layout. The side of the PE is 1 mm, and the side of the processor and the router is 0.3 mm. Technology: 65 nm.

Figure 3 – Processing element layout.

The main disadvantage of the VHDL model is its very high simulation time, with few debug facilities (e.g. waveforms). For this purpose, the VHDL-ISS model was implemented; reducing simulation time and obtaining some gains in debuggability (i.e. insertion of debug codes in the ISS processor model).

3.2 SystemC Platform

This Section presents a SystemC RTL model of the reference platform, which enables the exploration and the validation of large MPSoCs composed of dozens of PEs. The validation of such MPSoCs is unfeasible using the VHDL description of the reference platform due its high simulation cost. Underlying SystemC RTL model was co-developed with Eduardo Wachter, Ph.D. supervised by Prof. Fernando Moraes. The availability of such model enabled the exploration of distributed system management solutions, as presented in Section 4.4.

The proposed model preserved its reference platform accuracy, while reducing significantly the required simulation time. Another improvement of the proposed model is the additional validation and debugging capabilities inherited from SystemC facilities. The reference platform VHDL-ISS model was used as starting point for developing the SystemC RTL model. While ISS processor and the memory model descriptions are reused, the NoC, the NI, and the DMA were modeled as a SystemC RTL description, in order to preserve the VHDL accuracy. The modeled SystemC RTL NoC has been used in different research projects related to research on allocation (e.g. FP7 DreamCloud, http://www.dreamcloud-project.org), as well as in the exploration of compute accelerator architectures [GAR14].

Each VHDL module was rewritten in SystemC RTL without taking advantage of some SystemC language structures and primitives, in order to maintain a high accuracy. For example, SystemC supports channel primitives, such as *sc_fifo* that models the behavior of a FIFO buffer. Such primitive could be used to reduce the implementation and, possibly, the simulation time. However, this primitive is not cycle-accurate, which is the main requirement of the proposed model implementation.

	VHDL	SystemC
1	process(reset, clock_rx)	void fifo::in_proc()
2	begin	{
3	if reset='1' then	if(reset.read()==1){
4	last <=(others=>'0');	last.write(0);
5	elsif clock_rx'event and clock_rx='0' then	}else{
6	if avail_space='1' and rx='1' then	if((avail_space.read()==1)&&(rx.read()==1)){
7	buffer(CONV_INTEGER(last)) <= data_in;	buffer[last.read()] = data_in.read();
8	if(last = BUFFER_SIZE - 1) then	if(last.read()==(BUFFER_SIZE - 1))
9	last <= (others=>'0');	last.write(0);
10	else	else
11	last <= last + 1;	last.write((last.read() + 1));
12	end if;	}
13	end if;	}
14	end if;	}
15	end process;	}

Figure 4 – Description of the FIFO buffer control module in VHDL and SystemC RTL.

Figure 4 presents the code of a FIFO buffer control module, which verifies the buffer capacity to store incoming data. The left side of Figure 4 presents the VHDL code for this module, which is used in the reference model. The right side of the figure shows the code rewritten in SystemC RTL, preserving the same functionality. The only difference in both codes refers to the language syntax. Unlike VHDL, SystemC uses port methods when reading from (.read()) and writing to (.write()) a port, as exemplified in lines 3 and 4. In line 1, the SC_METHOD *fifo* is executed whenever an event occurs on its sensitivity list as the same way in the VHDL statement *process*. An SC_METHOD sensitivity list is defined in the constructor of an SC_MODULE, the equivalent of an ENTITY in VHDL. The “IF” conditional statement in SystemC replaces “THEN” and “END IF” by curly braces (“{“ and “}”, respectively), as seen in lines 3 and 12.

3.2.1 Comparison of the SystemC Model against the Reference Model

This Section compares the proposed SystemC model with the former reference platform models: the VHDL and VHDL-ISS models. The three discussed models’ characteristics are presented in Table 3, showing the description language for each system module.

Table 3 – Characteristics of the models.

Model→ Module↓	SystemC	VHDL-ISS	VHDL
Router	SystemC RTL	VHDL	VHDL
NI	SystemC RTL	VHDL	VHDL
DMA	SystemC RTL	VHDL	VHDL
RAM	SystemC RTL	SystemC RTL	VHDL
Processor	SystemC+ISS	SystemC+ISS	VHDL

Table 4 presents a qualitative comparison of the main models features regarding synthesizability; the precision of capturing area, frequency and power data; simulation time; throughput and latency values; and accuracy. As mentioned before, the VHDL model can be synthesizable, allowing capturing accurate performance figures. However, such model demands a very high simulation time, with few debug facilities. The VHDL-ISS reduces simulation time and increases debuggability when compared to the VHDL model. The proposed SystemC RTL model, avoids the VHDL-SystemC co-simulation of the VHDL-ISS model, boosting the simulation time and increasing debuggability.

Table 4 – HeMPS models comparison.

Model→ Parameter↓	SystemC	VHDL-ISS	VHDL
Synthesizable	No	No	Yes
Precise Area, Frequency, Power Evaluation	No	No	Yes
Simulation Time	LOW	MEDIUM	VERY HIGH
Accurate throughput and latency values	Yes	Yes	Yes
Accuracy	Clock cycle	Clock cycle	Clock cycle

The validation process of all models was performed using commercial simulators, such as Mentor Modelsim and Cadence Incisive, coupled to adequate verification models like the U-model and/or assertion-based verification. The debugging and validation of the SystemC model can be done in two ways: (*i*) generating a pre-compiled executable file of the MPSoC; or (*ii*) using a commercial RTL simulator, such as Modelsim. The use of commercial RTL simulators leads to higher simulation time than executable file approach. It is important to observe the last two rows of Table 4. The obtained values for throughput and latency are the same for the three models since all have clock-cycle accuracy.

The simulation cost of each HeMPS MPSoC model is evaluated by using different scenarios. Such scenarios consider 4x4, 5x5, 7x7 and 10x10 MPSoC configuration; different MPSoC occupation, static mapping, and SPs configured to execute up to 2 simultaneous tasks. Experiments use different instances of a synthetic application with six tasks, illustrated in Figure 5. The number of instances of such application varies from one; and a given number of instances that corresponds to approximately 25%, 50%, 75%, and 100% of the MPSoC occupation. Table 5 shows the simulation time for each scenario.

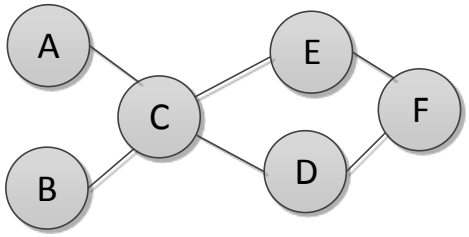


Figure 5 – Synthetic application with 6 tasks used in the test scenario.

The last two columns of Table 5 evaluate the speedup obtained using VHDL-ISS and SystemC models, compared to the VHDL model. The intermediate model, VHDL-ISS, provides a simulation time speedups of at least one order of magnitude compared with the VHDL model. In turn, improvements of two orders of magnitude are achieved with SystemC model. The last scenario, a 10x10 MPSoC instance, with all SPs running two tasks, took 8.5 minutes to simulate with SystemC. The VHDL model would require at least 25 hours (assuming a speedup value of 180) for this test case.

Table 5 - Simulation time (in seconds) and speedup. Simulations run on a 6-core, 64 bits Xeon architecture with 12 Gbytes of RAM, running Linux OS. [PET12]

MPSoC size	MPSoC occupation	VHDL	VHDL-ISS	SystemC	Speedup VHDL-ISS/ VHDL	Speedup SystemC/ VHDL
4x4	1 instace	3852.34	217.35	34.99	**17,7**	**110,1**
	25%	4219.99	220.17	35.26	**19,2**	**119,7**
	50%	4701.47	231.67	37.97	**20,3**	**123,8**
	75%	5162.13	245.78	40.56	**21,0**	**127,3**
	100%	7288.23	291.73	46.86	**25,0**	**155,5**
5x5	1 instance	7742.04	336.25	53.22	**23,0**	**145,5**
	25%	8134.28	359.32	60.83	**22,6**	**133,7**
	50%	9087.88	423.85	65.76	**21,4**	**138,2**
	75%	12205.26	429.10	82.84	**28,4**	**147,3**
	100%	12460.31	466.63	85.45	**26,7**	**145,8**
7x7	1 instance	19765.50	682.96	114.18	**28,9**	**173,1**
	25%	21797.14	724.61	129.06	**30,1**	**168,9**
	50%	32153.21	1049.32	179.42	**30,6**	**179,2**
	75%	41211.06	1272.44	226.46	**32,4**	**182,0**
	100%	50557.62	1538.20	274.07	**32,9**	**184,5**
10x10	1 instance	50862.91	2344.64	289.52	**21,7**	**175,7**
	25%	NA	4319.71	508.97	**NA**	**NA**
	50%	NA	5686.91	668.76	**NA**	**NA**
	75%	NA	7435.22	945.25	**NA**	**NA**
	100%	NA	9897.06	1122.85	**NA**	**NA**

Figure 6 plots the simulation time for the highlighted rows of Table 5 (MPSoC instances with 50% load). Note that the VHDL simulation time is presented in the secondary Y-axis, due the large amount of time required to simulate the underlying scenarios. A significant impact on simulation time is observed when a 10x10 MPSoC configuration is used. Resulting cost is mainly due to the co-simulation process. The trend of the simulation time growth with regard to the MPSoC size for the SystemC model is $O(SP^2)$ (time = $0.0632.SP^2 + 0.2932.SP + 20.533$, $R^2 = 0.99998$). For example, for 400

SPs it is expected a simulation time of 2.83 hours with the SystemC model.

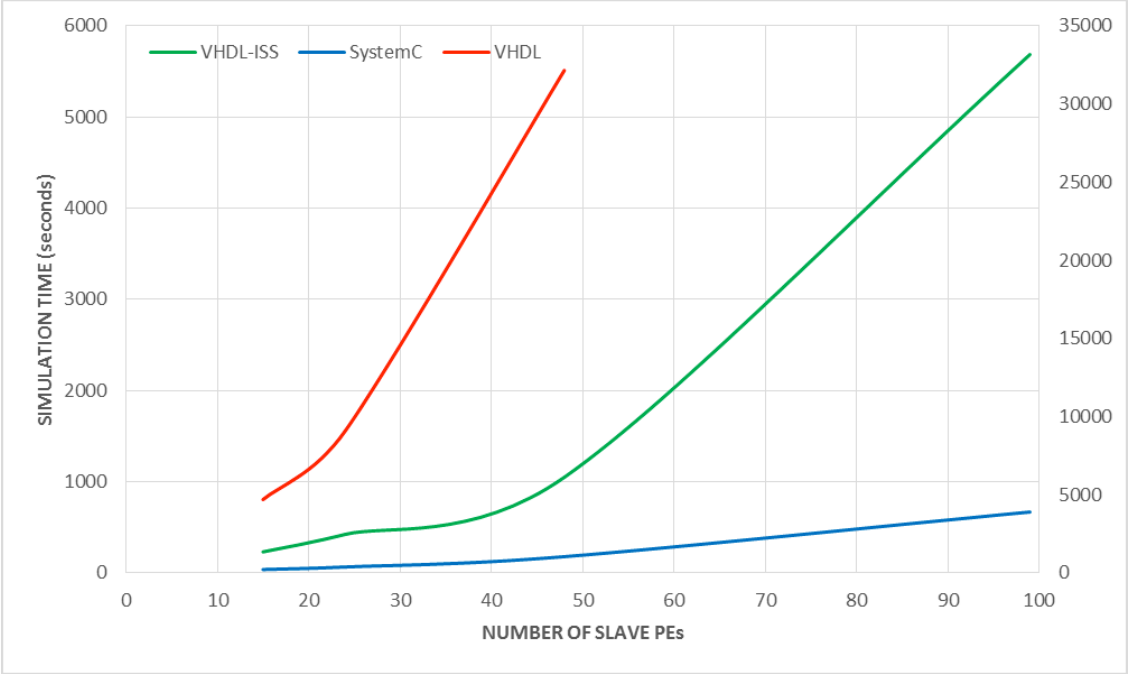


Figure 6 – Simulation time for MPSoC instances with 50% load (VHDL simulation time is presented in the secondary Y-axis).

To demonstrate the accuracy of the proposed SystemC model in terms of executed instructions and execution time, the VHDL model is used as reference. Table 6 presents the experimental setup. The scenarios use 4x4 platform instance, in which two applications are evaluated: a partial MPEG decoder, with 5 tasks; and synthetic VOPD (Video Object Plane Decoder) application, with 12 tasks. The first scenario (SC1) executes 2 MPEG and 1 VOPD instances, totalizing 22 tasks. The second (SC2) and third (SC3) scenarios contain 44 and 88 tasks, respectively.

Table 6 - Setup of applications, in a 4x4 MPSoC, with one manager PE.

Applications	**SC1**	**SC2**	**SC3**
MPEG (5 tasks)	2	4	8
VOPD (12 tasks)	1	2	4
Total tasks	**22**	**44**	**88**

Table 7 evaluates the simulation time (8-core Xeon processor, 32 GB RAM) for the three scenarios, whereas a speedup of two orders of magnitude is observed using the SystemC model.

Table 7 – Simulation time (in seconds) for RTL-VHDL and RTL-SystemC models.

Scenarios	**VHDL**	**SystemC**	**VHDL/SystemC**
SC1	2,425	19	**127**
SC2	4,407	37	**119**
SC3	7,932	51	**155**

Table 8 evaluates the execution time required to execute all applications of each scenario, and Table 9 presents the total number of instructions executed by all processors in the platform. *Such results demonstrate that the SystemC and VHDL models in practice have a similar behavior.* The differences observed in the Tables are due to simplifications made in the ISS, which do not reflect the real behavior of some instructions (e.g. multiplication and division instructions).

Table 8 – Execution time (in clock-cycles) for low-level models.

Scenarios	VHDL	SystemC	VHDL/SystemC
SC1	265,015	265,228	**0.998**
SC2	526,660	528,524	**0.996**
SC3	1,050247	1,055,543	**0.995**

Table 9 – Number of executed instructions for low-level models.

Scenarios	VHDL	SystemC	VHDL/SystemC
SC1	422,757	423,120	**0.999**
SC2	834,335	836,335	**0.997**
SC3	1,662,311	1,664,941	**0.998**

3.3 OVP Platform

Although the proposed SystemC platform provides a considerable simulation speedup compared to the VHDL model, the achieved simulation performance may remain a bottleneck for the exploration of large systems running several applications simultaneously. In attempt to overcome such bottleneck, this Section describes another contribution of this Thesis, the development of two new platforms: (*i*) mixing SystemC OVP – section 3.3.1; (*ii*) OVP only - section 3.3.2. The former keeps the clock-cycle accuracy of the NoC (RTL-SystemC), with the OVP flexibility to use different processors and software debugging. The OVP-only platform sacrifices accuracy but enables faster software development.

With 100-core chips already available [DE13], software development becomes one of the major challenges in MPSoC design. For instance, IBS [IBS13] projects that software development consumes at least 50% of the system's design cost, and that percentage is rising, as illustrated in Figure 7. Software development comprises, among others: (*i*) inter-processor communication protocol stacks definition; (*ii*) OS porting and analysis; (*iii*) exploration of better programming model facilities to address parallel programming [MAR12]; (*iv*) drivers development [GRA12]; (*v*) application software portability for heterogeneous multiprocessing hardware.

Virtual platforms have been employed to achieve concomitant hardware and software development, while providing more efficient design exploration support (e.g. debuggability) [CEN09]. A virtual platform is a full-system simulator that emulates hardware components

(e.g. CPUs, memories), allowing evaluating a given software stack, on the same machine, as it is running on a real physical hardware. Examples of such simulators are Simics [SIM13], PTLsim [YOU07], SimpleScalar [AUS02], gem5 [BIN11] and OVPSim [OVP13]. Except PTLSim that only supports x86, all mentioned simulators offer at least five processor architectures. For instance, Simics from WindRiver supports Alpha, ARM, MIPS, PowerPC, SPARC and x86 models. While Simics and OVPSim are respectively functionally-accurate and instruction-accurate, the remaining simulators can be considered as quasi-cycle-accurate.

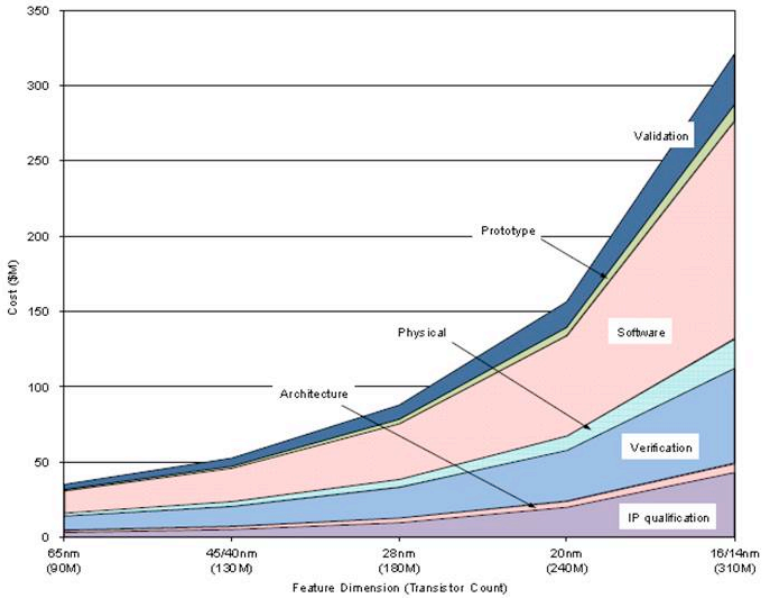


Figure 7 - Software and architectural design costs for embedded systems at advanced process technologies. Figure extracted from IBS 2013 [IBS13].

Cycle-accurate simulators target microarchitecture exploration since they provide specific modeling details, such as the pipeline implementation and cache coherence protocols [BIN11]. However, these simulators are not scalable to a large number of processors, specifically when it comes to simulation speed and debugging usability. This scenario points to the use of OVPSim since it covers our main requirements: (*i*) large number of processor architectures supported; (*ii*) scalable and acceptable simulation time (hundred of MIPS); (*iii*) open source license; (*iv*) component-oriented infrastructure; (*v*) active development support. Nevertheless, OVPSim does not model cycle-accurate processors but rather instruction accurate processors, which provides inaccurate application execution time. Another limitation inherent to OVPSim is the fact that only bus-based architectures are available in the original distribution.

OVPSim [OVP13] is a virtual platform and modeling framework proposed by Imperas, aiming to accelerate the development of embedded software, specifically for SoCs and MPSoCS. It is composed of three main components:

(i) APIs that enable to model in C language hardware components;

(ii) library of free open-source CPUs and peripheral models;

(iii) OVPsim simulator. OVPsim is a dynamically linked library, which supports the simulation of bus-based multiprocessor platforms. OVPSim relies on dynamic binary translation that increases simulation speed [OVP13].

OVPSim provides pre-defined models including: processor models (ARM, MIPS, PowerPC, etc.), system components (RAM, ROM, caches memories, etc.) and peripheral models (DMA, UART, FIFO, etc.). These models can be combined in a single platform by using the ICM (*Innovative CPU Manager*) API, which also provides fully control and observability of all components. Processor behavior models can be described using the VMI (*Virtual Machine Interface*) API, which decodes the target instruction to be simulated translating it to x86 instructions that are then executed on the host machine. Peripheral and components behavior models can be described using the PPM (*Peripherals Models*) and BHM (*Behavioral Models*) APIs. Such APIs use an event-based scheduling mechanism to enable modeling of time, events, and concurrency [OVP13].

When executing multiprocessor platforms, each processor is executed for a certain number of instructions (*quantum*) in OVPSim. The simulator defines 100,000 instructions by processor as the default quantum value. The number of million instructions executed per second (MIPS) can be specified in a processor model, defining its execution speed. OVP processor models have a default speed of 100 MIPS.

3.3.1 SystemC/OVP Platform

As mentioned before, one of the main limitations of OVP is the fact it provides only bus-based platforms. To overcome this restriction, this Section presents the integration of the SystemC NoC model to the OVP components' library. One could argue that a simple crossbar to interconnect CPUs at higher abstraction levels would be sufficient to develop and to validate applications and operating systems since a crossbar supports parallel transactions between CPUs. This work advocates that integrating instruction-accurate CPU models with a cycle-accurate NoC enables:

I. capturing the communication volume at each link – allowing to compute the power spent in the communication infrastructure as a function of the data volume and number of hops [HU10];

II. mapping quality evaluation – by using the hop number between tasks and the communication volume it is possible to evaluate different mapping heuristics;

III. drivers development at higher abstraction levels – the NoC model uses wires instead of TLM transactors, enabling to develop the required drivers.

The complex process of integrating distinct CPUs in NoC-based MPSoC platforms limits the implementation and the exploration of multiprocessor systems. For instance, the implementation of a network interface is a time-consuming task, which requires designer knowledge in terms of HW/SW implementation and protocols definition. In this sense, one

important feature of the proposed modeling is the easiness of integrating different CPU models, allowing the development of heterogeneous MPSoCs (this research topic is out of the scope of the present Thesis).

Figure 8 details the architecture of the SystemC/OVP platform, describing the interconnection of the OVP CPUs with the SystemC NoC. The numbers in Figure 8 correspond to a packet reception and its processing by the OVP CPU model.

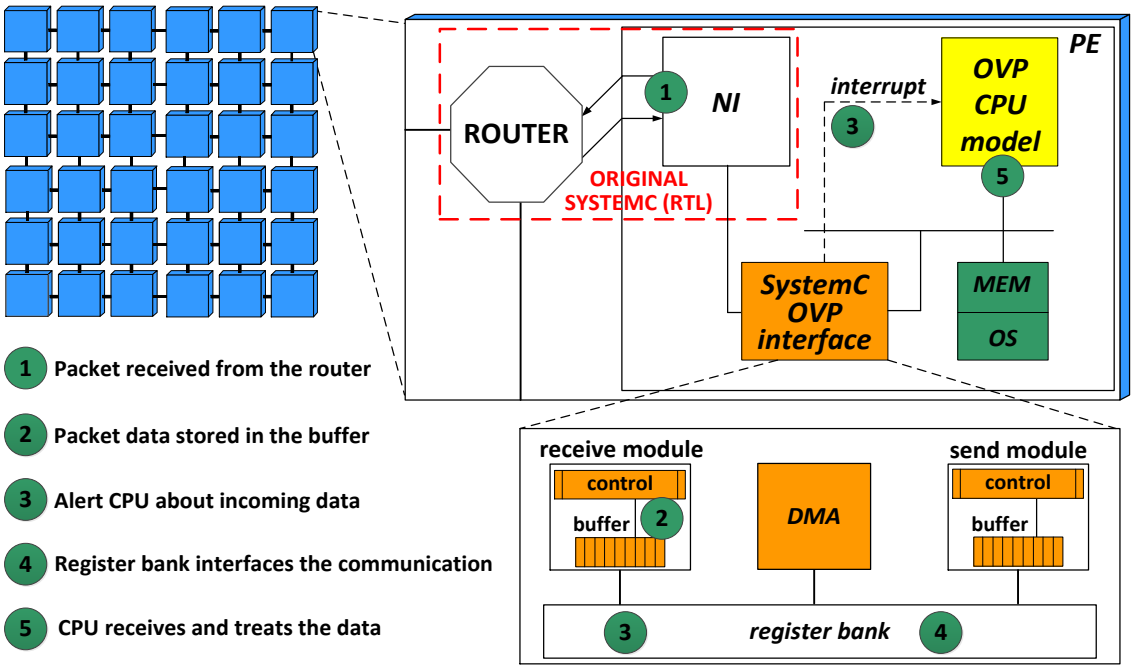


Figure 8 – Integration of SystemC NoC model with OVP CPU model.

Initially, the NI receives a packet from the router (step 1). An event is triggered notifying the receive module (block inside of the SystemC-OVP interface) related to the incoming packet. The receive module then reads the incoming packet, stores it into a buffer used to synchronize the communication between the untimed CPU and the clock-cycle accurate NI (step 2). After storing a complete packet, the module informs the CPU that there is data stored in the buffer.

As shown in step 3, there are two ways to inform a CPU about incoming data:

- for an SP, an interrupt is raised, and an ISR (Interrupt Service Routine) is called to read data.
- for MPs, a memory mapped register is used to alert the stored data. This CPU polls this register periodically. Once the CPU is ready, the data is read.

In both cases, the packet data is read by the DMA module using memory mapped registers (register bank in step 4). The register bank is implemented using an external memory mapped in the processor address space, where each register has a pre-defined address. The CPU is connected to a bus to which all address-mapped components are connected. This bus connects the local memory and the register bank.

Figure 9 shows the initialization of the register bank, with an address range from 0x00000000 to 0x0FFFFFFF (line 1). Once initialized, the extMem is connected to the processor bus in an address area (line 2), which is defined according to the adopted CPU model (in this case 0xF0000000 to 0xFFFFFFFF). This memory is also “connected” to the callback functions *regbankR* and *regbankW* (line 2).

```
1. extMem->init(0x00000000, 0x0fffffff);
2. proc->extMem(0xf0000000, 0xffffffff, regbankR, regbankW, extMem);
```

Figure 9 – Example of register bank external memory initialization, and the connection to the processor bus.

Callback functions are executed on every read (*regbankR*) or write (*regbankW*) access to the defined address area. In this case, when the processor accesses this area, the OVP simulator calls functions responsible to interconnect the SystemC to the OVP. Figure 10 shows an example of a callback function related to a read memory access. The callback function name (*regbankR*) is defined as a parameter of the ICM_MEM_READ_FN macro (Figure 10). The function parameter provides the memory address (*address*) accessed by the CPU, as well as the value (*value*) to be read from this address. Once a read memory access is triggered, the provided address is compared with the previously defined register address (line 3). The value is read by the processor (line 4) when the address is equal to REG_ADDRESS1 (line 3). Note that the read value comes from a SystemC signal (system_c_signal1), creating the communication between OVP and SystemC.

```
1. ICM_MEM_READ_FN (regbankR)
2. {
3.    if(address = REG_ADDRESS1)
4.           value = system_c_signal1.read();
5. }
```

Figure 10 – Example of a pseudo-code for a read callback function.

Figure 11 gives an example of a write callback function. This function is specified using the ICM_MEM_WRITE_FN macro, and the callback function *regbankW* as a parameter. This parameter provides the memory address (address) accessed by a CPU, as well as the value (value) to be written in this address. When a write memory access is triggered, the provided address is compared to the previously defined register address (line 3). If the address matches REG_ADDRESS1 (line 3), the value is written in a SystemC signal, sending, for example, a read data request from the processor to the NI.

```
1.ICM_MEM_WRITE_FN (regbankR)
2. {
3.    if(address = REG_ADDRESS1)
4.           system_c_signal1.write(value);
5. }
```

Figure 11 – Example of a pseudo-code for a write callback function.

Finally, using the memory-mapped registers as interface, the processor receives and processes data (step 5 in Figure 8). When a processor needs to send data through the NI, these five steps are repeated but using the send module (Figure 8, inside SystemC-OVP Interface). First, the CPU uses the memory-mapped registers as an interface to the communication protocol with the NI. For each register access, a memory callback is triggered, generating a SystemC event. Then, the send module receives the packet and stores it in the buffer. When the packet is completely stored, it is sent through the NI.

3.3.1.1 SystemC/OVP Platform Evaluation

This Section evaluates quantitatively the simulation time and the feasibility and the advantages of using the SystemC/OVP platform to boost software development when comparing to the SystemC model. Task mapping heuristics presented in Chapter 5 are taken as software development case study.

In order to compare the simulation time of both platform models, different scenarios are used varying: (*i*) platform size: 6x6 (36 PEs), 8x8 (64 PEs), 10x10 (100 PEs), 12x12 (144PEs), 14x14 (196 PEs) and 16x16 (256 PEs); (*ii*) resource occupation, i.e., number of PEs executing tasks: 30% and 50%. All scenarios execute one or more instances of a Digital Time Warping (DTW) application, with ten tasks, which recognizes patterns measuring similarities between two sequences that may vary in time or speed.

Figure 12 presents the simulation time for all scenarios. It is possible to observe a distinct behavior of the SystemC model compared to the SystemC/OVP model. Using the purely SystemC model the simulation time grows linearly with a load equal to 30% (R^2=0.998) and quadratic (R^2=0.999) for a load equal to 50% (confirming the trend presented in Figure 6). Besides the reduction of the simulation time achieved with the SystemC/OVP model, both SystemC/OVP graphs present a quadratic growth (R^2=0.998) in both graphs.

This behavior is due to the synchronization between the instruction-accurate OVP model with the clock-cycle accurate SystemC model. As the number of PEs increases, as well as the load applied to the system, the number of synchronization events also increases. To put in perspective the above results, Table 10 presents the speedup obtained using the SystemC/OVP model. Speedup suggests that systems containing up to 144 PEs (12x12) benefit from a 2x speedup. This is an unforeseen result since larger speedups were expected. The quest to reduce the simulation time is presented in Section 3.3.2, with a simplified NoC model in OVP.

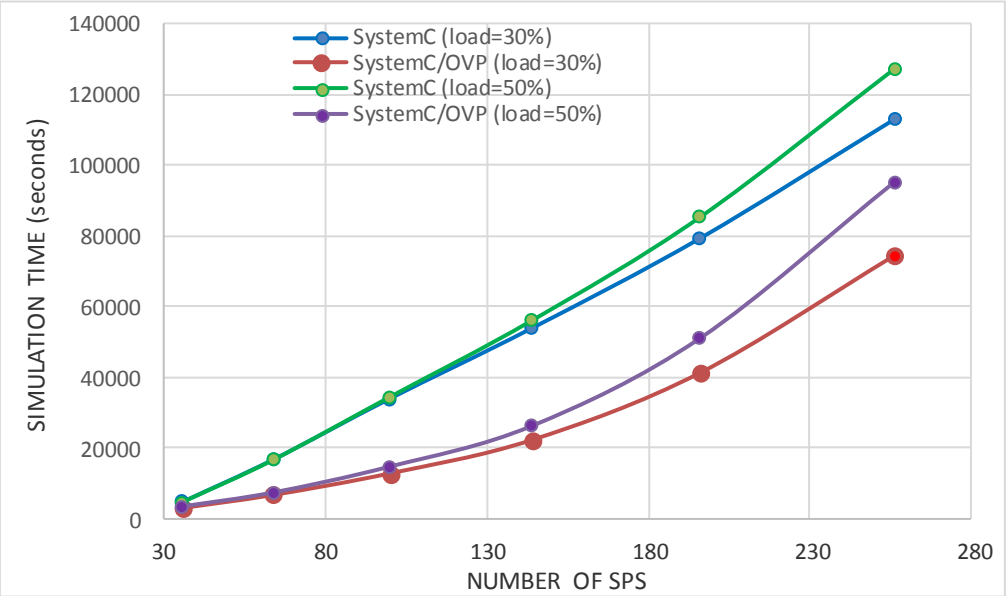


Figure 12 – Simulation time, varying the platform modeling, number of PEs, and MPSoC load. Simulations setup – processor: Core 2 Duo E4400 2x2GHz; memory: 3GB; gcc version: 4.7.2; gcc flags: -mfpmath=sse -Ofast -flto -march=native -funroll-loops.

Table 10 - Normalized simulation time speedup for the SystemC/OVP model compared to the SystemC model.

Load/PEs	36	**64**	**100**	**144**	196	256
30%	1.59	**2.50**	**2.66**	**2.42**	1.92	1.52
50%	1.36	**2.25**	**2.34**	**2.14**	1.67	1.34

Software development is evaluated by exploring dynamic mapping heuristics. The mapping heuristics evaluated are PREMAP-DN and LEC-DN, described in Chapter 5; and NN [CAR10]. Six applications are used: MWD (12 tasks), AAV (8 tasks), MPEG4 (12 tasks), Synth (9 tasks), VOPD (12 tasks), SegImg (6 tasks). Applications are modeled synthetically, i.e., from the application graph it is obtained the communication volume between each communicating pair, and such behavior is modeled in C language, using send and receive MPI-like primitives. This evaluation adopts a 6x6 MPSoC instance (1 manager PE and 35 slave PEs). Each slave PE may execute up to 2 tasks simultaneously. Therefore, the MPSoC can execute simultaneously up to 70 tasks. Three different scenarios are evaluated: (*i*) MWD, MPEG4 and AAV - 32 tasks; (*ii*) MWD, VOPD and Synth - 33 tasks; (*iii*) MPEG4, VOPD, MWD and SegImg - 42 tasks.

Table 11 presents the communication volume transmitted through the NoC for each mapping heuristic, in thousands of flits. The first two mapping heuristics, PREMAP-DN, LEC-DN, have in their cost function the communication volume. Comparing both models, the difference is smaller than 2%. Even though, NN (nearest neighbor) heuristic reports a difference of around 25%, the ranking among the scenarios remains the same. The

observed difference is due to a different injection rate in the network since the abstract processor models cannot generate data with cycle-accuracy.

Table 11 - Communication volume transmitted through the NoC for each mapping heuristic, in Kflits.

	PREMAP-DN		**LEC-DN**		**NN**	
	SC	SC/OVP	SC	SC/OVP	SC	SC/OVP
Scenario 1	214,7	203,0	262,4	255,5	236,6	310,9
Scenario 2	54,2	53,6	72,6	72,1	88,0	97,6
Scenario 3	140,3	145,5	97,5	105,0	108,7	143,2

Such results demonstrate that SystemC/OVP may be used at higher abstraction levels to develop MPSoC applications, but the simulation time is still an issue to be minimized.

3.3.2 OVP Platform

Experimental results regarding the SystemC/OVP platform showed that the simulation time presents a quadratic grow, suggesting values close to the SystemC platform for large systems. The main reason to adopt a SystemC NoC model integrated to OVP was to keep the accuracy of latency and throughput values. The analysis of the mapping results presented in Table 11 showed that, besides the obtained accurate communication volume, differences in the position of the mapped tasks in the SystemC and SystemC/OVP models were observed.

The explanation of such behavior comes from how processors are modeled. Instead clock-cycle accurate, OVP processors are instruction-accurate, thus packets are injected at different moments compared to clock-cycle models. This difference leads to the following consequences: (*i*) as it is not possible to determine *exactly* when packets are injected, it is not possible to evaluate precisely congestion; (*ii*) packets may arrive at the manager PE at different moments, changing the mapping order. Therefore, an OVP model is proposed, by replacing the NoC and NIs by OVP peripherals. This model is approximate timed since the execution time may be inferred from the number of executed instructions and latency/throughput values from the communication volume and the number of hops.

Routers detailed in Figure 13 compose the NoC OVP model. A router has five bi-directional ports (input and output data ports), input buffers, and arbiter modules. The local port establishes communication between a router and a PE, and the remaining ports are used to connect a router to its neighbors. All router connections are implemented by using OVP Net ports (represented by red arrows in the Figure), which model single or multi-bit wires. When data is written to a Net port, a *callback* function is triggered.

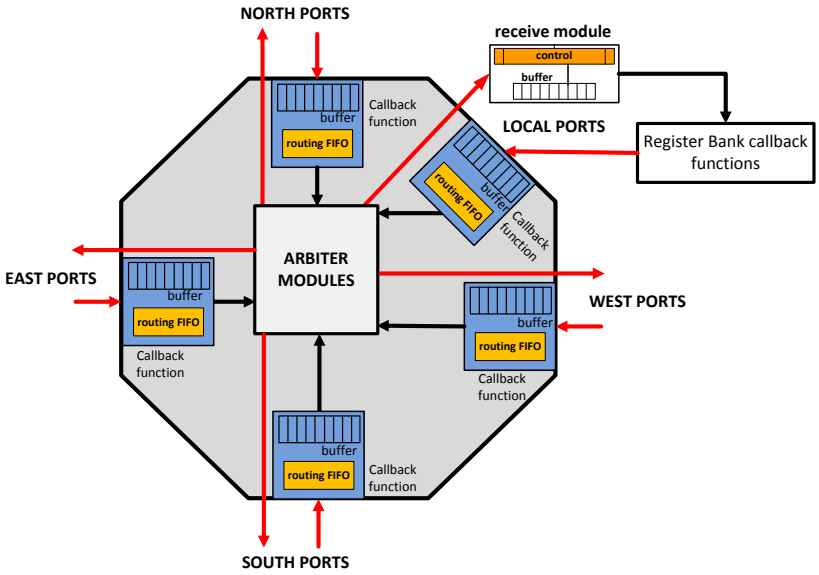


Figure 13 – Processor and NoC router connection in the HeMPS OVP platform.

Packets are sent through the network divided by *flits*. Since routers are interconnected through *OVP Nets*, when a *flit* arrives in an input port a *callback* function (illustrated as a blue box in Figure 13) is triggered. This function first stores the *flit* in a buffer. If the incoming *flit* is the first one of a packet (containing the packet destination), the routing algorithm (XY) is executed. The routing algorithm selects an output port to send the incoming *flit*, storing the selected port in the routing FIFO. All *flits* are sent through the selected output port if the arbiter grants access to the packet.

Note that this model assumes infinite input buffers (dynamically allocated/deallocated), but preserving the wormhole packet switching mode. In addition, it is important to mention that the routing algorithm is executed in a distributed fashion, at the input ports.

The arbitration is also distributed. An individual arbiter is used at each output port. An output port constantly seeks for data to transmit. The arbiter adopts a round-robin algorithm to select an input port to be connected to the output port. An output port arbiter selects an input port buffer only if the routing FIFO has pending packets to this output port.

For example, consider that the input port NORTH has a packet to be transmitted to the SOUTH output port, i.e., the NORTH routing FIFO has a pending routed packet to the SOUTH port. When the SOUTH arbiter schedules the NORTH input port, the NORTH buffer content starts to be transmitted through the SOUTH port. Both ports (input and output) are aware of the packet size (second packet flit). The output port keeps the connection until the transmission of the last packet flit, and the input port releases the buffer area when the packet was completed transmitted.

As in the SystemC/OVP model, PE communicates through the NoC using memory-mapped registers. A register bank is implemented in a processor memory address space,

where each register has a pre-defined address. *Callback* functions are triggered when a processor accesses the register bank area, making an interface between this processor and a router local port (as seen in “Register Bank callback functions” in Figure 13).

When a PE sends a packet through the network, it writes this packet flit by flit in a pre-defined register memory address, triggering a callback function as described in Figure 14. This function is specified using the *ICM_MEM_WRITE_FN* macro and the callback function *regbankW* as a parameter. This parameter provides the accessed memory address, as well as the value to be written to this address. In the function, the provided address is compared to a pre-defined register address (line 3). If the address matches *RegSendFlit* (line 3), the value (a packet flit) is sent (line 4). Data is written to an OVP Net port *data_out* by using the *icmWriteNet* primitive.

```
1.ICM_MEM_WRITE_FN (regbankW)
2. {
3.    if(address = RegSendFlit)
4.           icmWriteNet(data_out, value);
5. }
```

Figure 14 – Pseudo-code of a memory-mapped register callback function that sends a packet flit by flit through the network.

Then, packet data arrives at a router, being routed until the destination router. When data arrives at a destination router, it is sent through a net structure to an intermediary module called *receive module* (as shown in Figure 13), which makes an interface between a processor and the NoC. When this happens, a callback function is triggered inside this module, which receives the data and stores it in a buffer. The module alerts a processor each time an entire packet is stored. The processor receives this alert and uses a set of memory-mapped registers to read the packet from the buffer module.

Figure 15 shows the pseudo-code of memory-mapped register callback function that reads a flit of an incoming packet from the *receive module* buffer. The callback function name (regbankR) is defined as a parameter of the ICM_MEM_READ_FN macro. The function parameter provides the accessed memory address, as well as the value to be read from this address. Once a read memory access is triggered, the provided address is compared with a previously defined register address (line 3). If the address matches *RegReadFlit* (line 3), a flit is read from the *receive module* buffer (line 4).

```
1. ICM_MEM_READ_FN (regbankR)
2. {
3.    if(address = RegReadFlit)
4.           value = BUFFER[index];
5. }
```

Figure 15 – Pseudo-code of memory-mapped register callback function that reads a flit of an incoming packet.

The proposed OVP model also provides performance metrics, offering to designers some low-level performance results. Such metrics are computed at the end of the simulation, and include:

(i) *Communication volume*.

(ii) *Energy spent in the NoC*: characterized using the method proposed by [HU10], calibrated using the ST/IBM CMOS 65 nm technology at 1.0 V, adopting clock-gating, and a 100 MHz clock frequency.

(iii) *Execution time and consumed energy for each processor*: characterized using the model proposed by Rosa et al. [ROS13][ROS14]. This model counts and captures the executed instructions for a given processor, grouping them according to their behavior. Then, for each group of instructions, it is estimated the number of clock cycles and the energy consumed to execute them. Energy model is characterized by processor logic synthesis performed with Cadence RTL Compiler tool targeting a 65nm low power library from ST Microelectronics. This Energy Model is better explained in Section 4.1.

3.3.2.1 OVP Platform Evaluation

This Section evaluates the behavior of the OVP model for large MPSoCs, by using the proposed SystemC model as the reference model. This Section will not apply comparative evaluations efforts with VHDL model, considering that the SystemC model presented accurate results compared to VHDL model.

Table 12 presents the configuration of five scenarios used to evaluate the SystemC and OVP models in terms of simulation time. Scenarios use different MPSoC sizes: 4x4, 6x6, 8x8, 10x10, and 12x12. As illustrated in the left column of the Table, different real applications are used: a partial MPEG decoder (with 5 tasks), a Dijkstra (with 6 tasks), a Digital Time Warping (DTW, with 10 tasks), and a Fixed-Based application (with 15 tasks). The second to fifth lines of the Table contain the number of executed applications. For example, the 10x10 MPSoC scenario executes two instances of the MPEG and DTW applications, and 3 instances of the Dijkstra and Fixed-Based applications. The last line of the table contains the total number of executed tasks in each scenario.

The number of instructions executed by the GMP corresponds to the execution time in the OVP model. SPs enter in an idle state when they are not executing tasks, i.e., they stop executing instructions. On the other side, the GMP does not enter in idle state, executing instructions during all the simulation. The GMP starts the applications, mapping them into the system. When an application finishes its execution, the GMP is notified about its conclusion. At the end of the simulation, the number of instructions executed by the GMP is reported. Each instruction has a known CPI (Cycles Per Instruction), which enables to determine the number of clock cycles required to execute all applications.

Table 12 - Setup of applications distribution.

MPSoC size	4x4	6x6	8x8	10x10	12x12
MPEG (5 tasks)	1	1	1	2	4
Dijkstra (6 tasks)	-	1	3	3	5
DTW (10 tasks)	1	1	2	2	4
F.Base (5 tasks)	-	1	1	3	3
Total tasks	**15**	**36**	**58**	**93**	**135**

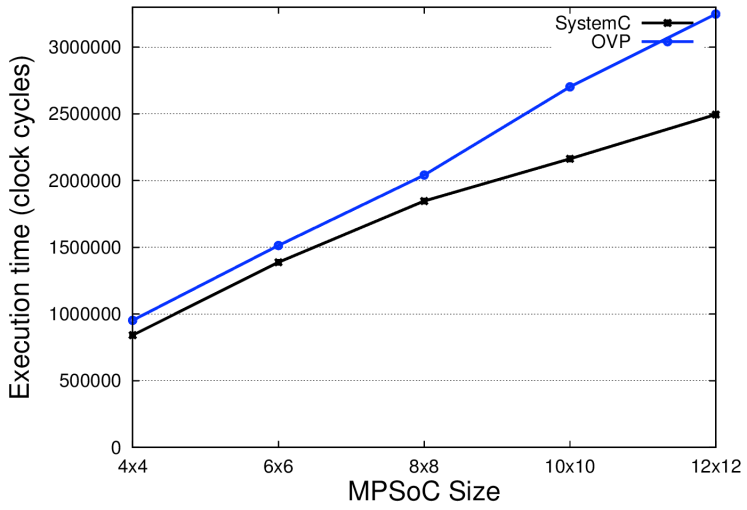
Figure 16(a) presents the execution time, in clock cycles, for all scenarios. For MPSoCs up to 64 PEs, the difference between RTL-SystemC and OVP is approximately 10%. For larger systems (up to 144 PEs), this difference reaches 25%. At the end of simulation, both SystemC and OVP models report the number of executed instructions per PE. The number of executed instructions enables to estimate the energy consumption, once each instruction also has a known energy cost. Figure 16(b) presents the total number of simulated instructions for all scenarios, with a difference of 17% for larger MPSoC scenarios. Even if the same workload is applied for both models, the number of instructions varies due to differences in number of tasks allocated per processor and NoC traffic.

Execution time and total number of simulated instructions are important for software designers. Even if the OVP model is not clock-cycle accurate, this model reports execution time and number of instructions (that enables to compute the energy consumption) with an error inferior to 10% compared to the gate-level implementation [ROS14].

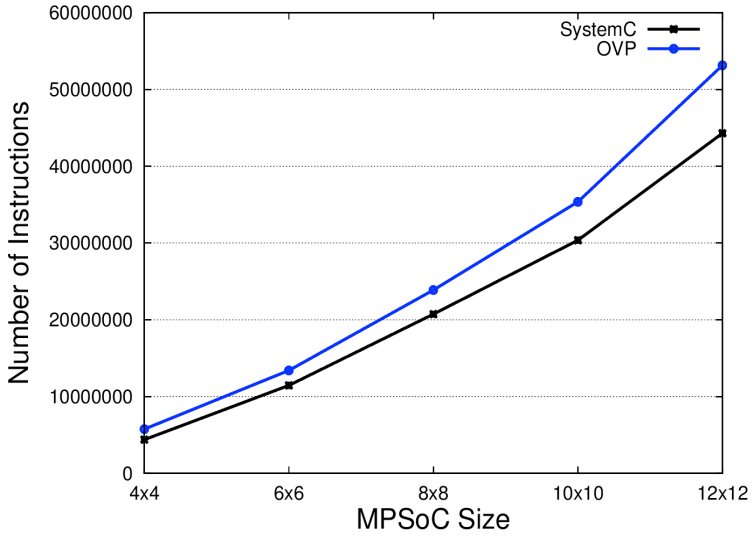
Figure 16(c) presents the simulation time for all evaluated scenarios. Simulation time is the real time required to finish the simulation. Note that the number of tasks for each scenario is approximately equal to the number of PEs, corresponding to a light workload. Even with this light workload, the OVP model achieves a speedup of five times compared to the SystemC model.

Beyond boosting the software validation, simulation speed of OVP makes it an attractive option for designers capturing initial performance indicators, which may be taken into account for further design decisions at early design phases. Achieved simulation speedup was expected because OVP executes at the system level. The execution time presents an error of 10 to 20% depending on the system size, which is an acceptable error for high-level models.

(a)



(b)



(c)

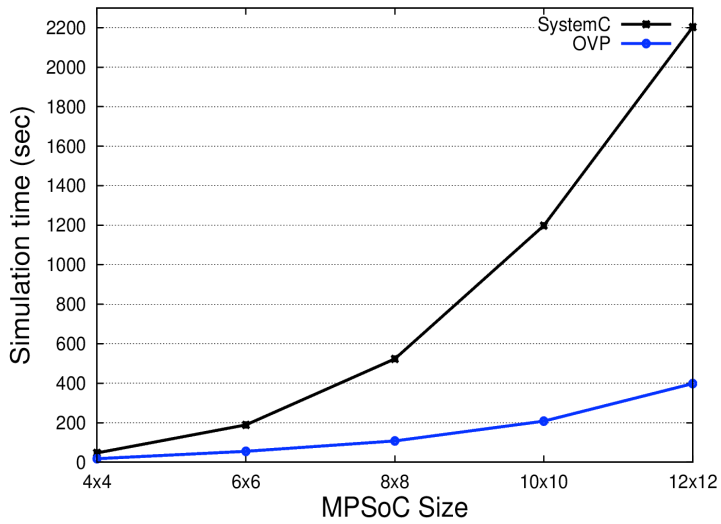


Figure 16 – Comparison between the SystemC and OVP models, where (a) presents the execution time, in clock cycles, for all scenarios; (b) presents the total number of simulated instructions for all scenarios; and (c) presents the simulation time for all scenarios.

Other experimental results were conducted comparing the simulation time of the SystemC and OVP models. Four scenarios executing 22 partial MPEG decoders with 5 tasks each onto different platform sizes: 6x6, 12x12, 16x16, and 20x20. Table 13 shows the obtained results, including simulation time in seconds and speedup obtained by the OVP model. The 12x12 scenario shows a speedup of 58.41.

Table 13 - Simulation time speedup, comparing SystemC and OVP platforms. Simulations setup – processor: Core 2 Duo E4400 2x2GHz; memory: 3GB; gcc version: 4.7.2; gcc flags: -mfpmath=sse -Ofast -flto -march=native -funroll-loops.

	6x6	**12x12**	**16x16**	**20x20**
SystemC	1080	9521	18289	28207
OVP	129	163	691	1247
speedup	8.37	58.41	26.47	22.62

Qualitative measures include flexibility and debuggability. The proposed model benefits from the high debuggability features supported in OVP, which provides a general view of each CPU model (e.g. registers, addressing, interrupts). Thus, software engineers can integrate the proposed OVP with GDB or Eclipse, accessing their debugging functionalities. For instance, the engineer can execute an application in single step mode (i.e. step-by-step), insert code breakpoints, observe variables values, etc. It is also possible, to set watchdogs for accessing defined memory regions, obtaining the fetched instruction or the read/write values. Such important features for software development do not exist in the SystemC or VHDL platforms.

3.4 Final Remarks

This Chapter presented the first contribution of this Thesis: the development of two model platforms, which can be used to exploring large MPSoCs. References related to this part of the work include [PET12][MAN12][MAN13][CAS13].

The quest for fast simulation time, enabling to predict important performance figures as execution time and energy consumption, was the driver of this part of the work. In 2011, the VHDL model enabled to validate system up to 42 PEs (7x6). In 2012, the SystemC model enabled the simulation of 144 PEs (144 PEs). In 2013, the mixed SystemC/OVP extended the MPSoC size to 256 PEs (16x16). Finally, in 2014, the OVP model allowed the simulation of 400 PEs, with a speedup to the SystemC model equal to 22.62.

Such result paves the way to the second part of the work, proposal of run-time and distributed mapping techniques for large scale MPSoCs, targeting communication volume reduction and workload balancing.

4. SYSTEM MANAGEMENT

This Chapter describes the management infrastructure implemented to support mapping techniques. The next Chapter details the heuristics, according to the steps defined in this Chapter.

This Chapter introduces in Section 4.1 and 4.2, respectively, the energy and application models used in this Thesis. Next, it discusses two approaches for system management: centralized, presented in Section 4.3; and distributed, presented in Section 4.4. The distributed system management approach is implemented to improve scalability and performance. Task mapping techniques in both approaches are the focus of this Section since they are essential in system management. Section 4.5 compares centralized and distributed management approaches. Section 4.6 concludes this Chapter.

4.1 Energy Model

The energy consumption in the MPSoC is mainly due to three components: memory, processors, and NoC. The number of memory accesses is identical for the same workload. Therefore, to fairly compare different mapping solutions using the same workload, it is enough to consider the energy consumption of both processor and NoC as main metrics.

As described in the literature [JEJ04], the energy consumption (*EC*) of a processor pe_i is defined by static and dynamic consumption. The processor *EC* related to the execution of a given task is a function of the number of executed instructions. In our model, the energy cost of each instruction is determined from a gate-level implementation, as proposed by Rosa et al. [ROS14].

Each processor pe_i contains an *instruction analyzer* module, which includes the energy cost of each instruction. Such module counts and classifies the executed instructions for different classes at runtime. The set of classes is defined as $C = \{c_0, c_2, \ldots, c_8\}$, with 9 different classes (e.g. arithmetic, logic, branch) [ROS14]. Equation (1) presents the energy dissipation for a given task.

$$energy_{task} = \sum_{i=0}^{8} (energy(c_i) \times total\_instructions(c_i)) \quad (1)$$

where: $energy(c_i)$, energy to execute a given instruction belonging to the class c_i, value obtained from simulating the synthetized processor; $total\_instructions(c_i)$, number of executed instructions belonging to the class c_i .

Results show that the accuracy of adopted *instruction analyzer module* varies from 0.06% to 8.05% when compared to a gate-level implementation [ROS14]. One can argue that this profile step may be inaccurate since the task workload may vary, according to the user data. During the profiling step, each application receives workloads representing real

execution scenarios. Therefore, the computed energy values are representative enough to guide the proposed mapping heuristic [TIW94][MUR07].

It is necessary to compute the power per *time slice* to evaluate the mapping heuristics using the *instruction analyzer* module employed during the profile step, now at runtime. Equation (2) computes the power consumption for each pe_i.

$$total\_power_{pe} = \frac{\sum_{i=0}^{8} (\text{energy}(c_i) \times total\_instructions(c_i))}{time\_slice} \quad (2)$$

where: *time_slice* is the power sampling period, in seconds.

The NoC *EC* is proportional to the number of transmitted flits at each router port [MAR14]. A gate level description of the NoC is used to determine the power consumption for the main router components: buffers, internal crossbar and control logic. Equation (3) presents energy consumption corresponding to one flit being transmitted through 1 buffer of the router.

$$E_{router} = \left[(n\_ports - 1) * P_{buffer}(0) + P_{buffer}(1) + P_{crossbar}(1) + P_{control\_logic}(1) \right] \times T \quad (3)$$

where: *n_port* is the number of ports of the router, $P_{component(0)}$ the average power without traffic, $P_{component(1)}$ the average power with an injection rate equal to 100%, *T* the clock period.

Equation (4) presents the power consumption of a given router for a given number of simulated cycles (*time_slice*), considering the number of flits transmitted by the router in the sampling period.

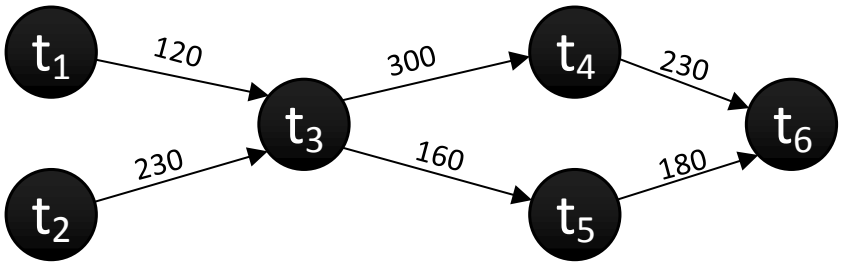
$$total\_power_{router} = \frac{E_{router} \times \sum flits}{time\_slice} \quad (4)$$

4.2 Application Model

An application app_i is modeled as an acyclic directed graph GApp = (T, E), where each vertex $t_i \in T$ represents an application task and each directed weighted edge $e_{ij} \in E$ represents a communication dependence between tasks t_i and t_j. The weight of an edge e_{ij} is denoted by $comm_{ij}$, representing the total data communication volume transferred between application tasks t_i and t_j. The mapping of the set of tasks $T = \{t_1, t_2, ..., t_n\}$ of GApp onto the set $SP = \{sp_1, sp_2, ..., sp_k\}$ of GMPSoC is defined by the mapping function: $T \rightarrow SP$, where $\forall t_i \in T, \exists sp_j \in SP$.

Figure 17 presents an example of an application modeled as a task graph. An application has *initial tasks* (e.g. t_1 and t_2) and *non-initial tasks*. Initial tasks are those that initialize the execution of the application when mapped in the system. Such tasks do not

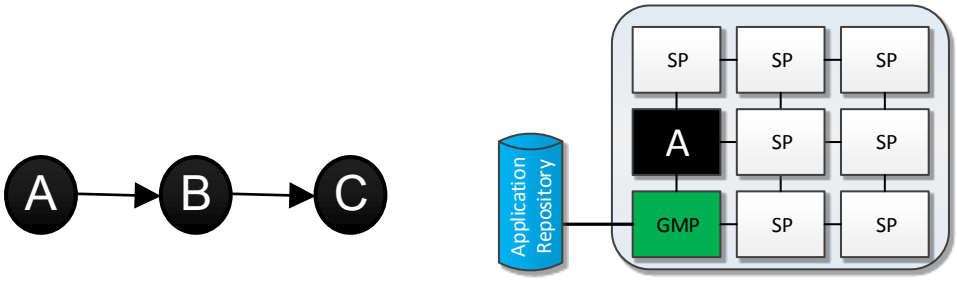
have dependences on other tasks to start executing.



Initial tasks: t_1, t_2 Non-initial tasks: t_3, t_4, t_5, t_6

Figure 17 - Application modeled as a task graph GApp = (T, E).

The mapping of non-initial tasks occurs whenever a given task t_i needs to communicate with a non-mapped task t_j. For example, Figure 18(a) shows an application with three tasks, being *A* the initial task. When such application is required to be executed, task *A* is mapped in the system to start the application execution, as illustrated in Figure 18(b). Task *A* starts its execution and at a given moment it needs to communicate with task *B*. Task *B*, a non-initial task, is not already mapped. Then, a task mapping algorithm selects an SP to map task *B*.



(a) (b)

Figure 18 – Initial task mapping.

In this context, the set T is divided in two subsets *iT* and *niT*, where (*iT* U *niT*) = T. The subset *iT* contains the initial tasks and the subset *niT* contains non-initial tasks.

A task $t_i \in T$ contains:

- a set C_i called communication task list. This set is defined as $C_i = \{(t_j, comm_{ij}); (t_k, comm_{ik}); \ldots (t_n, comm_{in})\}$, where each element is a tuple containing a task t_j that communicates with t_i and the value $comm_{ij}$, corresponding to the total volume transferred between t_i and t_j in both directions (i.e. t_i to t_j and t_j to t_i). Elements in a communication task list are sorted from the higher (first in the list) to the lower communication volume.
- an energy value E_i related to the execution of this task on the target SP. This value is captured according the energy model described in the previous Sub-section (Equation (1)).

An application app_i has an *application description* file containing information used to guide mapping decision. Such file contains:

(i) the application size defined by app_size_i, which corresponds to the total number of tasks of this application;
(ii) list of application initial tasks;
(iii) the sets C_i and E_i for each task t_i, of the application.

The application description file, including C_i and E_i for each task, is obtained from a *profiler* platform. The profiler platform is a slightly modified version of the OVP MPSoC model presented in Section 3.3, which employs monitors to capture the consumed energy for each task and their inter-communication volume. A network packet monitor is used to capture data communication among tasks. For each transmitted packet, the monitors capture the source task identifier, the destination task identifier, and the packet size. With such information, it is possible to define the set $C(t_i)$ of each task. The value E_i is obtained as explained in Section 4.1. The executed instructions required by each task are counted and classified according to different classes. Then, task energy E_i is computed according Equation (1). Each application is executed in the profiler platform, without any disturbing traffic.

Figure 19 presents the profiler platform flow. A single application is executed in the profiler platform. At the end of the simulation, the profiler generates the application description file. Then, such information is attached to the application, and included in the application repository when the application is compiled to be executed in the system.

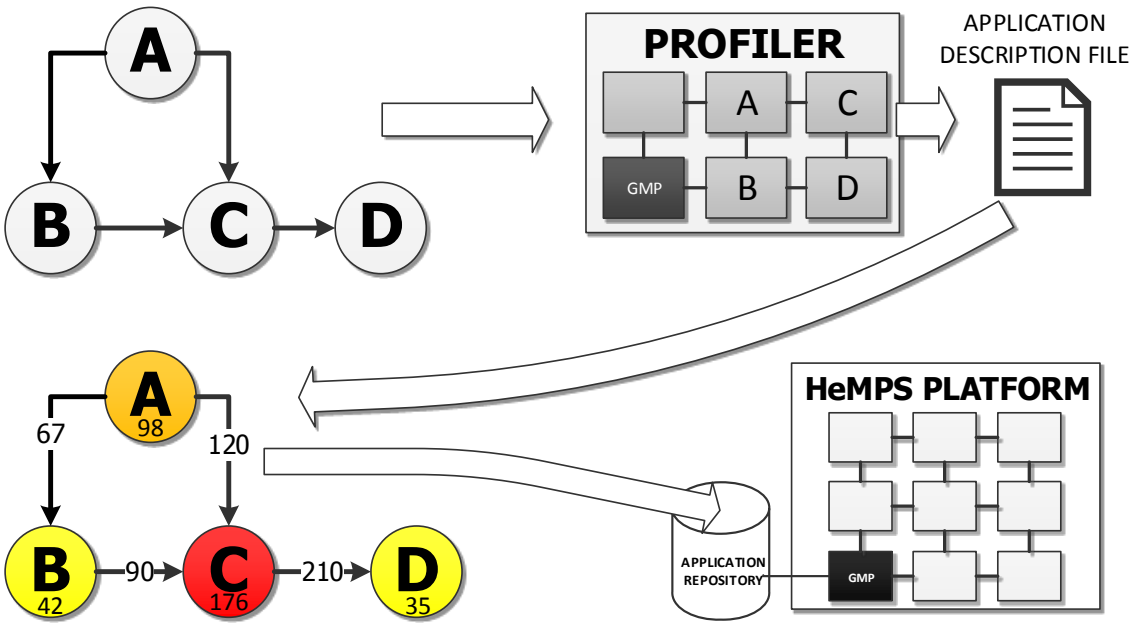


Figure 19 - Profiler platform flow.

4.3 Centralized System management

Figure 20 illustrates the centralized system management architecture. It contains the same components of the reference model presented in Section 3.1. Such approach uses only one Manager Processing Element: the Global Manager Processing Element (GMP), which is responsible for all management functions in the system. Such functions include receiving task requests from the NoC and computing mapping algorithms to select an SP for the requested task. Further, GMP maintains updated information about all system, including SPs where tasks are mapped, SPs availability, terminated tasks, and terminated applications. Such information is updated through information packets received from system SPs through the NoC.

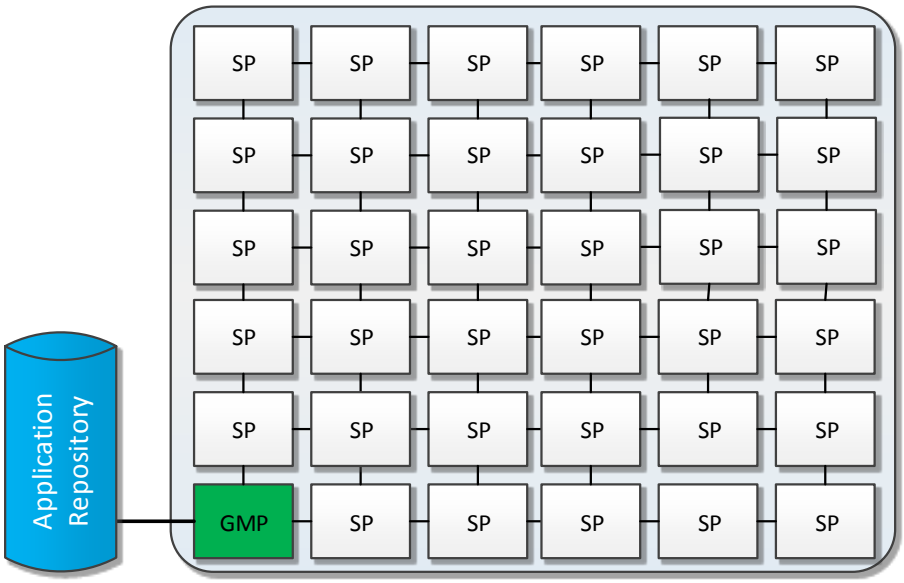


Figure 20 – Centralized system management architecture.

4.3.1 Centralized Task mapping protocol

The task mapping method used by the centralized approach has two steps: initial tasks mapping and non-initial tasks mapping.

4.3.1.1 Initial tasks mapping

Figure 21 illustrates the centralized mapping protocol for initial tasks. Whenever a new application is required to be mapped ("1 – New Application" in Figure 21), the system alerts the GMP, which verifies if the system has available resources to map the entire application. In not, the application is scheduled to be mapped later. Otherwise, first the GMP verifies the application description to obtain the application initial tasks.

Next, the GMP executes an algorithm to determine the mapping of the initial tasks ("2 – Initial Tasks Mapping" in Figure 21). When the algorithm selects an SP to map an initial task, the GMP obtains the object-code of such task from the application repository. Then, the GMP maps this task on the selected SP by using a *task allocation* packet message ("3 – Task Allocation" in Figure 21).

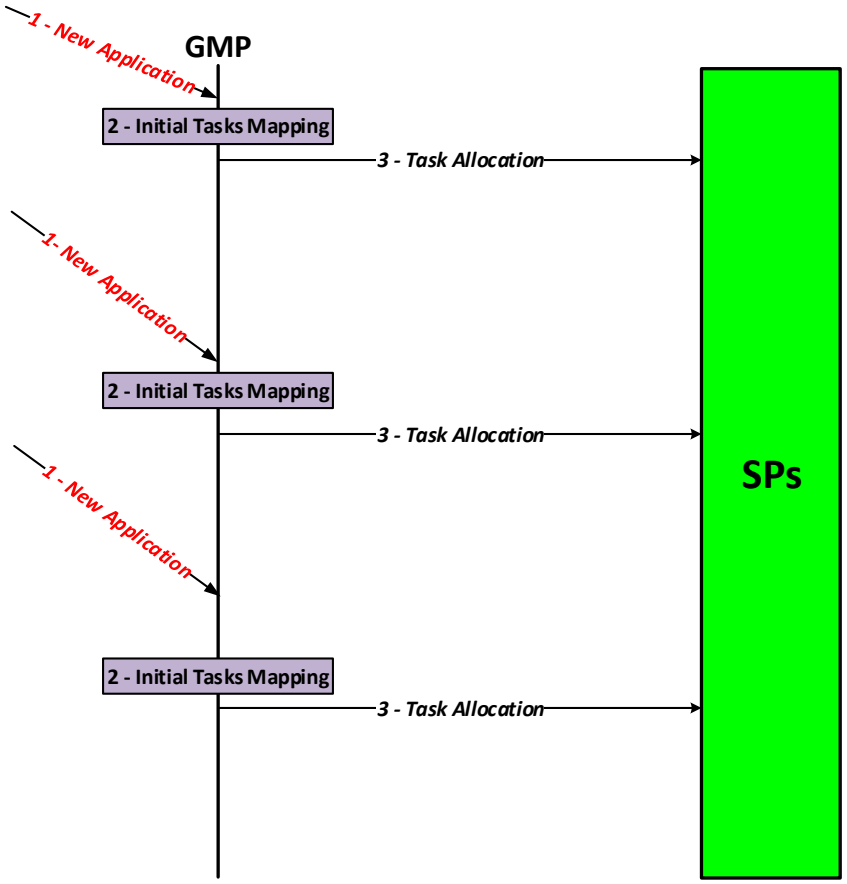


Figure 21 – Centralized initial tasks mapping protocol.

4.3.1.2 Non-initial tasks mapping

An application starts after the initial task mapping. Then, non-initial tasks are mapped, as illustrated in the example of Figure 22. In such example, an initial task t_1 executes the code shown in Figure 23. Task t_1 starts executing some functions, and then it communicates with task t_2 using an MPI-like primitive send (line 5 of Figure 23). The operating system of SP_1 hosting task t_1 verifies if the target task (t_2) is present in a task table, which contains the SP each task is mapped. If it is present, the message is transmitted to the SP assigned to task t_2 in the task table. If it is not, a packet with a task request service is transmitted to the GMP ('1 – Task Request' in Figure 22), asking the mapping of task t_2.

Receiving the task request, the GMP executes a task mapping heuristic to select the SP to receive t_2 ('2 – Task Mapping Algorithm',). Suppose the task mapping algorithm selected SP_2 to map task t_2. Thus, the GMP first obtains the object-code of task t_2 from the application repository. Then, the GMP transmits task t_2 object-code to SP_2 using a *"task allocation"* packet ('3 – Task Allocation, in Figure 22). Finally, the GMP sends to SP_1 the location of task t_2, and to SP_2 the location of task t_1. ('4 – Task Location' in Figure 22). These locations are stored in the task tables of each SP.

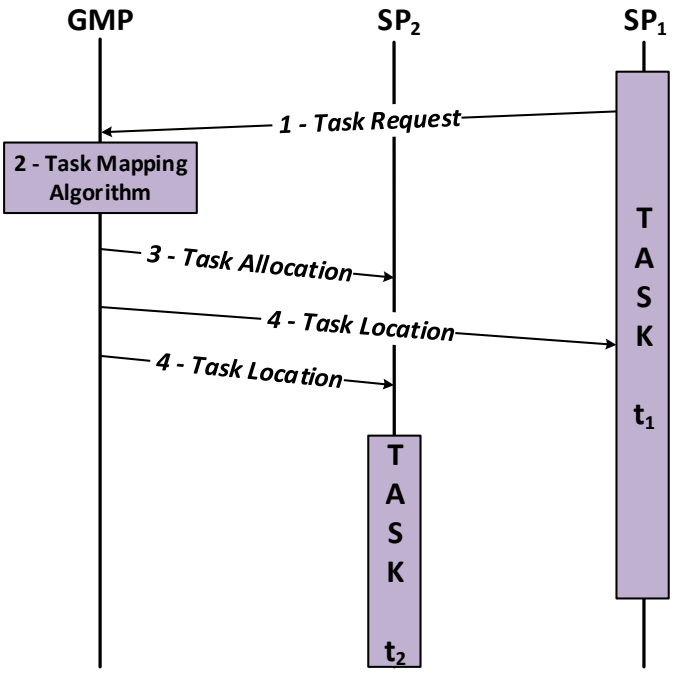


Figure 22 - Centralized non-initial tasks mapping protocol.

```
1. Message msg1;
2. int main(){
3.    msg1.length = 128;
4.    ...
5.    send(&msg1,t2); //communicate with t2
6.    ...
7. exit();
8. }
```

Figure 23 – Example of an initial task description, with a send command.

4.4 Distributed System management

As mentioned before, the constant growth in the number of cores implies in an important issue: scalability. Despite the scalability offered by NoCs and distributed processing, the MPSoC resources must be managed to deliver the expected performance. A single PE being responsible for system management may become a bottleneck since this PE will serve all other PEs of the system, increasing its computation load and creating a communication hot-spot region. An alternative to ensure scalability is to decentralize or distribute the management functions of the system. In light of this, this Section presents a distributed system management technique, which divides the system into regions (called *clusters*), improving system scalability and performance. This activity was co-developed with Guilherme Castilhos, Ph.D. advised by Fernando Moraes.

Two main approaches for distributed system management are discussed in the literature: (*i*) one manager per application [KOB11][SHA11]; (*ii*) one manager per MPSoC region, which are also called clusters [FAR08][ANA12][CUI12]. The proposed distributed management architecture relies on the second approach, dividing the MPSoC in equal-

sized clusters. Such distributed approach presents the following benefits:

I. The number of PEs dedicated to management functions is limited to the number of clusters. An approach using one manager per application may imply a larger overhead since the number of applications that will execute in the system is unknown at execution time.

II. The clustering approach reduces the number of hops among tasks belonging to the same application, reducing the overall traffic in the NoC (if the application fits in the cluster).

III. It is not necessary to create/destroy agents (manager PEs) each time a new application enters/leaves the system, enhancing in this way the overall system performance.

Figure 24 shows an example of the proposed distributed management architecture, using a 6x6 MPSoC instance with four 3x3 clusters. For this purpose, this approach uses two types of Manager Processing Elements:

- Local Manager Processing element (LMP) – responsible for cluster control, executing functions such as task mapping algorithm computation, and re-clustering.
- Global Manager Processing Element (GMP) – a single processing element responsible for the overall system management, such as defining application-to-cluster mapping, controlling external devices accesses (e.g. application repository). Further, the GMP manages one of the system clusters (for example, the left bottom cluster of Figure 24), executing all functions of an LMP.

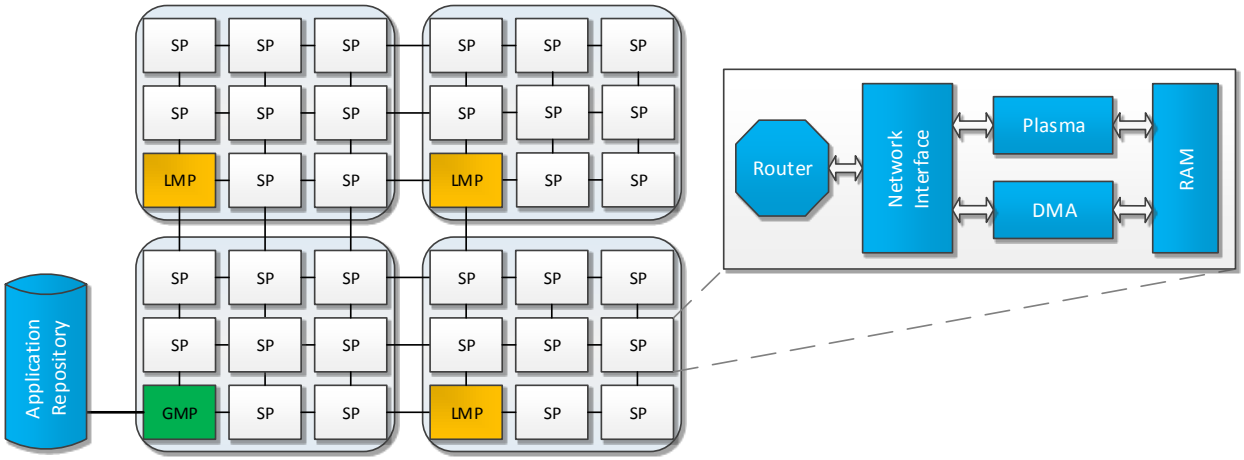


Figure 24 – Distributed system management architecture.

Slave processing elements (SPs) remain responsible for executing user's applications. In this context, the proposed distributed system management approach has three different PEs: LMPs, SPs and the GMP.

The definition of the clusters size occurs at design time. When the system starts, the GMP is responsible for cluster initialization, notifying the LMPs which region they will manage. Then, when an LMP knows the region it will control, it informs all SPs in this region that it will be their manager. This cluster and SPs initialization mechanism provide better system adaptability. For example, runtime re-clustering process, proposed by Castilhos [CAS13], enables the modification of the cluster size.

The re-clustering process occurs when there is no available SPs inside a cluster to map an application task. When a task is requested to be mapped to a given cluster, its LMP checks the availability of cluster SPs. If there are no SPs available inside the cluster to map the requested task, SPs are borrowed from neighbor clusters. When the task finishes its execution, the borrowed SP is released to the original cluster. The re-clustering process is better explained in Section 4.4.2.

The GMP is the only PE with access to the external devices (e.g the application repository). In Figure 24, four PEs are reserved for management functions, representing 11.1% of PEs not executing user applications. Using a 4x4 cluster in a 16x16 MPSoC, this overhead becomes 6.25%, which is an acceptable cost, considering the obtained benefits, as demonstrated next, with the evaluation of the proposal.

4.4.1 Distributed Task Mapping Protocol

The distributed task mapping protocol is divided into three main steps. Above the initial tasks and non-initial tasks mapping steps present in the centralized protocol, the distributed protocol has a cluster selection step. The cluster selection step occurs before the other steps, defining a cluster to map a required application.

4.4.1.1 Cluster Selection

As the centralized protocol, whenever a new application is required to be mapped, the system alerts the GMP (‘1 – New application’, in Figure 25). The GMP verifies if the system has available resources to map the entire application. If there are no available resources, the application is scheduled to be mapped later. Otherwise, the GMP selects a cluster to map the required application (‘2 – Cluster Selection’, in Figure 25). Then, the required application description is sent to the LMP of the selected cluster. Once a given cluster is selected, the GMP obtains the application description from the application repository, transmitting it to the selected cluster LMP (‘3 – Application Desc.’, in Figure 25).

4.4.1.2 Initial task mapping:

The LMP of the selected cluster receives and stores the application description. Then, such LMP verifies the application description to determine the application initial tasks. Next, the LMP computes a mapping algorithm to select SPs to map the application initial tasks inside the cluster (‘4 – Initial Tasks Mapping’, in Figure 25). The mapping of initial tasks starts the application execution.

After selecting an SP to map an initial task, the LMP sends a packet to the GMP with the service *task allocation request* ('5 – Task Allocation Request', in Figure 25). Such packet requests the allocation of the initial task object-code in the selected SP. This happens since the GMP is the only PE with access to the application repository. Then, the GMP obtains the task object-code from the application repository and transmits it to the selected SP ('6 – Task Allocation', in Figure 25). The SP will schedule the new task at the end of the "*task allocation*" packet reception. In addition, the LMP keeps a data structure, named task table, with the address of all mapped tasks.

Consider in Figure 25 the third application insertion. This situation illustrates a scenario where the selected cluster is the one managed by the GMP it-self. In this case, the GMP also executes the initial task mapping algorithm.

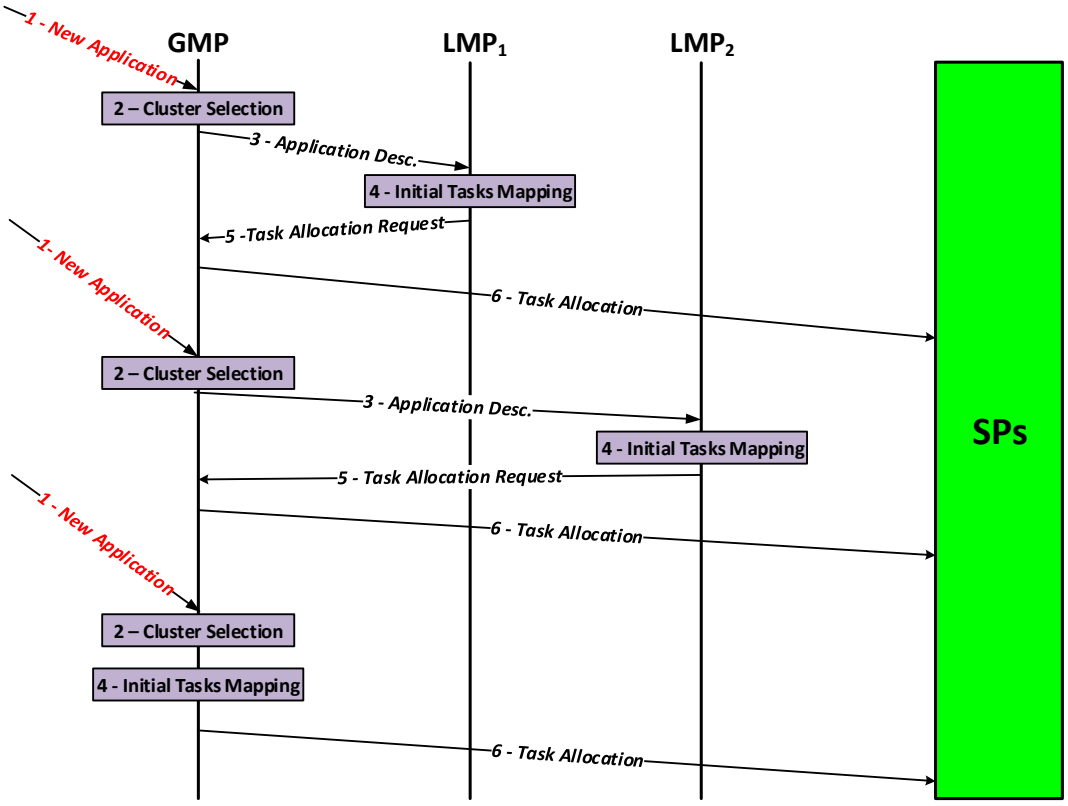


Figure 25 - Protocol to insert new applications into the system.

4.4.1.3 Non-initial tasks mapping:

As explained before, the mapping of non-initial tasks occurs whenever a given task t_i needs to communicate with a non-mapped task t_j. Suppose the example of Figure 26, where task t_1, mapped on SP_1, needs to communicate with a non-mapped task t_2. In this case, task t_1 requests the mapping of t_2 to its cluster LMP (LMP_1) by sending a Task Request packet message ('1 – Task Request', in Figure 26). LMP_1 receives the task request and executes a mapping algorithm to select an SP to map task t_2 ('2 – Task Mapping Algorithm', in Figure 26). The mapping algorithm selects SP_2 to map task t_2.

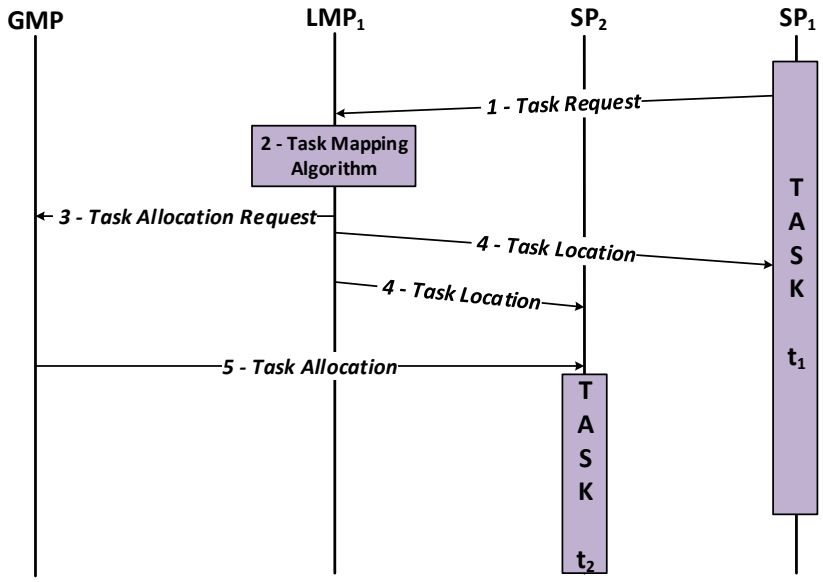


Figure 26 – Distributed non-initial mapping protocol.

Next, LMP_1 request the mapping of task t_2 on SP_2 to the GMP by sending a "Task Allocation Request" service packet ('3 – Task Allocation Request' in Figure 26). The LMP also uses a "Task Location" service packet to inform to SP_1 the location of t_2, and to SP_2 the location of task t_1 ('4 – Task Allocation', in Figure 26). These locations are stored in the SPs task tables. Finally, the GMP obtains task t_2 object code from the application repository and transmits it to SP_2 ('5 – Task Allocation', in Figure 26).

4.4.2 Re-clustering Process

Figure 27 presents an example of the re-clustering process, assuming the SP sp_{RM} (in red in Figure 27) requested the mapping of a given task to LMP_2 (the LMP that manages sp_{RM} cluster). The LMP_2 receives the task mapping request and verifies that there are no available SPs inside this cluster. In this case, the LMP_2 sends a "loan request" message, requesting an available SP to all neighbor clusters LMPs (GMP, LMP_1, and LMP_3; as shown in step 1 of Figure 27).

When receiving the "loan request" message, the neighbor clusters LMPs search for available SPs in their clusters. If there is only one available SP, this SP is reserved to be borrowed; otherwise, if there is more than one available SP, such LMPs reserve the closest one, in number of hops, to sp_{RM} (i.e. the SP that requested the mapping). After the reservation, such LMPs send a "loan delivery" message to the LMP_2 (cluster that requested available SPs), notifying the possible borrowed SP position (step 2 of Figure 27, blue SPs are reserved), if it exists.

The LMP chooses the closest SP from sp_{RM} (i.e. the SP that requested the mapping), sending a "loan release" message to all LMPs, which were not selected (step 3 of Figure

27). Next, the LMP send a “task allocation request” message to the GMP requesting the task mapping on the borrowed SP (step 4 of Figure 27). Therefore, the cluster size increases at runtime, because the borrowed SP is now part of this cluster. This process optimizes the system management, since applications can be mapped in clusters, even if the cluster has no sufficient SPs.

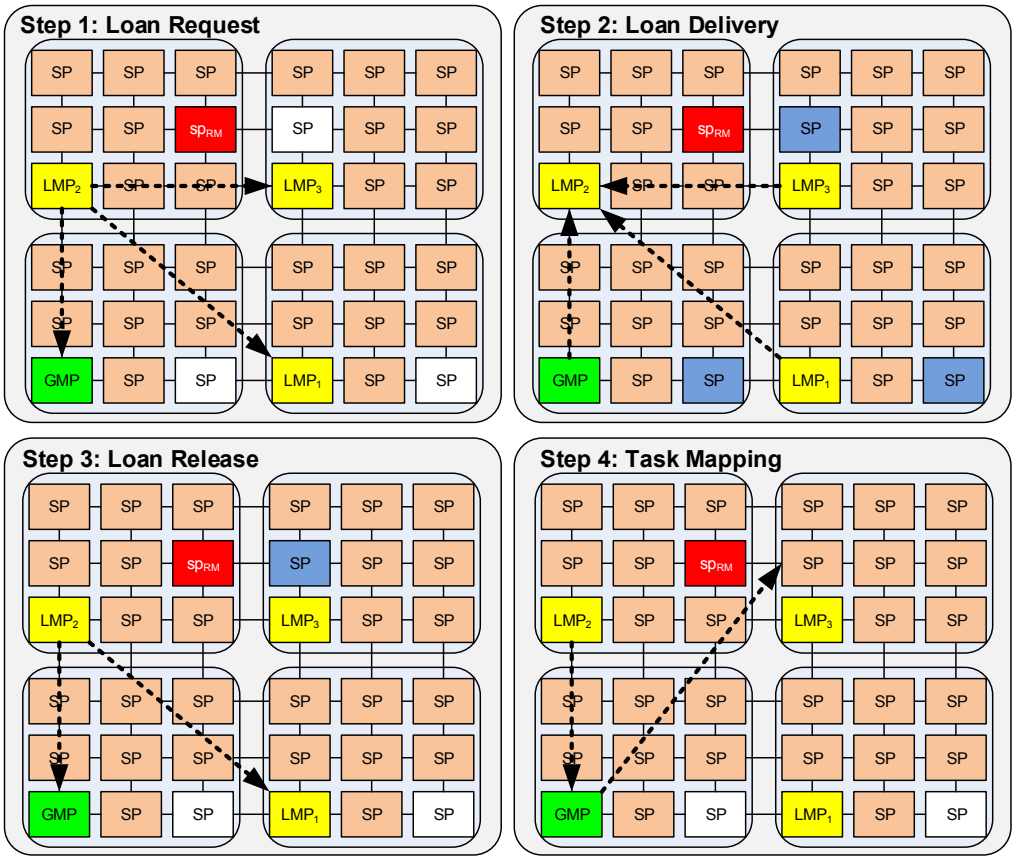


Figure 27 - Task mapping protocol, using PEs in neighbor cluster. White SPs (slave PEs) are available PEs [CAS13].

4.5 Centralized versus Distributed Task Mapping

In the centralized mapping, the GMP is responsible for computing the mapping of all tasks. In this case, all incoming mapping requests are serialized (Figure 28(a)), reducing the system performance, and increasing the NoC traffic in the GMP region. Using the distributed task mapping (Figure 28(b)), the mapping computation is distributed in several LMPs, reducing the communication load generated by mapping requests.

It is important to mention a limitation of the distributed approach. Even if the mapping is distributed, the access to the external world (application repository) is not. Transmitting the task data in burst, using a DMA approach, minimizes the impact of this issue.

Experiments use the SystemC platform, adopting clusters with 8 SPs (3x3 clusters). Each SP can execute up to 2 simultaneous tasks. Therefore, each cluster may execute 16 tasks. Three benchmarks were used: *MPEG*, executes a partial MPEG decoder; *multispec*

image analysis [TAN08], which evaluates the similarity between 2 images using different frequencies; and, a synthetic application (*synthetic*).

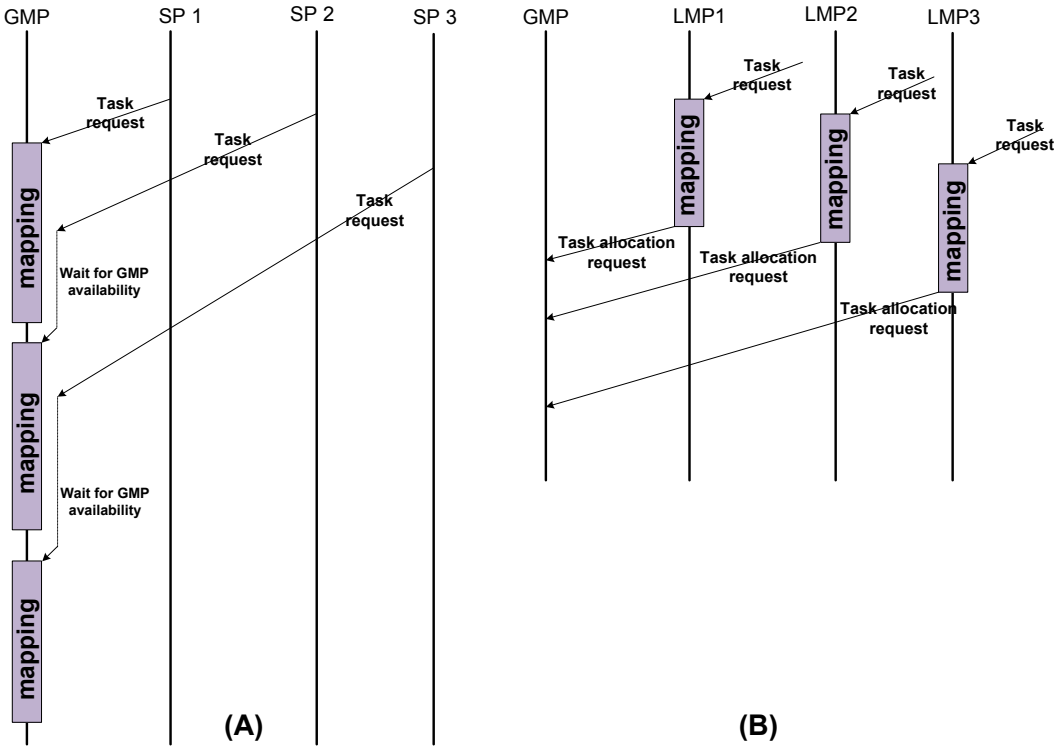


Figure 28 - Centralized (a) versus distributed (b) mapping.

Table 14 presents the characteristics of the nine evaluated scenarios (A, B, …, I). The second column of the Table contains the MPSoCs size, the number of clusters, and the number of SPs for the distributed and centralized management approaches. Note that the centralized approach has more SPs than the distributed one, since it does not use LMPs. The third column presents the number of tasks for each benchmark, while the forth column shows the number of application instances (App_{CL}) that fit in the cluster. The fifth column contains the total number of tasks that must be mapped onto the MPSoC platform. The last two columns present the system usage (SU) for the distributed and centralized approaches, i.e., the percentage of used system resources (nb of tasks / (nb of SPs * 2)).

Table 14 - Characteristics of the evaluated scenarios

	MPSoC Size	Benchmark -Nb. of Tasks	App_{CL}	Total number of tasks	SU_{dist}	SU_{centr}
A	**6x6** - 4 clusters	Syntetic - 6	2	48	75%	69%
B	- 32 SPs (distributed)	MPEG - 5	3	60	94%	86%
C	- 35 SPs (centralized)	Multispec -14	1	56	88%	80%
D	**9x9** - 9 clusters	Syntetic - 6	2	108	75%	68%
E	- 72 SPs (distributed)	MPEG - 5	3	135	94%	84%
F	- 80 SPs (centralized)	Multispec -14	1	126	88%	79%
G	**12x12** - 16 clusters	Syntetic - 6	2	192	75%	67%
H	- 128 SPs (distributed)	MPEG - 5	3	240	94%	84%
I	- 143 SPs (centralized)	Multispec -14	1	224	88%	78%

All applications instances are inserted in the system at the same time (1 ms) to maximize the use of SPs. Table 15 presents the total execution time reduction, adopting the centralized mapping as reference. Such results are a clear demonstration of the poor scalability related to the centralized management of the MPSoC resources. Scenario A does not reduce the total execution time using the distributed mapping due to the smaller system utilization. As the centralized approach has more free resources, some SPs may receive one task, instead of two, reducing the application execution time, since there is no time-sharing between tasks. Note that this behavior also occurs in scenarios D and G, where the synthetic benchmark presents smaller gains than scenarios E/F and H/I, respectively. The mapping process in the distributed approach has a smaller search space. For example, in the 12x12 MPSoC the centralized mapping has to evaluate the status of 143 SPs, while in the distributed mapping the search space is always the same, proportional to the cluster size. Therefore, the execution of the mapping heuristic is faster in the distributed version.

Table 15 – Total execution time reduction, adopting the centralized mapping as reference.

Scenario	MPSoC Size	Benchmark	Execution time reduction (w.r.t centralized mapping)
A	6x6	Synthetic	**-15% (increase of time)**
B		MPEG	**28%**
C		Multispect	**34%**
D	9x9	Synthetic	**50%**
E		MPEG	**63%**
F		Multispect	**54%**
G	12x12	Synthetic	**79%**
H		MPEG	**86%**
I		Multispect	**85%**

Figure 29 presents the execution time (in clock cycles) for each application instance, for scenarios B and E. It is important to observe in Figure 29 that all application instances have roughly the same execution time. Note that in the distributed mapping (black bars) a set of applications starts simultaneously. This is due to the distributed computation of the mapping heuristic, as illustrated in Figure 28(b). On the other hand, the centralized mapping (white bars) has to map the application tasks (5 tasks in the MPEG benchmark) and treat the request for new applications. This serialization of the mapping process is clearly observed in both figures, which also explain the results observed in Table 15. As the MPSoC size increases, the execution time for the centralized mapping grows dramatically.

Another benefit of the promoted distributed approach is the reduction of the traffic around the GMP. Most control messages are treated inside the clusters, and the only control message sent to the GMP is the task request.

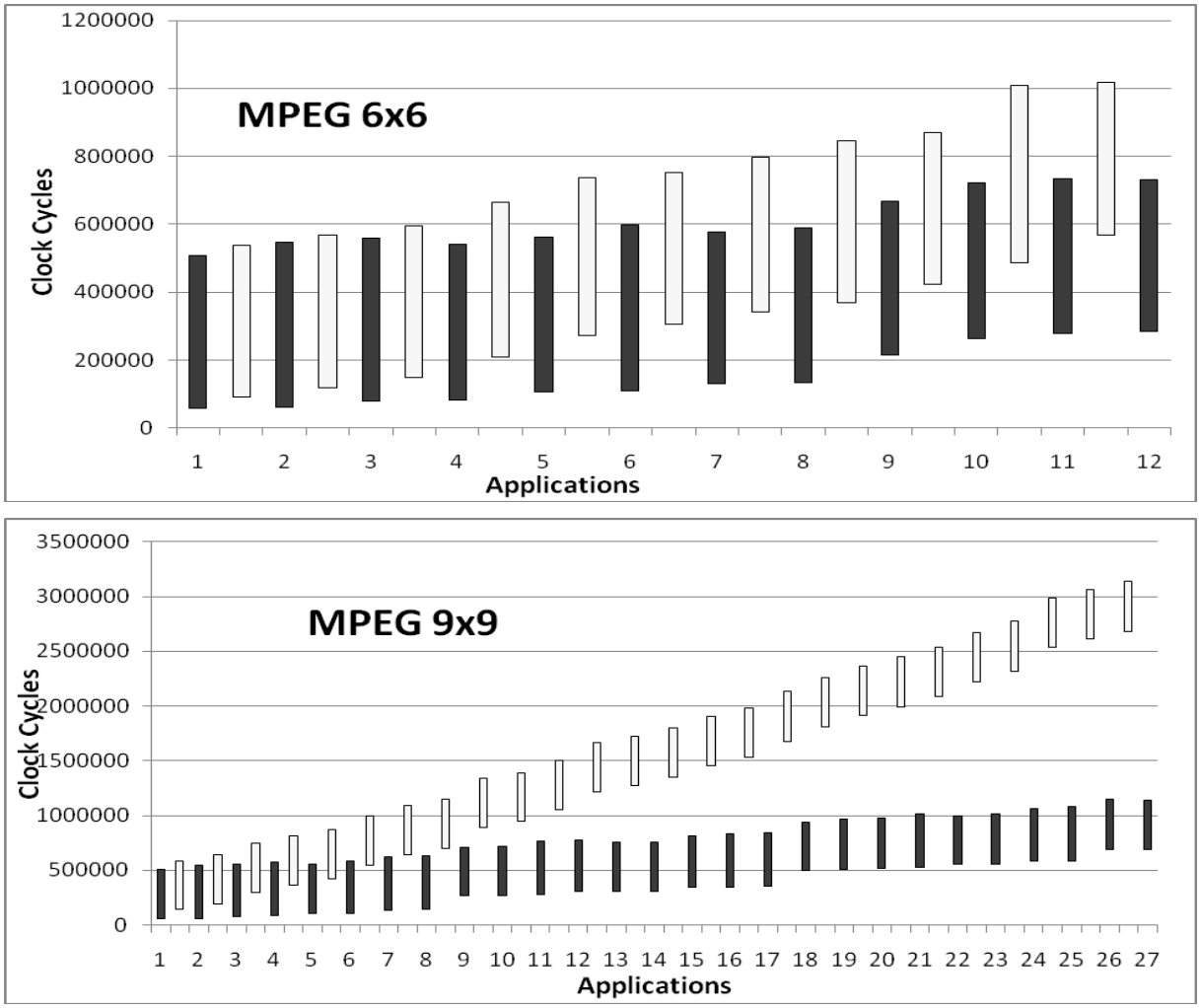


Figure 29 – Execution time for distributed (black bars) and centralized mapping (white bars), for the MPEG benchmark with two NoC sizes, scenarios B and E.

4.6 Final Remarks

This Chapter introduced two important features required by the mapping heuristics: (*i*) distributed system management; (*ii*) mapping protocol.

The distributed management ensures scalability at the cost of sacrificing some processor to management functions.

The mapping protocol comprises three steps: (*i*) cluster selection; (*ii*) initial task mapping; (*ii*) non-initial task mapping. The next Chapter presents a set of mapping heuristics, using the distributed management approach with the mapping protocol detailed in this Chapter.

5. TASK MAPPING HEURISTICS

This Chapter presents four distributed runtime task mapping heuristics. Sections 5.1 and 5.2 describes, respectively, the LEC-DN and PREMAP-DN heuristics. Such heuristics, previously presented in [MAN11b], were extended in this Thesis to support the distributed mapping protocol. The goal of the LEC-DN heuristic is to reduce the communication volume in the NoC. This heuristic maps communicating tasks as close as possible. The PREMAP-DN heuristic improves the LEC-DN heuristic, reducing even more the communication through the NoC. PREMAP-DN uses the PREMAP clustering method, which tries to group communicating tasks in the same SP. Sections 5.3 and 5.4 propose the new Load (L) and Load-Communication (LC) heuristics. The goal of the Load heuristic is to distribute the workload evenly, improving in long-term the system reliability. The Load-Communication heuristic mixes LEC-DN and L heuristics, making a trade-off between communication volume reduction and workload distribution. The heuristics described in this Chapter use the three mapping steps of the distributed mapping protocol, presented in Section 4.4.1. Section 5.5 evaluates the heuristics.

The heuristics use the following definitions:

Definition 1: *application size* (app.size) corresponds to the number of tasks of the application to be mapped.

Definition 2: *MAX_SP_TASKS* is the maximal number of tasks a given SP may execute simultaneously. The SP local memory is organized into SP_PAGES pages, being one reserved for the operating system. Therefore, MAX_SP_TASKS = SP_PAGES - 1.

Definition 3*: available_resources* corresponds to the number of *resources* (a resource is a page in the memory) that do not have a task mapped on it. This information may refer to the whole system, *available_resources*(*system*), or to a given cluster c_k, *available_resources*(c_k).

Definition 4: *available*(sp_i) returns *true* if sp_i is available to receive a new task, otherwise *false*. An SP is available when the number of tasks mapped on it is smaller than MAX_SP_TASKS.

Definition 5: *empty SP* is an SP with no tasks mapped on it. Therefore, an empty SP can receive MAX_SP_TASKS tasks.

The Load (L) and Load-Communication (LC) heuristics use specific definitions, *TE* and *cl_energy*(c_k).

Definition 6: *TE* is the total consumed energy by a given SP, corresponding to the energy (E_i) consumed by all already executed tasks and the tasks that are currently being executed on this processor. Whenever a task is mapped onto an SP, *TE* is updated.

Definition 7: *cl_energy*(c_k) corresponds to the consumed energy of a cluster c_k. This

function is computed by summing the *TE* value of each SP of the cluster.

5.1 LEC-DN

The LEC-DN heuristic reduces the communication volume through the NoC by nearing communicating tasks that exchange a high communication volume. Next sub-sections describe this heuristic according to the three distributed mapping protocol steps.

5.1.1 Cluster selection

LEC-DN heuristic selects the cluster with the largest number of available resources to map an application. Figure 30 presents the pseudo-code of the cluster selection algorithm. The heuristic first verifies if the system has available resources to map the application (line 4). If there are insufficient resources in the system, the application is scheduled to be mapped later. Then, the loop between lines 5 and 10 analyzes all clusters, selecting the one with the largest number of available resources.

```
Input: application size app.size
Output: selected_cluster
1. selected_cluster ← -1
2. selected_cluster_resources ← -∞
3. //Verify if the system has available resources to map the application
4. IF available_resources(system) >= app.size THEN
5.   FOR EACH cluster c_k in the system
6.     IF available_resources(c_k) > selected_cluster_resources THEN
7.       selected_cluster ← c_k
8.       selected_cluster_resources ← available_resources(c_k)
9.     END IF
10.  END FOR EACH
11. END IF
12. return selected_cluster
```

Figure 30 – Cluster Selection algorithm used in LEC-DN and PREMAP-DN heuristics.

It is important to note that a given application only starts its execution when there are enough resources to map the whole application (line 4). The next steps of the mapping algorithm try to map the applications' task in the cluster. The re-clustering is responsible to extend the cluster size to enable the mapping of the initial and non-initial tasks when the cluster becomes full.

5.1.2 Initial tasks mapping

The initial task mapping evaluates all SPs inside the *selected_cluster*, selecting the SP with the largest *region_free*. The function *region_free(sp_i, n_hops)* returns the total number of available resources of the set containing sp_i and all SPs up to *n* hops far from

sp_i. Consider the example in Figure 31, which uses a 5x5 cluster, sp_i is the central SP (in green), and *n_hops* is equal to 2. In Figure 31, the colored SPs define an area 2 hops far from sp_i. Suppose, the numbers inside at each SP corresponds to the number of available resources. Then, the sum of these numbers corresponds to the *region_free*(sp_i, *2)*, which is equal to 25.

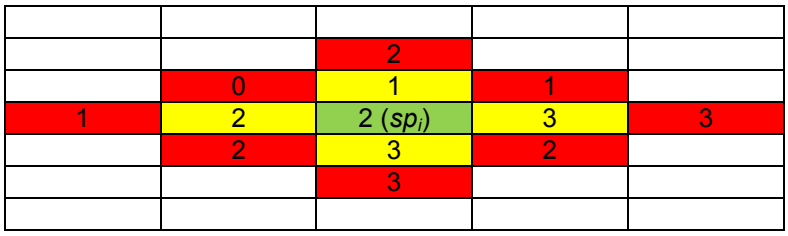


Figure 31 - Hypothetical example to compute the function *region_free*.

Figure 32 shows the pseudo-code of the initial tasks mapping algorithm used in LECD-DN. The loop between lines 3 and 8 evaluates all SPs inside the selected cluster and selects the SP with the largest *region_free*. If the application has more than one initial task, this algorithm is re-executed for the other initial tasks. The initial tasks mapping used in LEC-DN aims to map initial tasks in regions that have the largest number of available resources. This method increases the probability of mapping the application's tasks closer to each other, reducing communication volume.

Input: *n_hops*
Output: *selected_sp*
1. selected_sp ← *-1*
2. selected_region_free ← *+∞*
3. **FOR EACH** SP sp_i in the *cluster*
4. **IF** available(sp_i) **AND** region_free(sp_i, n_hops)>selected_region_free **THEN**
5. selected_sp ← sp_i
6. selected_region_free ← region_free(sp_i, n_hops)
7. **END IF**
8. **END FOR EACH**
9. return selected_sp

Figure 32 – Pseudo-code of the initial tasks mapping algorithm used in LEC-DN and PREMAP-DN heuristics.

5.1.3 Non-initial task mapping

When a non-initial task t_i is required to be mapped, the LECD-DN heuristic creates a list containing all tasks that communicates with t_i that are already mapped within the selected cluster. Next, this heuristic defines the search space to map t_i. Such search space is defined by a bounding box rectangle, containing the mapped tasks that communicate with t_i. When there is more than one mapped task, the bounding box is enlarged by one hop offering a larger search space. Figure 33 illustrates the mapping search space when one (Figure 33(a)) or two communicating tasks (Figure 33(b)) are

mapped in the cluster. The adoption of a bounding box aims to reduce the distance among communicating tasks. The bounding box size is enlarged by one hop, if no available SPs are found on it.

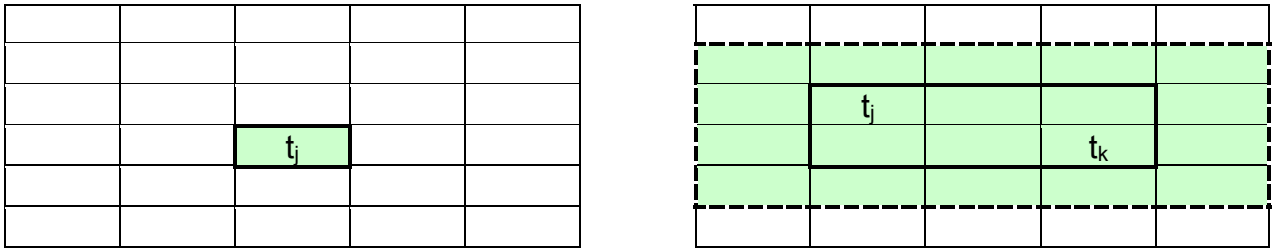


Figure 33 - (a) search space when one communicating task is already mapped (t_j); (b) search space when more than one communicating task is already mapped (t_j and t_k). Solid lines correspond to the original bounding box, dashed lines to the bounding box increased by one hop.

If just one communicating task t_j is mapped, the heuristic will map t_i on an SP as close as possible, in number of hops, to the one where t_j is mapped. Otherwise, the heuristic map t_j onto the SP with the lowest communication cost. The communication cost is based on the volume-based energy model proposed by Hu et al [HU10], where the energy spent in the communication is proportional to the number of hops and the number of transmitted flits.

Figure 34 illustrates the LEC-DN approach with one task already mapped. Consider the application of Figure 34(a), with three tasks. To start the application, task A (the initial task) is mapped. At a given moment, the mapping of task B is required. Task B has only one communicating task already mapped in the cluster (task A), as illustrated in Figure 35(b). Therefore, the search space to map task B corresponds to the bounding box with the SP of task A. If this SP is available, task B is mapped to it. Otherwise, the bounding box is enlarged by one hop, as defined by the red line in Figure 35(b). Inside such bounding box, each sp_i has a cost that represents the distance between task A and sp_i. The LEC-DN heuristic selects the first available SP with the smallest distance cost.

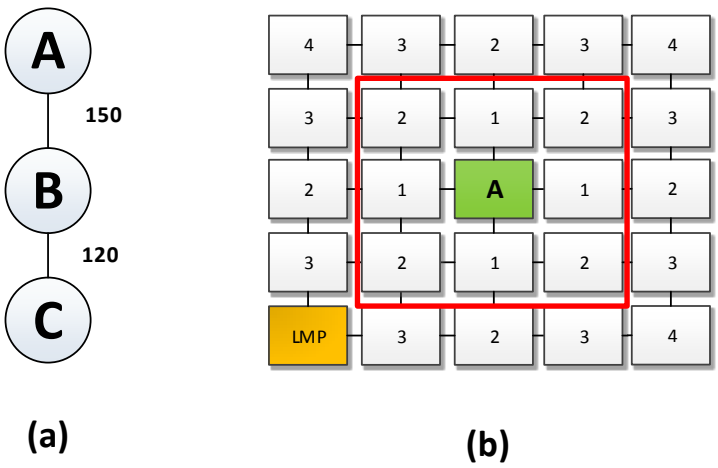


Figure 34 - (a) application graph of a given application; (b) search space to map task B.

Figure 35 illustrates the LEC-DN approach with more than one task already mapped. The application has 4 tasks, where A and B are initial tasks (Figure 35(a)). At a given moment, the mapping of task C is required. Thus, LEC-DN evaluates the list of tasks that communicate with task C that are already mapped. Task C has two communicating tasks already mapped in the cluster (tasks A and B), as illustrated in Figure 35(b). Therefore, the search space to map task C corresponds to the bounding box defined by the coordinates of tasks A and B increased by one hop (Figure 35(b)). An SP inside the bounding box has a cost that represents the total amount of communication volume that will be transferred by mapping the task C on this SP. This cost is computed considering the amount of data A and B transfer to C, and the distance in number of hops from a given SP to task A and B. For example, the cost of the SP to which A is mapped is 300, since:

- the SP is 0 hops far from A and A transfers 150 (it can be the number of flits or a given rate in Mbps) to C, so the volume on the NoC generated by the communication A→C is be 0 x 150 = 0;
- the SP is 3 hops far from B and B transfers 100 to C, so the volume on the NoC generated by the communication B→C is 3 x 100 = 300;
- the total communication volume generated by mapping C on this SP is the sum of the volume generated by A→C and B→C, resulting in 300.

Task C will be mapped on the processor that has the lowest cost, which is, in this case, the processor where A is mapped (assuming each SP may execute simultaneously more than one task). Note that task D is not yet mapped, since it depends from task C.

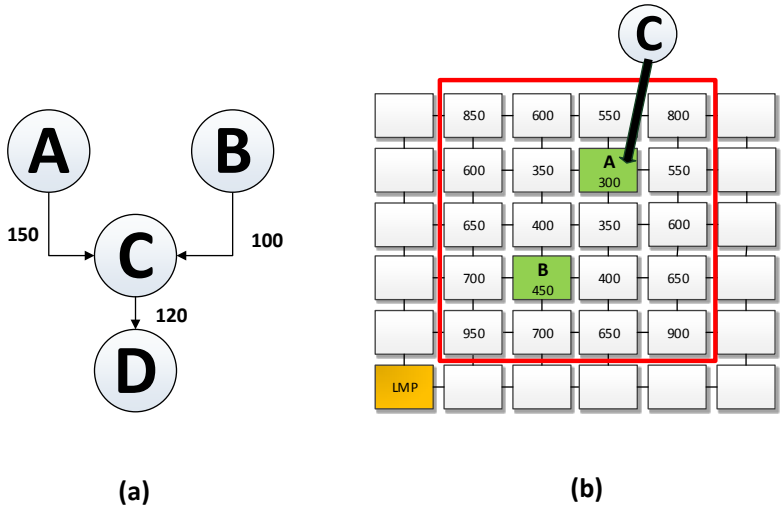


Figure 35 - (a) application graph of the application (b) search space to map task C, where each SP has a cost, and the final mapping of C.

Figure 36 presents the pseudo-code for the heuristic used to map non-initial tasks. When a non-initial task t_i is required to be mapped, LEC-DN starts analyzing the set C(t_i), containing all tasks that communicates with t_i. All mapped tasks within the cluster of C(t_i) are inserted in the set MC(t_i) (line 2). In the sequel, a bounding box rectangle (line 3) is

defined, according to the position of all tasks of MC(t_i). Next, the size of the set MC(t_i) is evaluated.

```
Input: t_i , set C(t_i)
Output: selected_sp
1.  selected_sp ← -1
2.  MC(t_i)← mapped_tasks(C(t_i))        // all tasks communicating with t_i already mapped
3.  bounding_box ← area(MC(t_i))
4.  IF |MC(t_i)| = 1 THEN
5.      selected_sp_distance ← +∞
6.      WHILE all SPs in the cluster were not evaluated AND selected_sp=-1 DO
7.          sp_list ← search_SPs(bounding_box)
8.          FOR EACH SP sp_i IN sp_list
9.              IF available(sp_i) = true  AND  evaluated_sp_distance < compute_distance(t_i, sp_i)  THEN
10.                 selected_sp ← sp_i
11.                 selected_sp_distance ← compute_distance(t_i, sp_i)
12.             END IF
13.         END FOR
14.         IF selected_sp = -1 THEN
15.             increase(bounding_box, 1)
16.         END IF
17.     END WHILE
18. ELSE IF |MC(t_i)|  > 1 THEN
19.     selected_sp_cost ← +∞
20.     increase(bounding_box, 1)
21.     WHILE all SPs in the cluster were not evaluated AND selected_sp=-1 DO
22.         sp_list ← search_SPs(bounding_box)
23.         FOR EACH SP sp_i IN sp_list
24.             IF available(sp_i) = true  AND  evaluated_sp_cost < compute_cost(t_i, sp_i)  THEN
25.                 selected_sp ← sp_i
26.                 selected_sp_cost ← compute_cost(t_i, sp_i)
27.             END IF
28.         END FOR
29.         IF selected_sp = -1 THEN
30.             increase(bounding_box, 1)
31.         END IF
32.     END WHILE
33.  END IF
34. return selected_sp
```

Figure 36 - Mapping of non-initial tasks used in LEC-DN and PREMAP-DN heuristics.

If MC(t_i) contains only one task (line 4), the loop between line 6 and 17 is executed. A list *sp_list* with the SPs of the bounding box is created (line 7). Next, the loop between lines 8 and 13 evaluates each SP of *sp_list*, selecting the available one with the lowest distance in hops to the only task that communicates with t_i inside the cluster. If no available SP is found in this list, the bounding box size is enlarged in one hop (lines 14-16), and another loop iteration is executed.

If MC(t_i) contains more than one task (line 18), the bounding box size is increased in

one hop (line 20). Then, the loop between line 21 and 32 is executed. This loop is similar to the previous loop (lines 6-17). However, the selected SP is the one inside the bounding box with the smallest communication cost. If no available SP is found, the bounding box size is enlarged in one hop, and another loop iteration is executed.

5.2 PREMAP-DN

The PREMAP-DN heuristic uses the same three mapping steps of LEC-DN heuristics. The difference relies on the integration of a PREMAP clustering method [MAN11b] to optimize the communication volume reduction. The goal of this method is to group a set of communicating tasks onto the same SP. When a given task is *pre-mapped*, its placement is just reserved. The effective mapping of the *pre-mapped* tasks is executed when the task is requested.

The integration of LEC-DN and the PREMAP method occurs as illustrated in Figure 37. Suppose a task *t* is required to be mapped. If task *t* is a non-initial task, it is verified if it is pre-mapped. If true, task *t* is mapped to the SP it was pre-mapped. If task *t* is an initial task or it is a non-initial that was not pre-mapped, LEC-DN is used to select an SP *selected_sp* to map this task. Then, task *t* is mapped to the SP *selected_sp*. If the SP *selected_sp* was empty before the mapping of task *t*, the PREMAP method is executed.

The PREMAP method is executed whenever an empty SP sp_i receives a task t_i. The method analyzes the communicating task list $C(t_i)$ to select the tasks to be pre-mapped onto sp_i. The communicating task list $C(t_i)$ contains all tasks that communicates with t_i sorted from the one that transfers the highest to the lowest communication volume, as explained in Section 4.2. PREMAP method follows the order $C(t_i)$ is sorted, evaluating task by task from such list. Suppose the first task to be evaluated is task t_k, the one that communicates most with t_i. Task t_k is pre-mapped in sp_i iff two conditions are satisfied:

(i) if t_k is a non-mapped task;

(ii) if task t_k communicates only with task t_i; or, if task t_k communicates with more than one task, t_i must be the task it transfers the highest communication volume, i.e., task t_i is first task of $C(t_k)$.

When a task is pre-mapped to an SP, a resource of this SP is *reserved* to receive such task. Thus, the number of available resources of such SP is decreased. Tasks are pre-mapped to an SP while it has available resources.

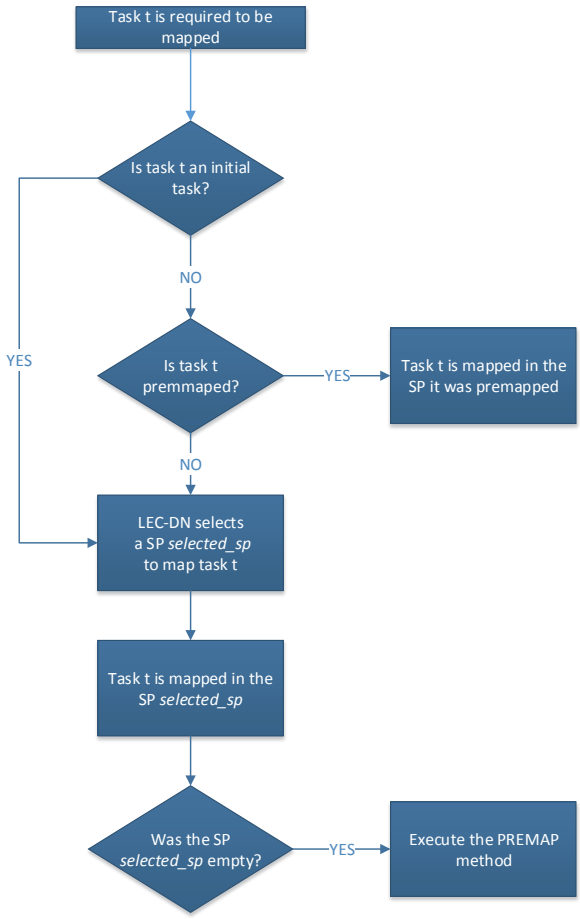
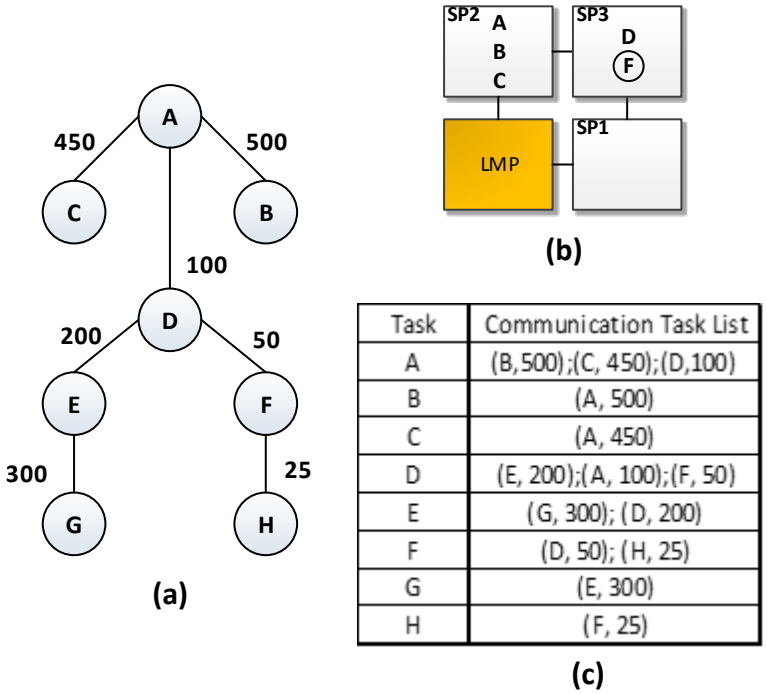


Figure 37 – Integration of the PREMAP method in the LEC-DN heuristic.

Consider as an example the application of Figure 38(a), with 8 tasks, being task A the initial task. A 2x2 MPSoC cluster is used in the example, where each SP is able to execute up to 3 tasks. Figure 38(c) presents the communicating task list C(t_i) for each task t_i of the application. When the application execution starts, the initial task A is mapped in SP2 (Figure 38(b)). Since SP2 was empty, the PREMAP method is executed. The method evaluates each task in the set C(t_A) to be pre-mapped to SP2. The first task to be evaluated is the one that exchanges the highest communication volume with t_A, which is t_B. Task B is pre-mapped since it is a non-mapped task that only communicates with t_A (see C(t_B)). For the same reason, t_C (the second in C(t_A) list) is also pre-mapped. At this moment the method stops, since the SP has already 3 tasks assigned to it, being unavailable for receiving new tasks. During system execution, tasks B and C are required to be mapped. Since such tasks were already pre-mapped, it is not necessary to find an SP to map them. It is only necessary to transmit the object codes to SP2.

In the sequel, task D is required to be mapped, and LEC-DN heuristic chooses SP3. As task D was mapped on an empty SP, the PREMAP method is executed. In this case, the non-mapped task E is the first to be evaluated. Task E is not pre-mapped in SP3 since it communicates with other task (i.e. G) with a higher volume than it communicates with task D. An easy way to verify this situation is to verify the first element of task E

communicating task list. Next, task A is not pre-mapped since it is already mapped. Finally, task F is evaluated. Task F is pre-mapped since: (*i*) it is a non-mapped task; (*ii*) task D is the first element of C(t_F), proving that task F communicates with task D transferring a higher volume than with any other task.



Task	Communication Task List
A	(B,500);(C, 450);(D,100)
B	(A, 500)
C	(A, 450)
D	(E, 200);(A, 100);(F, 50)
E	(G, 300); (D, 200)
F	(D, 50); (H, 25)
G	(E, 300)
H	(F, 25)

Figure 38 - PREMAP method example.

Figure 39 shows the implementation of the PREMAP method.

Input: The SP sp_i, the task t_i mapped onto sp_i
Output: A set of tasks *pre-mapped* onto sp_i
1. $NC(t_i) \leftarrow non\text{-}mapped\_tasks(C(t_i))$ // all non-mapped tasks communicating with t_i
2. $d_i \leftarrow first(NC(t_i))$ //Get the first task in the $NC(t_i)$
3. **WHILE** all tasks in NC(t_i) were not evaluated **OR** $tasks(sp_i)<MAX\_SP\_TASKS$ **DO**
4. $h_i \leftarrow first(C(d_i))$ // Get the first task h_i (with highest communication volume) in $C(d_i)$
5. **IF** $h_i=t_i$ **THEN**
6. $premap(d_i,sp_i)$ // premap d_i onto p_i
7. $tasks(sp_i)$++ // increase the number of mapped/*pre-mapped* tasks onto p_i
8. **END IF**
9. $d_i \leftarrow next(NC(t_i))$ // Get the next task in the $NC(t_i)$
10. **END WHILE**

Figure 39 – PREMAP method algorithm pseudo-code [MAN11b].

The PREMAP method begins by assigning to the set NC(t_i) the non-mapped tasks of C(t_i) (line 2). The next step evaluates each task d_i from NC(t_i), to choose the tasks to be pre-mapped onto sp_i. This evaluation (line 3-10) is executed while sp_i has less than MAX_SP_TASKS mapped/pre-mapped tasks onto it, or if all possible tasks in NC(t_i) were already evaluated. For each task d_i, the first task h_i with the highest communication volume

in its C(d_i) is obtained (line 4). Then, task h_i is compared to the task t_i (line 5). This comparison verifies if d_i communicates with a higher volume with t_i than it communicates with any other tasks. If true, t_i is pre-mapped onto sp_i, also increasing the sp_i number of mapped/pre-mapped tasks (lines 6-7). This process continues if other tasks are available in NC(t_i).

5.3 LOAD (L)

The goal of the Load (L) heuristic is to distribute the workload evenly, improving in long-term the system reliability. To achieve this goal, this heuristic assigns tasks to the less overloaded processors. The mapping protocol steps are presented in Section 5.3.1, which describes the cluster selection; and Section 5.3.2, which discusses the initial and non-initial tasks mapping.

5.3.1 Cluster selection

This heuristic computes the *cl_energy(c_k)* (see definition 7) energy value for each cluster. Then, the cluster with the smallest *cl_energy(ck)* is selected. This procedure avoids mapping an application in a high overloaded cluster, which improves the workload distribution. Figure 40 presents the pseudo-code of the cluster selection heuristic.

```
Input: application size APP.size
Output: selected_cluster
1.  selected_cluster ← -1
2.  selected_cluster_energy ← +∞
3.  //Verify if the system has available resources to map the application
4.  IF available_resources(system) >= APP.size THEN
5.      FOR EACH cluster c_k in the system
6,          IF available_resources(c_k) >= APP.size AND cl_energy (c_k)< selected_cluster_energy THEN
7.              selected_cluster ← c_k
8.              selected_cluster_energy ← cl_energy (c_k)
9.          END IF
10.     END FOR
11.  // There is no cluster with enough resources to receive the application
12.  IF selected_cluster = -1 THEN
13.     FOR EACH cluster c_k in the system
14.         IF cl_energy (c_k)< selected_cluster_energy THEN
15.             selected_cluster ← c_k
16.             selected_cluster_energy ← cl_energy (c_k)
17.         END IF
18.     END FOR
19.  END IF
20. END IF
21. return selected_cluster
```

Figure 40 - Cluster selection heuristic used in Load and Load-Communication heuristics.

The heuristic in Figure 40 first verifies if the system has available resources to map the application (line 4). If there are no sufficient resources in the system, the application is scheduled to be mapped later. The first loop (lines 5-10) analyzes all clusters that have available resources to map the application, selecting the one with the smallest accumulated energy. If there are no clusters with available resources to map the application, a cluster with the smallest accumulated energy is selected, regardless the number of available resources (lines 12-19). Note that the application is mapped in the MPSoC iff the system has available resources for the application. This heuristic aims to distribute the energy homogeneously when a new application arrives in the system. In the long-term, this procedure avoids hotspots, and processors stressed over the time. Consequently, this heuristic contributes to minimizing aging effects, as wearout.

5.3.2 Initial and non-initial tasks mapping

The L heuristic uses the same procedure to map both initial and non-initial tasks. All SPs inside the cluster are evaluated, and the one with the lowest TE is selected. Figure 41 shows the pseudo-code of the initial and non-initial tasks mapping heuristic. The loop between lines 3 and 8 evaluates all SPs inside the cluster, selecting the one with the lowest accumulated energy TE. In this context, this heuristic tries to balance system workload, assigning tasks to the less overloaded SPs.

```
Input: selected_cluster
Output: selected_sp
1. selected_sp ← -1
2. selected_sp_energy ← +∞
3.   FOR EACH SP sp_i IN selected_cluster
4.     IF available(sp_i) = true AND TE(sp_i)< selected_sp_energy THEN
5.         selected_sp ← sp_i
6.         selected_sp_energy ← TE (sp_i)
7.     END IF
8.   END FOR
9. return selected_sp
```

Figure 41 - Initial and non-initial tasks mapping used in Load heuristic.

5.4 LOAD-COMMUNICATION (LC)

The Load-Communication heuristic mixes LEC-DN and L heuristics, making a trade-off between communication volume reduction, and workload distribution.

5.4.1 Cluster selection

This heuristic uses the same cluster selection approach of the Load heuristic, presented in Section 5.3.1.

5.4.2 Initial tasks mapping

This heuristic divides the initial task mapping process into two phases. The first phase selects an SP with the smallest *region_energy* to receive an initial task. A second phase is executed if there is more than one initial task. In such phase, a set with all SPs up to *n* hops far from the selected SP is created, selecting the SP of this set with the smallest TE.

The function *region_energy*(sp_i, *n_hops*) returns the average TE from the set containing sp_i and all SPs up to *n_hops* hops far from sp_i. Figure 42 shows a hypothetical example using a 7x7 cluster, where sp_i is the central SP $sp_{central}$ (in green); and *n_hops* is 3 hops. In Figure 42, the numbers inside each rectangle represent the TE of each SP. The value of *region_energy*($sp_{central}$, *3*) corresponds to 64, since: (*i*) inside a region 3 hops far from $sp_{central}$ there is 25 SPs; (*ii*) the sum of the TEs of the SPS in this area is equal to 4100; (*iii*) the average TE in this area is equal to 4100/25=64.

Suppose a hypothetical example of an application with two initial tasks: t_i and t_j. The first initial task t_i is mapped in $sp_{central}$ of Figure 42. For the mapping of the t_j a region 3 hops far from $sp_{central}$ is defined, as delimited by the colored SPs in Figure 42. Then, the SP with the smallest TE in this region is selected to map t_j. In the example, such SP has TE equal to 66.

			123			
		66	178	280		
	114	200	80	109	77	
120	210	120	200	110	350	327
	124	156	85	413	95	
		149	123	189		
			102			

Figure 42 - Hypothetical example of *region_energy*.

The pseudo-code of the first phase of the initial tasks mapping heuristic is detailed in Figure 43. The main loop (lines 3-8) selects an SP (*selected_sp*) with the lowest *region_energy*. This procedure ensures that application's tasks that will be mapped later will be assigned closer to the selected SP and in SPs with a lower accumulated energy.

Input: *selected_cluster, n_hops*
Output: *selected_sp*
1. selected_sp ← *-1*
2. selected_region_energy ← *+∞*
3. **FOR EACH** SP sp_i in the *selected_cluster*
4. **IF** available(sp_i) **AND** region_energy(sp_i, n_hops)< selected_region_energy **THEN**
5. selected_sp ← sp_i
6. selected_region_energy ← region_energy(sp_i, n_hops)
7. **END IF**
8. **END FOR EACH**
9. return selected_sp

Figure 43 - First phase of the initial tasks mapping used in Load-Communication heuristic.

If the application has only one initial task, the SP chosen by the heuristic of Figure 43 is selected to execute the task. Otherwise, the heuristic presented in Figure 44 is executed for each non-mapped initial task. In line 4 it is created a set *neighbors_list* with all SPs up to *n_hops* from *selected_sp* computed in the previous phase. The loop between lines 6-11 selects an available SP from the *neighbors_list* with the smallest TE. If there is no available SP inside the list, the search space increases 1 hop (lines 12-15), until visiting all SPs of the cluster (line 5).

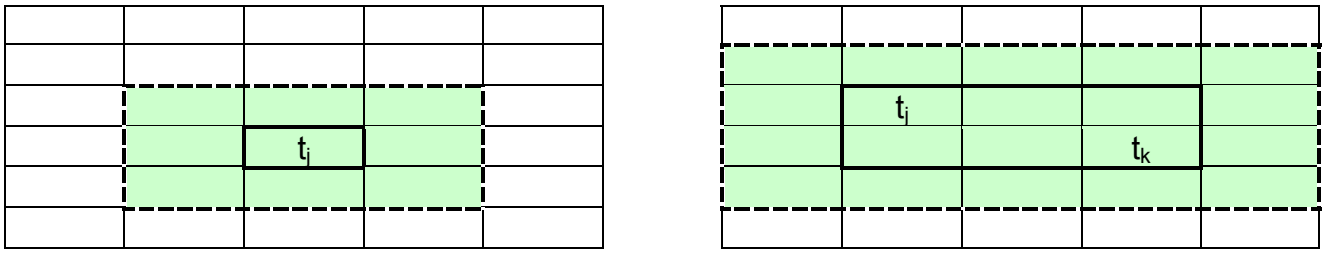
Input: $SP_{address}$, *n_hops* // $SP_{address}$ is the *selected_sp* address obtained in the 1st phase
Output: *selected_sp*
1. selected_sp ← -1
2. selected_sp_energy ← $+\infty$
3. // Get all neighbors of *selected_sp* within a distance *n_hops*
4. neighbors_list ← neighbors($SP_{address}$, n_hops)
5. **WHILE** all SPs in the cluster not evaluated **AND** selected_sp=-1 **DO**
6. **FOR EACH** SP sp_i **IN** neighbors_list
7. **IF** available(sp_i) = true **AND** *TE*(sp_i) < selected_sp_energy **THEN**
8. selected_sp ← sp_i
9. selected_sp_energy ← *TE* (sp_i)
10. **END IF**
11. **END FOR**
12. **IF** selected_sp = -1 **THEN**
13. n_hops ← n_hops +1
14. neighbors_list ← neighbors($SP_{address}$, n_hops)
15. **END IF**
16. **END WHILE**
17. return selected_sp

Figure 44 – Second phase of the initial tasks mapping used in Load-Communication heuristic.

5.4.3 Non-initial task mapping

Suppose a non-initial task t_i is required to be mapped. The LC heuristic evaluates the set C(t_i), and creates a bounding box containing all t_i communicating tasks mapped within the cluster. Then, such bounding box is increased in one hop offering a large search space. Figure 45 illustrates the mapping search space in the cluster. This heuristic selects the SP inside the bounding box with the lowest TE. This heuristic mixes concepts used in LEC-DN and L heuristics, aiming to make a trade-off between workload balancing and communication volume reduction. For this purpose, LC uses a similar bounding box search method used in the LEC-DN heuristic. The difference is that the bounding box is increased by one hop in both cases: when there is one and when there are more than one communicating tasks mapped in the cluster. In both cases, the heuristic selects the SP inside the bounding box with the lowest TE. Such approach is different from the one used in the L heuristic. In L heuristic it is selected the SP with the lowest TE inside the cluster,

which can increase the distance between communicating tasks. The approach used in LC selects the less overloaded SP in a region close to the communicating tasks of the required task, by using a bounding box.



(a) search space when one communicating task is already mapped (t_i)

(b) search space when more than one communicating task is already mapped (t_i and t_j). Solid lines correspond to the original bounding box, dashed lines to the bounding box increased by one hop

Figure 45 - Load-Communication heuristic search space.

Figure 46 describes the algorithm used to select an SP to receive a non-initial task t_i. The heuristic creates a list with all tasks communicating with t_i already mapped onto the SPs of the cluster (line 3). In the sequel, a bounding box rectangle is defined (line 4), with all mapped communicating tasks. This bounding box is increased by one hop (line 5), offering a larger search space to map t_i. A list with candidate SPs is created (line 7). The available SP in the list with the smallest TE is selected (lines 8-13). If no SP can be selected, the bounding box is increased by one hop (lines 14-16). This process continues up to find a SP or visiting all SPs of the cluster.

Input: t_i , set $C(t_i)$
Output: selected_sp
1. selected_sp ← -1
2. selected_sp_energy ← $+\infty$
3. $MC(t_i)$← mapped_tasks($C(t_i)$) // all tasks communicating with t_i already mapped
4. bounding_box ← area($MC(t_i)$)
5. increase(bounding_box, 1)
6. **WHILE** all SPs in the cluster were not evaluated **AND** selected_sp=-1 **DO**
7. neighbors_list ← search_SPs(bounding_box)
8. **FOR EACH** SP sp_i **IN** neighbors_list
9. **IF** available(sp_i) = true **AND** $TE(sp_i)$< selected_sp_energy **THEN**
10. selected_sp ← sp_i
11. selected_sp_energy ← $TE(sp_i)$
12. **END IF**
13. **END FOR**
14. **IF** selected_sp = -1 **THEN**
15. increase(bounding_box, 1)
16. **END IF**
17. **END WHILE**
18. return selected_sp

Figure 46 - Mapping of non-initial tasks.

5.5 TASK MAPPING HEURISTICS EVALUATION

This Section evaluates the four mapping heuristics using the OVP platform model, and the SystemC platform model. Section 5.5.1 presents the experiments concerning the OVP platform. Section 5.5.2 presents experiments using the SystemC platform. Finally, section 5.5.3 discuss the evaluated experiments.

5.5.1 Task mapping evaluation using the OVP platform model

This Section first employs the OVP platform model to compare the mapping heuristics in terms of workload distribution (Section 5.5.1.1), communication volume, (Section 5.5.1.2), and total execution time (Section 5.5.1.3). Section 5.5.1.4 evaluates the heuristics in a larger MPSoC (12x12), to demonstrate the scalability of the approach. For this purpose, three applications are used as benchmarks (all applications are real applications, described in C language): (*i*) DTW - Digital Time Warping (DTW), with ten tasks; (*ii*) MPEG decoder, with five tasks; (*iii*) DJK - Dijkstra, with six tasks.

Table 16 presents the six evaluated scenarios. Such scenarios use a 10x10 MPSoC instance with a 5x5 cluster size with different applications. All applications start at the beginning of the simulation while there are enough resources. When resources become available, a new application starts. Such behavior induces a large system usage, increasing the effort of the heuristics to obtain the best results.

Table 16 – Evaluated scenarios.

Scenario	Applications	Number of Applications	Number of tasks
A	120 x MPEG	120	600
B	100 x DJK	100	600
C	15 x DTW, 35 x MPEG	50	325
D	65 x MPEG, 35 x DJK	100	535
E	10 x DTW, 25 x MPEG, 25 x DJK	60	375
F	15 x DTW, 5 x MPEG, 40 x DJK	60	415

5.5.1.1 Workload Distribution

This Section compares the heuristics concerning the workload distribution in all scenarios. For this purpose, the experiments evaluate the number of instructions and the energy consumed by executing such instructions at the end of the execution. Table 17 presents the number of executed instructions and the consumed energy values by the SPs (total, average and standard deviation). The number of instructions includes the instructions required to execute the applications of each scenario and the number of instructions required by the microkernel.

Table 17 – Instructions (thousands of instructions) and energy (mJ) for the evaluated scenarios, using a 10x10 MPSoC size – OVP platform.

	PREMAP-DN – Instructions			LEC-DN – Instructions			L – Instructions			LC – Instructions		
	Total	Average	Std. Dev.	Total	Average	Std. Dev.	Total	Average	Std. Dev.	Total	Average	Std. Dev.
A	68,413	713	1,022	68,106	709	1,063	72,751	758	236	71,663	746	158
B	654,483	6,818	4,539	652,489	6,797	4,848	659,583	6,871	1,988	657,915	6,853	2,900
C	101,229	1,054	930	102,484	1,068	879	100,361	1,045	278	99,558	1,037	376
D	267,845	2,790	2,609	267,520	2,787	2,616	271,865	2,832	1,096	269,807	2,810	1,569
E	232,794	2,425	2,076	233,086	2,428	2,098	233,324	2,430	1,219	232,361	2,420	1,375
F	219,419	2,286	1,182	219,782	2,289	1,189	211,190	2,200	494	207,305	2,159	601
	PREMAP-DN – Energy (mJ)			LEC-DN – Energy (mJ)			L – Energy (mJ)			LC – Energy (mJ)		
A	2,056	21	30	2,047	21	32	2,187	23	7	2,155	22	5
B	21,635	225	154	21,636	225	164	21,800	227	69	21,740	226	100
C	3,070	32	28	3,110	32	27	3,042	32	9	3,015	31	12
D	8,735	91	87	8,725	91	87	8,866	92	38	8,798	92	54
E	7,506	78	69	7,515	78	70	7,523	78	42	7,491	78	47
F	6,669	69	37	6,681	70	37	6,412	67	17	6,283	65	20

Results show that, for a given scenario, the total number of executed instructions and the consumed energy has small variations for the different heuristics (up to 3.6% for instructions and energy). This is an expected result since the workload is the same. The number of instructions and the consumed energy per scenario are directly related to task mapping. Two main factors induce a different number of executed instructions per scenario. The first one is related to the CPU sharing. When tasks are mapped to the same processor, the number of instructions varies due to task scheduling, which comprise context saving and defining the next task to be scheduled. The second one is related to the inter-task communication. When communicating tasks are mapped to a same processor, the communication between such tasks does not occur through the NoC. A communication through the NoC requires creating and treating packets and programming the DMA module, which increases the number of executed instructions.

The most relevant result in Table 17 is the standard deviation value. A small standard deviation value represents a better load distribution among the SPs. Table 18 presents the energy standard deviation values normalized w.r.t. the L heuristic. The evaluated results are normalized to the L heuristic since it presents the small values in most scenarios. This result is explained since the L heuristic has a larger mapping search space. The L heuristic is exhaustive, evaluating all SPs of the cluster.

The L and LC heuristics reduces the standard deviation in all scenarios when compared to PREMAP-DN and LEC-DN heuristic. The PREMAP-DN and LEC-DN heuristics increases up to 4.29 times the energy standard deviation values w.r.t. the L heuristic. The reason explaining this result is the fact that LEC-DN and PREMAP-DN heuristics do not take into account the tasks' load (i.e., energy per task) in the mapping

decision. In addition, the LEC-DN and PREMAP-DN heuristics tend to group tasks together in the same SP, increasing the chance of overloaded SPs.

The LC, a non-exhaustive heuristic, also increases the energy standard deviation values normalized w.r.t. the L heuristic. The LC heuristic has an average increase of 29% compared to L heuristic in all scenarios except in scenario A. LC heuristic reduces the standard deviation in 44% compared to L in scenario A.

Table 18 – Energy standard deviation values normalized w.r.t. the L heuristic.

	L	PREMAP-DN	LEC-DN	LC
A	1.00	4.14	4.29	0.66
B	1.00	2.22	2.37	1.45
C	1.00	3.06	2.89	1.29
D	1.00	2.27	2.27	1.41
E	1.00	1.63	1.65	1.12
F	1.00	2.13	2.14	1.15

Figure 47 details how the standard deviation values of Table 18 reflect the workload distribution. Figure 47 presents the energy consumed by each SP in scenario A for the four heuristics. Red squares represent an energy value above 50 mJ, and green squares represent an energy below 5 mJ. Figure 47(a) and Figure 47 (b) show the energy distribution, respectively, for the PREMAP-DN and LEC-DN heuristics. Such heuristics produce an unbalanced distribution of the energy among SPs, creating energy hotspots, i.e., some SPs consume a high amount of energy while others consume a low amount of energy. PREMAP-DN presents 19 SPs consuming more than 50 mJ; and 53 consuming less than 5 mJ, where 48 SPs were not used to execute tasks. LEC-DN presents 18 SPs consuming more than 50 mJ; and 55 consuming less than 5 mJ, and 43 SPs were not used. This behavior shows that a high number of tasks share few SPs.

Figure 47(c) and Figure 47(d) present a uniform load distribution for the L and LC heuristics. Using the L heuristic, only two SPs consume more than 50 mJ, and no one consumes less than 5 mJ. Using LC heuristics, there are no SPs consuming more than 50 mJ, neither SPs consuming less than 5mJ. In both heuristics, all SPs are used to execute tasks. Such different behavior highlights the benefits of using heuristics that consider the load (i.e. energy) in the mapping process to avoid hotspots, therefore inducing a better system reliability. Further, Figure 47 illustrates the results obtained in Table 18 where L and LC have a reduced standard deviation value compared to PREMAP-DN and LEC-DN heuristics.

Results showing the energy consumed at each SP for the other scenarios are presented in Appendix A.1.

0.48	12.79	1.39	0.48	0.48	0.48	0.48	0.48	0.48	0.48
0.48	72.15	50.25	3.99	6.66	0.48	32.77	41.11	0.93	4.86
0.48	86.58	51.98	94.02	41.53	0.48	73.00	62.82	75.85	26.22
0.48	0.48	27.82	31.82	18.21	0.48	0.48	28.41	23.04	0.48
LMP	0.48	0.48	0.48	0.48	LMP	0.48	0.48	0.48	0.48
0.48	0.48	0.48	0.48	0.48	0.48	7.36	0.48	0.48	0.48
0.48	81.38	30.33	32.16	7.69	0.48	66.83	47.79	7.41	3.05
14.22	114.97	105.01	56.50	16.70	0.48	64.82	72.25	89.78	34.46
0.48	19.75	81.76	91.74	38.86	0.48	13.98	54.43	11.59	0.48
GMP	0.48	0.48	0.48	0.48	LMP	0.48	0.48	0.48	0.48

(a) PREMAP-DN

0.48	3.62	0.93	0.48	0.93	0.48	0.48	3.63	1.38	1.35
0.48	54.74	38.81	9.84	11.05	0.48	70.65	26.57	23.26	15.84
0.48	51.16	104.36	76.42	25.67	0.48	53.00	47.63	118.17	36.40
0.48	0.48	56.07	23.80	6.81	0.48	0.48	14.75	3.53	0.48
LMP	0.48	0.48	0.48	0.48	LMP	0.48	0.48	0.48	0.48
0.48	3.34	0.48	0.48	0.48	0.48	0.48	0.93	0.48	3.04
0.48	85.11	65.64	27.82	8.47	0.48	73.30	36.83	16.59	24.00
0.48	119.72	86.28	49.30	29.27	0.48	58.51	44.73	116.83	33.93
0.48	3.08	82.93	87.56	53.81	0.48	0.48	27.79	6.58	0.48
GMP	0.48	0.48	0.48	0.94	GMP	0.48	0.48	0.48	0.48

(b) LEC-DN

18.38	32.57	15.84	28.57	20.96	21.93	23.23	32.39	41.11	22.51
20.71	19.31	16.51	21.73	21.63	20.31	16.52	16.54	20.00	31.40
50.05	20.84	21.74	15.54	19.63	22.69	26.40	28.57	19.23	36.19
54.13	18.69	22.20	14.84	25.11	24.70	27.93	23.06	23.30	21.58
LMP	15.72	19.16	15.08	15.30	LMP	18.75	22.62	21.95	16.76
19.81	19.61	22.81	17.77	15.54	23.80	16.17	23.52	33.09	24.64
16.75	18.37	28.36	19.47	15.53	25.40	18.86	25.24	15.95	18.68
18.67	17.37	36.03	15.23	18.78	25.53	23.41	14.93	19.24	23.30
26.54	23.17	21.81	17.07	42.30	24.71	21.08	17.22	33.04	43.26
GMP	20.59	22.06	15.91	19.94	LMP	24.50	18.86	18.93	20.40

(c) L

26.14	24.84	29.60	23.91	25.75	18.18	32.23	19.71	34.39	30.41
21.07	18.29	19.92	22.82	18.49	21.93	21.96	17.87	23.14	28.28
20.66	18.94	19.08	19.58	30.07	15.57	24.93	20.32	21.68	23.02
25.59	21.99	24.54	23.38	21.78	18.01	24.25	27.99	19.60	19.67
LMP	23.22	23.74	17.33	22.37	LMP	23.21	16.37	35.22	23.20
20.38	22.61	16.52	17.50	24.15	23.52	23.25	19.03	18.67	27.99
17.25	19.61	32.48	29.17	19.80	18.95	20.84	16.47	27.86	27.91
11.73	30.93	13.98	17.66	22.15	25.89	19.03	36.99	25.47	15.17
19.05	26.24	17.63	27.53	15.36	17.34	23.84	16.42	24.36	21.38
GMP	19.99	17.75	16.72	26.52	LMP	23.29	21.31	28.06	24.84

(d) LC

Figure 47 – Energy consumed per SP (mJ) for each heuristic in Scenario A with the OVP platform. Each rectangle represents a PE. Green rectangles represent SPs consuming less than 5 mJ. Red rectangles represent SPs consuming more than 50 mJ.

5.5.1.2 Communication Volume

This Section compares the heuristics considering the total communication volume transferred through the NoC. Table 19 shows the total communication volume in thousands of flits for each scenario.

Table 19 – Total communication volume (thousands of flits).

	PREMAP-DN	LEC-DN	L	LC
A	159	180	888	390
B	910	943	2,420	965
C	410	387	1,141	661
D	344	340	1,357	507
E	425	441	1,425	718
F	1,010	1,008	2,629	1,395

PREMAP-DN and LEC-DN heuristics have the largest reduction in communication volume compared to others heuristics. Such result was expected since such heuristics consider the communication volume in the mapping decision. Table 20 shows the results normalized w.r.t the PREMAP-DN heuristic since it presents the highest reduction in communication volume in the most scenarios.

Table 20 - Total communication volume normalized w.r.t. the PREMAP-DN heuristic

	PREMAP-DN	LEC-DN	L	LC
A	1.00	1.14	5.60	2.46
B	1.00	1.04	2.66	1.06
C	1.00	0.94	2.78	1.61
D	1.00	0.99	3.94	1.47
E	1.00	1.04	3.35	1.69
F	1.00	1.00	2.60	1.38

L and LC heuristics increase the communication volume in average 3.27 and 1.97 times, respectively, compared to PREMAP-DN. Such result is explained since the main cost function of L and LC heuristics is to distribute the tasks onto the SPs, which induces a large use of the NoC. Otherwise, the PREMAP-DN heuristic tends to map tasks together in the same SP. In this case, there is an intra-SP communication between tasks and tasks do not communicate using the NoC.

The LC heuristic reduces the total communication volume in average 52.89% compared to the L heuristic. Such result is explained since the LC heuristic considers the distance of the communicating tasks in the mapping decision. On the other hand, the L heuristic spreads tasks all over the cluster.

5.5.1.3 Total Execution Time

In this Section, heuristics are evaluated concerning the total execution time. Three main factors influence the total execution time for different mapping solutions: CPU sharing; NoC traffic and congestion; and mapping algorithm computation. Table 21 presents the execution time for each scenario.

Table 21 – Total execution time (thousands of clock cycles).

	PREMAP-DN	LEC-DN	L	LC
A	7,645	7,842	6,886	7,576
B	28,993	36,190	19,614	25,207
C	5,538	5,771	4,566	5,246
D	16,083	14,859	13,117	13,942
E	11,297	13,044	10,212	10,728
F	8,276	8,288	7,147	7,848

The L heuristic presents the lowest execution time among the heuristics. Table 22 presents the total execution time normalized w.r.t. such heuristic.

Table 22 – Total execution time normalized w.r.t. the L heuristic.

	L	PREMAP-DN	LEC-DN	LC
A	1.00	1.11	1.14	1.10
B	1.00	1.48	1.85	1.29
C	1.00	1.21	1.26	1.15
D	1.00	1.23	1.13	1.06
E	1.00	1.11	1.28	1.05
F	1.00	1.16	1.16	1.10

Results show that the PREMAP-DN, LEC-DN and LC heuristics increase the total execution time, respectively, by 22%, 30%, and 12% (average values) when compared to the L heuristic. This result comes from the better workload distribution obtained in L heuristic, which reduces the number of tasks sharing the same CPU. Distributing tasks and using as much as possible SPs of the system at the same time may reduce the execution time.

5.5.1.4 Evaluation in a 12x12 MPSoC

The same scenarios of Table 16 were executed in a 12x12 MPSoC instance, with 6x6 clusters. The goal of this experiment is to evaluate the scalability of the approach.

Table 26 presents the number of executed instructions and the consumed energy values by the SPs (total, average and standard deviation). Similar results were obtained, with a similar number of executed instructions and consumed energy per scenario, and a significant reduction in the standard deviation values for L and LC heuristics.

Table 23 – Instructions (thousands of instructions) and energy (mJ) for the evaluated scenarios, using a 12x12 MPSoC size – OVP platform.

	PREMAP-DN – Instructions			LEC-DN – Instructions			L – Instructions			LC – Instructions		
	Total	Average	Std. Dev.	Total	Average	Std. Dev.	Total	Average	Std. Dev.	Total	Average	Std. Dev.
A	69,090	494	846	69,512	497	823	74,324	531	182	72,862	520	140
B	655,452	4,682	3,518	655,615	4,683	3,307	660,185	4,716	1,640	658,970	4,707	1,932
C	102,543	732	978	103,527	739	910	102,040	729	314	99,049	707	349
D	267,826	1,913	2,040	268,020	1,914	1,850	272,542	1,947	914	270,583	1,933	1,404
E	233,907	1,671	1,613	234,202	1,673	1,592	233,672	1,669	961	231,716	1,655	1,132
F	219,553	1,568	1,468	218,606	1,561	1,385	214,339	1,531	463	208,451	1,489	625
	PREMAP-DN – Energy (mJ)			**LEC-DN – Energy (mJ)**			**L – Energy (mJ)**			**LC – Energy (mJ)**		
A	2,081	15	25	2,094	15	25	2,239	16	6	2,194	16	4
B	21,667	155	120	21,671	155	112	21,825	156	57	21,781	156	67
C	3,115	22	30	3,146	22	28	3,101	22	10	3,005	21	11
D	8,739	62	68	8,744	62	61	8,891	64	32	8,828	63	48
E	7,544	54	53	7,553	54	53	7,537	54	33	7,475	53	39
F	6,677	48	45	6,647	47	43	6,526	47	16	6,332	45	20

Table 24 compares the heuristics considering the total communication volume transferred through the NoC. As observed previously, PREMAP-DN and LEC-DN have the largest reduction in the communication volume, and L presents the worst results. The LC heuristic, which distributes evenly the workload, has an important reduction in the communication volume compared to L heuristic.

Table 24 – Total communication volume (thousands of flits), using a 12x12 MPSoC size.

	PREMAP-DN	LEC-DN	L	LC
A	139	144	1,096	395
B	916	881	2,640	1,010
C	345	344	1,385	683
D	315	315	1,568	542
E	467	419	1,581	726
F	882	942	3,088	1,494

Table 25 presents the total execution time. In most scenarios, L and LC heuristics reduce the total execution time compared to PREMAP-DN and LEC-DN heuristics.

Table 25 – Total execution time (thousands of clock cycles), using a 12x12 MPSoC size.

	PREMAP-DN	LEC-DN	L	LC
A	9,772	9,829	8,593	9,200
B	26,317	24,715	17,451	20,768
C	6,657	6,648	5,580	6,074
D	13,853	13,064	10,389	13,365
E	10,460	10,823	8,474	9,617
F	7,913	8,148	7,222	7,895

5.5.2 Task mapping evaluation using the SystemC platform model

In this Section, the scenarios described in Table 16 (10x10 MPSoC) are evaluated using the SystemC cycle-accurate platform. The goal of executing the experiments in the SystemC platforms is to obtain accurate results for execution time, and power. The four heuristics are compared according to different performance figures: (*i*) workload distribution, in Section 5.5.2.1; (*ii*) communication volume, in section 5.5.2.2; (*iii*) total execution time, in Section 5.5.2.3; (*iv*) temperature distribution, in Section 5.5.2.4; (*v*) and power traces, in Section 5.5.2.5.

5.5.2.1 Workload Distribution

This Section compares the heuristics in terms of the instructions and energy distribution. Results evaluate the number of instructions executed and the energy consumed by each SP at the end of execution. Table 26 presents the number of executed instructions and the consumed energy values (total, average and standard deviation), for the heuristics.

Table 26 – Instructions (thousands of instructions) and energy (mJ) for the evaluated scenarios – SystemC platform.

	PREMAP-DN – Instructions			**LEC-DN – Instructions**			**L – Instructions**			**LC – Instructions**		
	Total	Average	Std. Dev.	Total	Average	Std. Dev.	Total	Average	Std. Dev.	Total	Average	Std. Dev.
A	43.584	454	842	43.646	455	847	45.534	474	121	45.249	471	98
B	510.356	5.316	1.620	510.448	5.317	1.619	513.293	5.347	1.826	511.970	5.333	1.364
C	68.240	711	749	68.481	713	694	69.069	719	154	69.019	719	275
D	202.672	2.111	1.465	202.222	2.106	1.497	203.450	2.119	889	203.429	2.119	942
E	173.412	1.806	1.217	173.137	1.804	1.151	174.075	1.813	822	173.574	1.808	894
F	148.298	1.545	534	147.949	1.541	521	148.796	1.550	152	148.951	1.552	246
	PREMAP-DN – Energy (mJ)			**LEC-DN – Energy (mJ)**			**L – Energy (mJ)**			**LC – Energy (mJ)**		
A	1.238	13	24	1.240	13	24	1.299	14	4	1.289	13	3
B	15.010	156	48	15.013	156	48	15.114	157	54	15.073	157	41
C	1.987	21	22	1.994	21	20	2.013	21	5	2.010	21	8
D	5.939	62	43	5.926	62	44	5.968	62	27	5.967	62	28
E	5.087	53	36	5.080	53	34	5.111	53	24	5.095	53	27
F	4.340	45	16	4.331	45	15	4.358	45	5	4.361	45	7

The number of executed instructions and energy consumed per scenario is similar, with a small variation due to the different mappings (up to 2.57%). A similar result was obtained with the OVP platform (up to 3.6%).

The gray columns highlight the most relevant results of the Table 26, showing the standard deviation of the average number of instructions/energy consumption of each SP. As explained before, a smaller standard deviation reflects in a better workload balancing. The L heuristic reduces the standard deviation in most scenarios, as pointed out by the

results obtained the in OVP platform. Table 27 compares the heuristics' energy standard deviation normalized w.r.t. the L heuristic.

When compared to the L heuristic, the PREMAP-DN and LEC-DN increase the standard deviation, respectively, 3.62 and 3.53 times, except in Scenario B. In Scenario B, PREMAP-DN and LEC-DN heuristics reduce the standard deviation by 12% compared to the L. This result can be explained since the scenario B executes only the Djikstra application, which tasks have similar load values. The LC heuristic has an average increase of 38% compared to L, excepting scenario A and B. LC heuristic reduces the standard deviation in average 22.5% compared to L in Scenarios A and B.

Table 27 – Energy standard deviation normalized w.r.t. the L heuristic.

	L	PREMAP-DN	LEC-DN	LC
A	1.00	6.81	6.84	0.80
B	1.00	0.88	0.88	0.75
C	1.00	4.75	4.40	1.75
D	1.00	1.62	1.65	1.05
E	1.00	1.47	1.39	1.09
F	1.00	3.44	3.36	1.61

To understand the standard deviation reduction values of Table 26, Figure 48 presents the energy consumed per SP for scenario A. Note in Figure 48(a) and Figure 48(b) that the PREMAP-DN and LEC-DN heuristic produce an unbalanced load distribution, with several underused processors. PREMAP-DN produces 12 SPs consuming more than 50 mJ (red rectangles), and 65 SPs consumes less than 5 mJ (green rectangles). LEC-DN produces 12 SPs that consumes more than 50 mJ, and 67 consuming less than 5 mJ. Further, PREMAP-DN has 51, and LEC-DN 58 SPs that do not execute tasks.

On the other hand, L and LC heuristic provide a better load distribution, with all SPs executing applications' tasks, and no SPs consumes more than 50 mJ. The highest energy values are, respectively, 92.01mJ, 96.62mJ, 31.17mJ, and 22.73, for PREMAP-DN, LEC-DN, L and LC heuristics. PREMAP-DN and LEC-DN concentrate the load in few SPs, increasing the probability to failures due to wear of process. The L and LC heuristic use the total energy consumed in an SP (TE) to avoid overused cores and reduce the probability of failures due to the wear of process.

Results presenting the energy consumed at each SP for the other scenarios are presented in Appendix A.2. Such results present a similar behavior of Scenario A, where PREMAP-DN and LEC-DN heuristics tend to concentrate the load in few SPs. Note that the used SPs in PREMAP-DN and LEC-DN with higher consumed energy tend to concentrate in the central SPs of a cluster. This result is explained by the initial tasks mapping approach used for such heuristics. The central SP of a cluster is the point that has a reduced distance to all cluster SPs. Therefore, mapping tasks in such central SPs

help in reduce communication distances and the communication volume transferred through the NoC. However, as mentioned before, such approach tends to compromise reliability, increasing the wear of SPs within such central area.

0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
0.02	1.77	41.08	3.56	1.72	0.02	12.49	26.12	3.30	2.63
0.02	59.26	80.13	47.47	1.22	0.02	50.73	83.11	69.31	11.01
0.02	8.45	19.43	10.44	1.72	0.02	0.02	8.48	10.22	1.76
LMP	0.02	0.02	0.02	0.02	LMP	0.02	0.02	0.02	0.02
0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
0.02	45.15	12.99	8.78	2.10	0.02	25.50	5.15	0.63	1.23
0.02	50.97	92.01	52.47	4.19	0.02	67.37	81.94	70.57	17.34
0.02	0.02	78.52	22.44	1.77	0.02	0.02	25.33	13.70	1.76
GMP	0.02	0.02	0.02	0.02	LMP	0.02	0.02	0.02	0.02

(a) PREMAP-DN

0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
0.02	35.98	6.46	0.32	0.02	0.02	26.43	20.86	3.31	2.03
0.02	67.19	96.62	27.93	0.62	0.02	50.98	69.51	49.88	4.51
0.02	0.02	42.09	8.61	0.02	0.02	0.02	42.35	8.71	0.02
LMP	0.02	0.02	0.02	0.02	LMP	0.02	0.02	0.02	0.02
0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
0.02	37.39	13.07	0.02	1.22	0.02	14.64	1.74	0.02	1.54
0.02	50.68	95.74	76.62	17.51	0.02	59.01	68.82	74.74	18.60
0.02	0.02	60.93	10.87	0.02	0.02	0.02	50.55	18.17	2.99
GMP	0.02	0.02	0.02	0.02	LMP	0.02	0.02	0.02	0.02

(c) LEC-DN

11.50	14.30	10.05	9.66	15.69	14.12	16.82	15.10	9.68	11.19
11.55	12.63	14.73	10.54	12.67	9.86	23.37	11.61	9.59	13.32
10.03	9.73	14.52	13.49	9.98	15.00	11.73	10.30	15.97	13.64
11.08	16.31	17.06	14.08	10.50	11.26	10.87	16.23	17.71	12.03
LMP	10.68	12.03	20.06	31.17	LMP	10.73	13.26	13.86	17.11
10.29	15.03	16.48	12.20	12.38	23.35	10.47	10.95	10.67	9.71
17.13	11.02	13.49	11.30	18.02	10.39	11.81	11.02	13.32	18.55
15.76	9.80	11.34	16.19	11.22	11.77	15.41	12.62	14.53	13.66
20.79	10.91	11.73	11.11	15.37	16.14	11.10	14.21	15.76	10.83
GMP	16.56	11.43	14.70	10.58	LMP	13.34	11.50	17.38	16.82

(c) L

17.99	18.70	9.68	10.37	12.40	18.37	17.89	9.69	10.40	12.40
15.88	16.09	9.68	15.58	17.16	14.67	12.38	11.38	15.64	17.35
13.76	12.29	12.79	8.78	13.49	11.32	12.00	12.80	10.72	13.46
15.12	13.55	10.30	10.25	10.77	16.67	12.07	11.26	9.90	9.43
LMP	11.31	14.37	14.27	16.51	LMP	16.71	14.36	14.25	16.53
15.29	13.27	12.15	15.56	15.22	12.09	17.13	9.67	10.40	12.44
11.96	17.31	10.97	11.02	14.92	14.80	16.15	9.69	15.71	17.27
12.42	18.28	12.40	13.88	9.48	12.90	12.01	12.89	10.56	13.58
15.23	12.29	22.73	13.81	8.26	15.99	13.79	11.58	10.68	10.85
GMP	12.52	14.63	11.32	8.18	LMP	17.58	14.43	14.32	16.85

(d) LC

Figure 48 – Energy consumed per SPs (mJ) for each heuristic in Scenario A in the SystemC platform. Each rectangle represents a PE. Green rectangles represent SPs consuming less than 5 mJ. Red rectangles represent SPs consuming more than 50 mJ.

5.5.2.2 Communication Volume

Table 28 presents the total communication volume transferred through the NoC for all evaluated scenarios. As obtained in the OVP platform results, the PREMAP-DN and LEC-DN heuristics achieve the highest reductions in the communication volume in all scenarios. Table 29 presents results normalized w.r.t. the PREMAP-DN heuristic.

Table 28 - Total communication volume (thousands of flits)

	PREMAP-DN	LEC-DN	L	LC
A	235	237	1,189	706
B	1,004	1,040	3,467	2,029
C	717	732	1,708	1,050
D	625	566	1,887	1,014
E	793	804	1,921	1,183
F	1,898	1,884	3,571	2,342

Table 29 - Total communication volume normalized w.r.t. the PREMAP-DN heuristic

	PREMAP-DN	LEC-DN	L	LC
A	1.00	1.01	5.07	3.01
B	1.00	1.04	3.45	2.02
C	1.00	1.02	2.38	1.46
D	1.00	0.90	3.02	1.62
E	1.00	1.01	2.42	1.49
F	1.00	0.99	1.88	1.23

L and LC heuristics have an average increase of 3.04 and 1.81 times, respectively, when compared to the PREMAP-DN. Such result is similar to the one obtained in the OVP platform, where L and LC heuristics have an average increase of 3.27 and 1.97 times, respectively.

The LC heuristic reduces the total communication volume in average 40% compared to the L heuristic. In the OVP platform, the LC heuristic had the same behavior. However, in such platform the reduction of LC heuristic was 52%.

5.5.2.3 Total Execution Time

A consequence of the better workload distribution is the reduction in the total execution time of the simulated scenarios. Table 30 presents the execution time for the evaluated scenarios. As observed in the OVP platform, the L heuristic has the larger reduction in total execution time compared to other heuristics.

Table 30 – Total execution time (thousands of clock cycles).

	PREMAP-DN	LEC-DN	L	LC
A	5,950	5,814	5,361	5,667
B	13,463	13,896	15,500	17,288
C	4,452	4,449	3,455	4,297
D	10,623	10,241	8,325	9,113
E	9,340	8,825	7,011	7,749
F	4,670	4,545	3,798	4,289

Table 31 presents the total execution time normalized w.r.t. the L heuristic. LC heuristic has an average increase of 12%, when compared to the L heuristic. In OVP platform such increase was the same (i.e. 12%).

Table 31 – Total execution time normalized w.r.t. the L heuristic

	L	PREMAP-DN	LEC-DN	LC
A	1.00	1.11	1.08	1.06
B	1.00	0.87	0.90	1.12
C	1.00	1.29	1.29	1.24
D	1.00	1.28	1.23	1.09
E	1.00	1.33	1.26	1.11
F	1.00	1.23	1.20	1.13

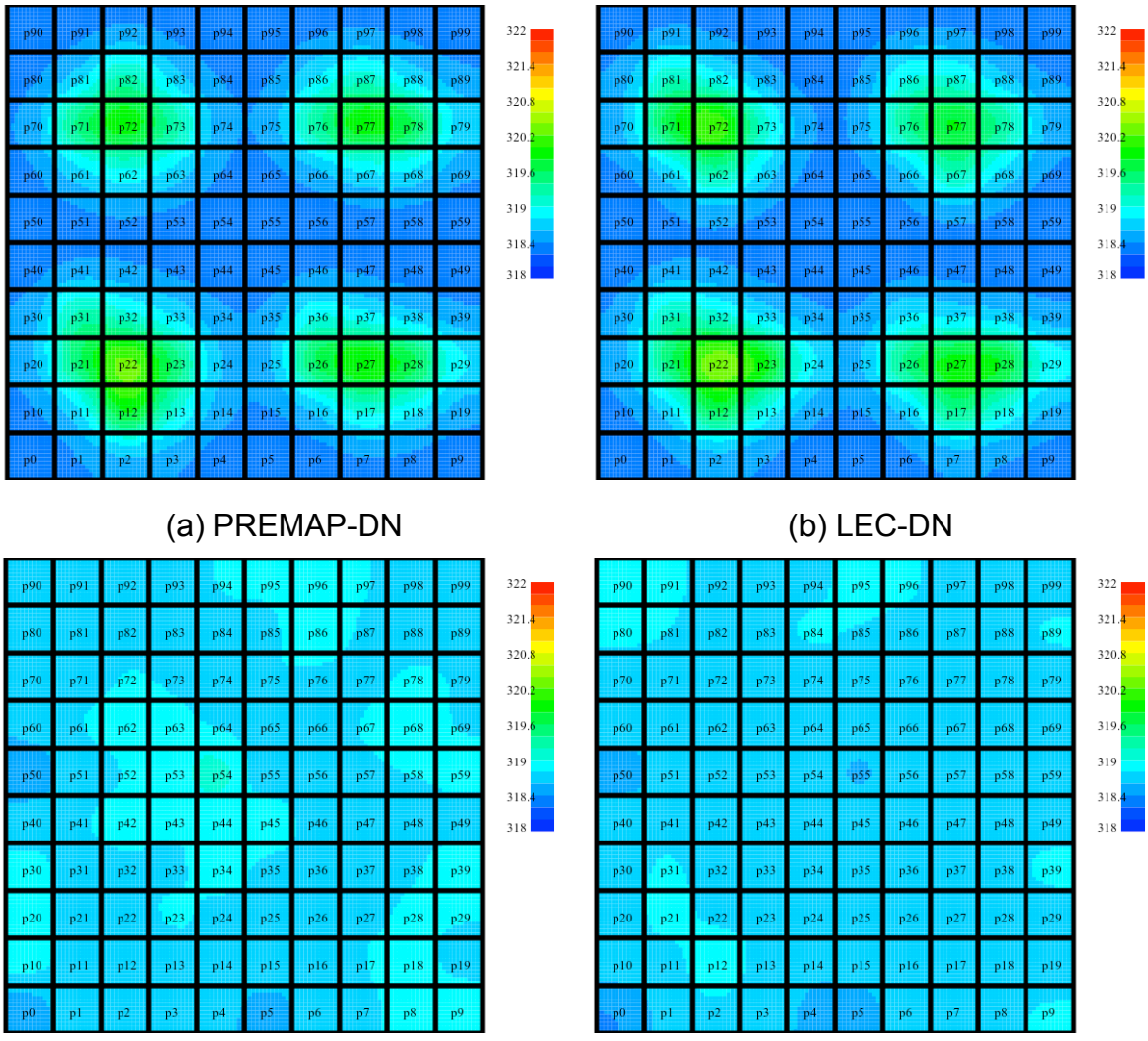
5.5.2.4 Temperature distribution

This Sub-section compares the heuristics regarding the temperature distribution at the end of the simulation. For this purpose, a power report is generated using the proposed energy model (Section 4.1). The HotSpot simulator [WEI06] computes the temperature of the MPSoC at the end of the simulation using the power report (65 nm, 1 GHz, 1.2V, ambient temperature 318.15 K). Figure 49 presents four heat maps showing the temperature distribution of the MPSoC for scenario A. The small temperature gradient (4 K) comes from the simple processor architecture (Plasma) used in the adapted platform, with a peak consumption of 25 mW.

The result verified in the heat maps of Figure 49 was expected according to the results observed in Table 26. The L and LC heuristic generate a uniform temperature distribution because most PEs has a warm temperature (represented as blue rectangles). Such better temperature distribution is related to the reduced standard deviation values compared to PREMAP-DN and LEC-DN heuristics. Moreover, note in Figure 49 that PREMAP-DN and LEC-DN have a high temperature (i.e. green area) in SPs in a central area of the cluster, while other SPs are colder (i.e. blue area). This illustrates an unbalanced temperature distribution with the presence of hotspots.

Results illustrating the temperature distribution for the other scenarios are presented in Appendix B. In some scenarios, the L and LC heuristics achieve a higher temperature

than the LEC-DN and PREMAP-DN heuristics. Such behavior is explained due two reasons. First, L and LC heuristic execute the same workload in a reduced execution time, which reflects in increased power and a higher temperature. Second, since an SP temperature affects the temperatures of neighboring SPs. The L and LC heuristics present a higher number of SPs executing tasks at the same time. Such SPs generate heat and increase the temperature of their neighbors. However, in the most scenarios L and LC evenly distribute the temperature across the system, with a reduced number of hotspots areas compared to PREMAP-DN and LEC-DN heuristics.



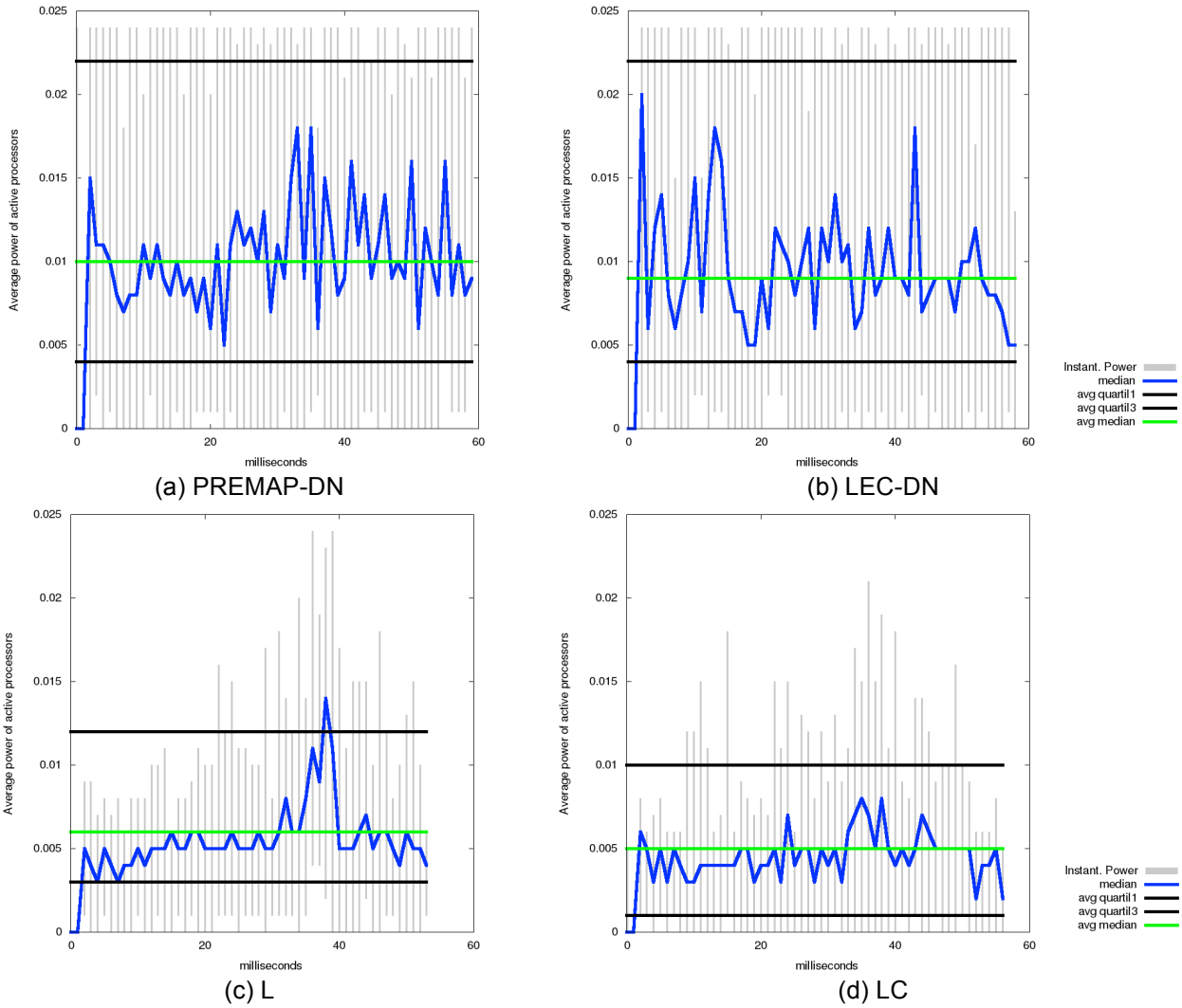
(a) PREMAP-DN (b) LEC-DN

(c) L (d) LC

Figure 49 – Temperature distribution for Scenario A.

5.5.2.5 Power Traces

This Section compares the heuristics regarding the power distribution during the execution time. Figure 50 details the power traces along the execution time for the four heuristics in scenario A. Note that although different, the power traces of Figure 50 correspond to the execution of the same workload but varying the task mapping.



(a) PREMAP-DN (b) LEC-DN

(c) L (d) LC

Figure 50 – Power traces for scenario A. X-axis: time in milliseconds (only PEs executing tasks are considered). Y-axis: average power of active processors (W). Gray bars: 50% of the population, first to third quartiles. Black lines: average first and third quartiles. Green line: average median. Blue line: instantaneous median.

The graphs of Figure 50 show:

- Median: the LEC-DN and PREMAP-DN have a higher median (blue lines) than L and LC heuristic, corresponding to a higher power dissipation of SPs at most sampled periods.
- Quartiles (black lines): the average first and third quartiles are much higher in the PREMAP-DN and LEC-DN heuristic, also corresponding to a higher power dissipation of SPs.
- Instantaneous power dissipation (gray bars): most of the time the gray bars of the PREMAP-DN and LEC-DN heuristic are higher than the L and LC heuristic, corresponding to a worst power distribution.

- L and LC (approximately 54 ms and 57 ms) provides a better execution time reduction compared to PREMAP-DN and LEC-DN (approximately 60 ms and 58 ms) The execution time may be also observed in Table 30.

Results containing the power traces for the other scenarios are presented in Appendix C. When compared to PREMAP-DN and LEC-DN heuristics, the L and LC produce lower power dissipation at most sampled periods. This is explained due to the higher CPU sharing provided by PREMAP-DN and LEC-DN heuristics. When tasks are executing in the same SP, power dissipation increases.

5.5.3 Discussion

The heuristics presented in this Chapter were evaluated regarding different parameters: workload distribution, execution time, communication volume, temperature distribution and power traces.

Results show that taking into account the tasks' load in the mapping decision can considerably improve the workload balancing. However, achieving a perfect workload distribution, where all SPs consume the same energy, may not be achievable because tasks may present different loads. Tasks may have a large difference in their loads, which may generate unbalanced workloads and, possibly, the presence of hotspots. Take, for example, the application illustrated in Figure 51. Figure 51(a) shows the application graph, where the numbers above each vertex represent the task load (energy consumption). Such application is mapped as presented in Figure 51(b). Since the application tasks' loads have a large difference between them, an unbalanced workload is produced (the SP where task C is mapped consumes ten times as much power as other SPs).

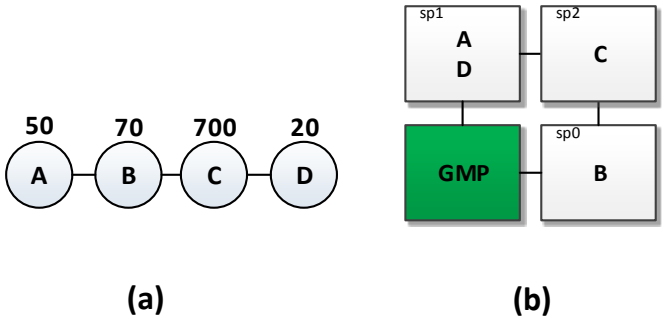


Figure 51 – Example of unbalanced workload.

L and LC heuristics improve workload balancing compared to the LEC-DN and PREMAP-DN heuristics. This result shows that heuristics that focus only on communication reduction produces unbalanced workloads, with an increased occurrence of hotspots, which can compromise system reliability.

Further, PREMAP-DN and LEC-DN heuristics do not take into account processor wear. Even if they can provide a better workload balancing in a specific scenario, such

heuristics tend to concentrate tasks in the same area (i.e. central SPs of a cluster). In this case, system reliability is compromised since SPs of this area are more susceptible to failures due to wear of process.

On contrary, L and LC heuristics can minimize the accumulation of wear on processors by using the TE value. The TE value is the total consumed energy by a given SP, corresponding to the energy (E_i) consumed by all already executed tasks and the tasks that are currently being executed on this processor. Chantem et al. [CHA13] use a similar technique, called LTF (i.e. largest-task first [CHE08]), considering the accumulated energy consumed by a processor in order to slow down wear process. The LTF technique considers a given task has an energy value to execute on a given processor. When an application is required to be mapped, the LTF technique analyzes all tasks and orders them by their energy values. Then, the task with the largest energy value is assigned to the processor with the least total energy consumption. When compared to this Thesis, the main drawback of that technique is that it cannot handle dynamic workloads.

Experiments also show that the L heuristic presented a better (in average 23.52%) workload balancing compared to LC. However, L heuristic increases an average of 40% the communication volume compared to the LC. This result shows that heuristics that focus only on workload balancing increase the communication volume. A high communication volume transferring through the NoC can compromise QoS and induce to links failures. Links failures may produce isolated and, consequently, unusable cores, which compromise the workload balancing.

The L heuristic presented a reduced execution time compared to other heuristics. However, L heuristic may compromise the execution time in scenarios with an increased data transfer between tasks, according to results presented in [MAN15].

The LC heuristic provides the best tradeoff between communication volume reduction and workload balancing, inducing to better system reliability without compromising performance. The LC heuristic reduces the total execution time when compared to heuristics that focus on the communication volume reduction. Using scenarios with higher communication volume transferred through the NoC, the LC heuristic presents a reduced execution time compared to L [MAN15]. Further, as presented next, the LC heuristic also has a low effort to compute mapping decisions, which does not compromise scalability and performance.

The LC heuristic has a small computation complexity (worst case $O(n^2)$). In fact, the execution time to execute the heuristic is small due to its hierarchical implementation. Table 32 presents the scenarios used to evaluate the LC complexity, and Table 33 shows the average number of instructions to execute each step of the mapping.

Table 32 - Evaluated scenarios configuration for the SystemC platform.

Scenario	MPSoC size	Cluster size	Applications	Number of tasks
1	6x6	3x3	8 x DTW, 8 x MPEG, 8 x DJK, 8 x SYN1, 8 x SYN2	360
2			8 x DJK, 11 x SYN1, 11 x SYN2	312
3			16 x MPEG, 8 x DJK, 8 x SYN2	224
4			3 x DTW, 6 x MPEG, 9 x DJK, 5 x SYN1, 8 x SYN2	270
5	12x12	4x4	10 x DTW, 25 x MPEG, 18 x DJK, 14 x SYN1, 12 x SYN2	645

Table 33 – Average number of instructions to execute each step of the Load-Communication heuristic.

Scenario	**Cluster selection**	**Initial tasks mapping**	**Non-initial tasks mapping**
Sc1	280	2497	688
Sc2	288	2337	674
Sc3	299	3057	645
Sc4	282	2588	664
Sc5	562	8584	749

The cluster selection is fast since it only verifies the availability of resources. The most complex step is the mapping of non-initial tasks because it is necessary to evaluate all possible regions inside a cluster. The mapping of the non-initial task requires few instructions because the fixed region restricts the search space. The hierarchical implementation of the mapping heuristic ensures scalability because the cluster size defines the search space, not the MPSoC size. With larger MPSoCs, the cluster selection step increases its execution time (Sc5). With larger clusters, the initial tasks mapping may also increase its execution time as shown in Sc5. In both cases, the execution time to map each non-initial task is negligible compared to the time to execute them (thousands of clock cycles).

Regarding the platforms used to evaluate the scenarios, both platforms generate qualitatively similar results, such as:

- L and LC heuristics improve the workload balancing compared to PREMAP-DN and LEC-DN.
- L heuristic improves the workload balancing compared to LC.
- LC heuristic reduces the communication volume compared to LC.
- L heuristic reduces the execution time compared to the other heuristics. LC heuristic reduces the execution time compared to PREMAP-DN and LEC-DN.

Such results demonstrate the relevance of the OVP platform in the design-space exploration. Even if the OVP model is not accurate, it enables faster simulations (two orders of magnitude) compared to SystemC model, enabling a fast analysis of the proposed heuristics.

5.6 Final Remarks

This Chapter presented one of the main contributions of this Thesis: the LC heuristic. Such heuristic present the following main features:

- Dynamic, i.e., tasks are mapped at runtime;
- Distributed, contributing to increasing performance and reliability for large-scale MPSoCs;
- Scalable, due to its hierarchical implementation;
- Efficient workload distribution, contributing to increasing the system reliability;
- Important reduction of hotspots in the system compared to heuristics aiming to improve only the NoC usage.

The LC heuristic showed that the mapping heuristics must evolve in the sense that the NoC usage is not the most relevant performance figure to consider. Workload in the processing elements, consumed energy, hotspots, reliability, and lifetime are also important parameters to take into account in the mapping decision.

6. CONCLUSION AND FUTURE WORKS

The Introduction of the Thesis stated the following hypothesis: "*This Thesis relies on two hypotheses: (i) untimed abstract models can be used to evaluate task mapping solutions; and (ii) task mapping solutions that focus on communication volume reduction can compromise system reliability; and task mapping solutions that focus on workload balancing, can increase system communication volume"*.

The simulation of the untimed OVP abstract model, calibrated from the low-level model (RTL), enabled to capture the execution time, the energy consumption, and the communication volume. Such abstract model allowed to evaluate mapping heuristics in large MPSoCs, with similar results to the clock-cycle accurate SystemC platform. Consequently, the first part of the above hypothesis is proven.

The extensive evaluation of the heuristics showed that optimizing the NoC communication lead to overused processors, and consequently temperature hotspots, and optimizing the workload distribution lead to overused NoC links. Otherwise, the L heuristic, which focuses only on workload balancing, increased the communication volume considerably when compared to the other heuristics. The proposed LC heuristic makes a tradeoff between load (energy) and NoC communication, which leads in long-term in higher reliability and system lifetime. Thus, the second part of the above hypothesis is also proven.

Appendix D presents the Author's publications related to the present work.

6.1 Conclusions Related to the Multi-level Platform Framework

This Thesis presented a multi-level framework that includes different modeling and debugging capabilities that improve the design space exploration of large MPSoCs. The proposed framework uses a synthesizable RTL VHDL model as reference, which allows to captures accurate area, frequency and power performance figures. Then, two other models were implemented.

The SystemC RTL model is effectively clock-cycle accurate with a simulation speedup of two orders of magnitude compared to the reference model. The simulation of this model produces accurate performance results, and higher debuggability compared to the reference model.

An OVP model is proposed to enable software development and validation at early design stages, reducing design cost and improving time-to-market. Such model provides fast simulation (approximately 20 times faster than the SystemC model in a 20x20 MPSoC instance) and higher software debuggability (i.e. integration with GDB and Eclipse, watchdogs for capturing fetched instruction, etc). Moreover, such model can provide

performance figures (some estimated) including: (*i*) communication volume; (*ii*) number of executed instructions and energy consumed by processors; (*iii*) execution time. The provided performance figures help designers for decision making, decreasing the risks of the development software flow. For instance, the OVP model proved to be able to evaluate task mapping heuristics, generating qualitatively results similar to the SystemC model.

Further, the OVP simulator provides pre-defined models including different processors models (ARM, MIPS, PowerPC, etc.). In this context, one important feature of the OVP modeling is the easiness of integrating different CPU models, allowing the fast development of heterogeneous MPSoCs.

6.2 Conclusions Related to the Distributed System Management

This Thesis proposed a distributed system management approach that divided the system into clusters controlled by a local manager. Such approach proved to provide system scalability and gains in execution time compared to a centralized approach. Further, the proposal reduces mapping computational effort and NoC traffic, improving system reliability.

6.3 Conclusions Related to the Task Mapping Heuristics

This Thesis proposed four runtime distributed heuristics, which were evaluated using the OVP and SystemC platform models. The proposed heuristics are LEC-DN, PREMAP-DN, L, and LC. Among these heuristics, the LC is the one that best tackles the following challenges: (*i*) scalability; (*ii*) dynamic workload; and (*iii*) reliability.

Scalability comes from to the hierarchical mapping approach, which divides the process into three steps: cluster selection, initial task mapping, and non-initial tasks mapping. One processor centralizes the cluster selection, but it corresponds to a fast procedure to select a region to receive the application. The initial and non-initial task mapping are distributed in manager processors, enabling to execute in parallel several mapping decisions.

Support to dynamic workload also comes from the mapping approach, which does not use pre-computed mapping templates. Applications are mapped into the available resources, according to some criteria (e.g. accumulated energy).

The increased reliability is the result of the tradeoff between the use of the NoC and processing resources. The proposed LC heuristic distributes the workload evenly, avoiding hotspots and power peaks, as demonstrated in the temperature maps and power traces.

As mentioned before, this Thesis focuses on general-purpose MPSoC platforms, able to execute several applications that are unknown in advance. This Thesis also assumes that underlying applications can be inserted into the system in a non-

deterministic way, according to user requirements. As presented in Section 2.2, the literature presents different task mapping solutions focusing on improving system reliability for general-purpose MPSoC platforms. However, such solutions present results only for small systems (up to 10 cores). Those solutions have the following drawbacks for large scale MPSoCs:

- Centralized mapping approach. Centralized mapping approaches can compromise performance and reliability for large MPSoCs. This is explained since a single core is responsible to execute the mapping functions, creating a communication and computation system hotspot. **The LC heuristic uses a distributed mapping approach that improves performance and reliability compared to a centralized approach, as presented in Chapter 4.**
- The mapping algorithm of such techniques may increase in complexity in large MPSoCs, imposing high time-consuming and high computational algorithms. High time-consuming algorithms may compromise system performance. In turn, high computational algorithms can compromise reliability since they require higher power consumption to be executed, generating a higher temperature. For instance, different mapping algorithms compute the system gradient temperature, which grows in complexity for large systems. **The LC heuristic has small computation complexity for a 12x12 MPSoC, as presented in Section 5.5.3.**
- Some solutions do not consider the system communication volume in the mapping decision, which can compromise reliability and performance. In large systems, communicating tasks may be mapped too far away from each other, increasing the system communication volume. **The LC considers the communication volume in the mapping decision.**
- Such solutions use wear and temperature sensors, which impose a higher cost for large MPSoCs. **The LC does not use sensors.**

6.4 Limitations of the Proposal

The main limitations of this Thesis are:

- the adopted MPSoC architecture employs a single application repository. Further, only the GMP has access to such repository;
- the task mapping heuristics are limited to homogeneous NoC-based MPSoC;
- task mapping heuristics experiments considered only 3 real benchmarks;
- task mapping heuristics evaluation can be improved by using numeric models to demonstrate system reliability improvements.

6.5 Future Works

Aforementioned limitations do not invalidate the obtained results, but they point out possible future directions that can be investigated. Future works include:

- integration of a lifetime model to obtain an MTTF (Mean Time to Failure) metric. Different works [CHA13][HUA09][HAR12] in the literature use the MTTF as the main metric for evaluating lifetime reliability improvements. Such metric is computed by a lifetime model that considers different system aging effects. In this context, results of this Thesis can be better evaluated by showing the expected system lifetime when executing a given mapping heuristic.
- use of monitoring to better estimate processors wear-state and workload variations at runtime. A monitoring system also can capture system data at runtime to help heuristics to provide better mapping decisions.
- add the power due to the static consumption in the power reports to obtain more accurate thermal maps.
- explore task mapping heuristics for heterogeneous and 3D MPSoCs.
- explore system architectures using more application repositories instances, or architectures where a larger number of processors have access to the application repository.

REFERENCES

[ANA12] Anagnostopoulos, I.; Bartzas, A.; Kathareios, G.; Soudris, D. “A divide and conquer based distributed runtime mapping methodology for many-core platforms”. In: DATE, 2012, pp. 111-116.

[AUS02] Austin T.; Larson E.; Ernst D. “SimpleScalar: An Infrastructure for Computer System Modeling”. Computer, vol. 35(2), 2002, pp. 59–67.

[AZE05] Azevedo, R.; Rigo, S.; Bartholomeu, M.; Araujo, G.; Araujo, C.; Barros, E. The ArchC Architecture Description Language and Tools. International Journal of Parallel Programming, Vol. 33(5), October 2005, pp. 453–484.

[BAN02] Banakar, R.; Steinke, S.; Bo-Sik Lee; Balakrishnan, M.; Marwedel, P. "Scratchpad memory: a design alternative for cache on-chip memory in embedded systems". In: CODES, 2002, pp. 73-78.

[BEN02] Benini, L.; De Micheli, G. “Networks on chips: a new SoC paradigm”. IEEE Computer, vol. 35(1), January, 2002, pp. 70-78.

[BEN12] Benini, L.; Flamand, E.; Fuin, D.; Melpignano, D. "P2012: Building an Ecosystem for a Scalable, Modular and High-Efficiency Embedded Computing Accelerator". In: DATE, 2012, pp.983-987.

[BIN11] Binkert N.; et al. “The gem5 simulator”. ACM SIGARCH Computer Architecture News, v.39 (2), 2011, 7p.

[BOL13] Bolchini, C.; Carminati, M.; Miele, A.; Das, A.; Kumar, A.; Veeravalli, B. “Runtime mapping for reliable many-cores based on energy/performance trade-offs”. In: DFT, 2013, pp. 58–64.

[BUS11] Busseuil, R.; Barthe, L.; Almeida, G.; Ost, L.; Bruguier, F.; Sassatelli, G.; Benoit, P.; Robert, M.; Torres, L. “Open-Scale: A Scalable, Open-Source NoC-based MPSoC for Design Space Exploration”. In: ReConFig, 2011, pp.357-362.

[CAR09] Carara, E.; Oliveira,R.; Calazans,N.; Moraes,F."HeMPS - a Framework for NoC-based MPSoC Generation". In: ISCAS, 2009, pp. 1345-1348.

[CAR10] Carvalho, E.; Calazans, N.; Moraes, F. “Dynamic Task Mapping for MPSoCs”. IEEE Design and Test of Computers, vol. 27-5, Set-Oct 2010, pp. 26-35.

[CAS13] Castilhos, G. M.; Mandelli, M.; Madalozzo, G. A.; Moraes, F. “Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes”. In: ISVLSI, 2013, pp.153-158.

[CEN09] Ceng, J.; et al. “A High-Level Virtual Platform for Early MPSoC Software Development”. In: CODES+ISSS, 2009, pp. 11-20.

[CHA13] Chantem, T.; et al. "Enhancing multicore reliability through wear compensation in online assignment and scheduling". In: DATE, 2013, pp. 1373 -1378

[CHE08] Chen, J-J. et al. “Approximation Algorithms for Multiprocessor Energy-Efficient Scheduling of Periodic Real-Time Tasks with Uncertain Task Execution Time”. In: RTAS, 2008, pp. 13-23.

[CHO07] Chou, C-L.; Marculescu, R. “Incremental Runtime Application Mapping for Homogeneous NoCs with Multiple Voltage Levels”. In: CODES+ISSS, 2007, pp.161-166.

[CHO08] Chou, C-L.; Marculescu, R. “User-Aware Dynamic Task Allocation in Networks-on-Chip”. In: DATE, 2008, pp. 1232-1237.

[CHO10] Chou, C-L.; Marculescu, R. “Runtime task allocation considering user behavior in embedded multiprocessor networks-on-chip". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 29(1), 2010, pp. 78–91.

[COS07] Coskun, A.K.; et al. "Temperature Aware Task Scheduling in MPSoCs", In: DATE, 2007, pp. 1-6.

[COS09] Coskun, A.K.; et al. “Dynamic thermal management in 3D multicore architectures”. In: DATE, 2009, pp.1410-1415.

[CUI12] Cui, Y; Zhang, W; Yu, H. “Decentralized Agent Based Re-Clustering for Task Mapping of Tera-Scale Network-on-Chip System”. In: ISCAS, 2012, pp. 2437-2440.

[DAS13] Das, A.; Kumar, A.; Veeravalli, B. “Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems”. In: DATE, 2013, pp. 689–694.

[DAS14] Das, A; et al. “Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs”. In: DATE, 2014, pp. 1-6

[DE13] de Dinechin, B.D.; Ayrignac, R.; Beaucamps, P.-E.; Couvert, P.; Ganne, B.; de Massas, P.G.; Jacquet, F.; Jones, S.; Chaisemartin, N.M.; Riss, F.; Strudel, T., “A clustered manycore processor architecture for embedded and accelerated applications”. In: HPEC, 2013, pp.1-6

[DUE14] Duenha, L.; Guedes, M.; Almeida, H.; Boy, M.; Azevedo, R. "MPSoCBench: A toolset for MPSoC system level evaluation". In: SAMOS, 2014, pp.164-171.

[FAR08] Faruque, M. A.; et al. “ADAM: Runtime Agent-based Distributed Application Mapping for on-chip Communication”. In: DAC, 2008, pp. 760-765.

[FAZ08] Fazzino, F., M. Palesi, and D. Patti. “Noxim: Network-on-chip simulator”. Obtained in: http://sourceforge. net/projects/noxim, 2008.

[GAR14] Garibotti, R. F. “Exploration of Compute Accelerators for High Performance Computing”. PhD's thesis, Université Montpellier 2, 2014, 108 pages.

[GE10] Ge, Y.; et al. "Distributed task migration for thermal management in many-core systems" In: DAC, 2010, pp.579-584.

[GRA12] Gray, I.; Audsley, N.C., "Challenges in software development for multicore System-on-Chip development". In: RSP, 2012, pp.115-121.

[HAR12] Hartman, A., et al. “Lifetime improvement through runtime wear-based task mapping”. In: CODES+ISSS, 2012, pp. 13-22.

[HEN13] Henkel, J.; et al. “Reliable on-chip systems in the nano-era: Lessons learnt and future trends”. In: DAC, 2013, pp. 1-10.

[HÖL07] Hölzenspies, P. K. F.; Smit, G. J. M.; Kuper, J. “Mapping streaming applications on a reconfigurable MPSoC platform at runtime”. In: SoC, 2007, pp.1-4.

[HÖL08] Hölzenspies, P. K. F.; Hurink, J. L.; Kuper, J.; Smit, G. J. M. “Runtime Spatial Mapping of Streaming Applications to a Heterogeneous Multi-Processor System-on-Chip (MPSOC)”. In: DATE, 2008, pp. 212-217.

[HOS09] Hosseinabady, M.; Nunez-Yanes, J. "Runtime resource management in fault-tolerant network on reconfigurable chips". In: FPL, 2009, pp. 574–577.

[HU10] Hu, W.; Tang, X.; Bin Xie; Chen, T.; Wang, D. “An Efficient Power-Aware Optimization for Task Scheduling on NoC-based Many-core System”. In:CIT, 2010, pp. 171–178.

[HUA09] Huang, L.; et al. “Lifetime reliability-aware task allocation and scheduling for MPSoC platforms”. In: DATE, 2009, pp. 51-56

[IBS13] International Business Strategies, Inc. (IBS), 2013.

[IND14] Indrusiak,L. “End-to-end Schedulability Tests for Multiprocessor Embedded Systems based on Networks-on-Chip with Priority-Preemptive Arbitration”. Journal of Systems Architecture, v.60(7), 2014, pp.553-561.

[INT12] Intel. “The Intel® Xeon Phi™ Coprocessor”, 2012.

[ITR13] International Tecnology Roadmap for Semiconductors. Accessed in: http://www.itrs.net/reports.html. February 2013.

[JAN03] Jantsch, A. “Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation”. San Francisco: Morgan Kaufmann Publishers Inc., 2003, 375p.

[JEJ04] Jejurikar, R., Pereira, C. and Gupta, R. “Leakage aware dynamic voltage scaling for real-time embedded systems”. In: DAC, 2004, pp. 275-280.

[KHA12] Khajekarimi, E.; Hashemi, M. R. “Communication and congestion aware runtime task mapping on heterogeneous MPSoCs”. In: CADS, 2012, pp. 127–132.

[KOB11] Kobbe, S.; Bauer, L.; Lohmann, D.; Schroder-Preikschat, W.; Henkel, J. “DistRM: Distributed Resource Management for On-Chip Many-Core Systems”. In: CODES+ISSS, 2011, pp. 119-128.

[LEM12] Lemaire, R.; Thuries, S.; Heiztmann, F. “A flexible modeling environment for a NoC-based multicore architecture”. In: High Level Design Validation and Test Workshop, 2012, pp. 140-147.

[LIU15] Liu, Z.; et al. "Task Migrations for Distributed Thermal Management Considering Transient Effects" IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.23(2), 2015, pp.397-401

[LU10] Lu, S.; Lu, C.; Hsiung , P. “Congestion- and energy-aware runtime mapping for tile-based network-on-chip architecture”. In: Frontier Computing. Theory, Technologies and Applications, 2010, pp. 300 – 305.

[LUK08] Lukovic, A.; Fiorin, L. “An Automated Design Flor for NoC-based MPSoCs on FPGAs”. In: RSP, 2008, pp.58-64.

[MAN11a] Mandelli, M. G.; Ost, L. C.; Carara, E. A.; Guindani, G. M.; Rosa, T.; Medeiros, G.; Moraes, F. G. “Energy-aware dynamic task mapping for NoC-based MPSoCs”. In: ISCAS, 2011, pp. 1676-1679.

[MAN11b] Mandelli, M. G.; Ost, L. C.; Amory, A. M.; Moraes, F. G. “Multi-Task Dynamic Mapping onto NoC-based MPSoCs”. In: SBCCI, 2011, pp. 191-196

[MAN12] Mandelli, M; Castilhos, G. M.; Moraes, F.G. “Enhancing Performance of MPSoCs through Distributed Resource Management”. In: ICECS, 2012, p. 544-547.

[MAN13] Mandelli, M.; Rosa, F.; Ost, L.; Sassatelli, G.; Moraes, F. G. “MPSoC Modeling for Reducing Software Development”. In: ICECS, 2013, pp. 489-492.

[MAN15] Mandelli, M.; Ost, L.; Sassatelli, G.; Moraes, F. "Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability". In: ISQED, 2015, pp.392-396

[MAR12] Marongiu, A.; Benini, L. “An OpenMP Compiler for Efficient Use of Distributed Scratchpad Memory in MPSoCs”. IEEE Transactions on Computers, vol. 62(1), 2012, pp. 222-236.

[MAR14] Martins, A.; Silva, D.; Castilhos, G.; Monteiro, T.; Moraes, F. "A method for NoC-based MPSoC energy consumption estimation". In: ICECS, 2014, pp. 427-430

[MEI10] Meier, M.; Engel, M.; Steinkamp, M.; Spinczyk, O. “LavA: An Open Platform for Rapid Prototyping of MPSoC”. In: FPL, 2010, pp.452-457.

[MEY14] Meyer, B; et al. “Cost-effective lifetime and yield optimization for NoC-based MPSoCs”. In: ACM Transactions on Design Automation Electronic Systems, vol. 19(2), 2014

[MOR04] Moraes, F.; Calazans, N.; Mello, A.; Möller, L.; Ost, L. “HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip”. *Integration, the VLSI Journal*, vol. 38-1, Oct. 2004, pp. 69-93.

[MUR07] Murali, S.; et al. “Temperature-aware processor frequency assignment for MPSoCs using convex optimization”. In: CODES+ISSS, 2007, pp. 111-116.

[NGO06] Ngouanga, A.; Sassatelli, G.; Torres, L.; Gil, T.; Soares, A.; Susin, A. “A contextual re-sources use: a proof of concept through the APACHES platform”. In: DDECS, 2006, pp.42-47.

[OST10] Ost, L. C.; Indrusiak, L. S.; Maatta, S.; Mandelli, M. G.; Nurmi, J.; Moraes, F. G. “Model-based design flow for NoC-based MPSoCs”. In: ICECS, 2010, pp. 750-753.

[OST11a] Ost, L. C.; Mandelli, M. G.; Almeida, G. M.; Indrusiak, L. S.; Moller, L. S.; Glesner, M.; Sassatelli, G.; Robert, M.; Moraes, F. G.. “Exploring dynamic mapping impact on NoC-based MPSoCs performance using a model-based framework”. In: SBCCI, 2011, pp. 185-190.

[OST11b] Ost, L. C.; Almeida, G. M.; Mandelli, M. G.; Wachter, E.; Varyani, S.; Indrusiak, L. S.; Sassatelli, G.; Robert, M.; Moraes, F. G. “Exploring Heterogeneous NoC-based MPSoCs: from FPGA to High-Level Modeling”. In: RECOSOC, 2011, pp 1-8.

[OST12] Ost, L. C.; Varyani, S.; Mandelli, M. G.; Wachter, E.; Almeida, G. M.; Indrusiak, L. S.; Sassatelli, G.; Moraes, F. G. “Exploring Adaptive Techniques in Heterogeneous MPSoCs based on Virtualization”. ACM Transactions on Reconfigurable Technology and Systems, vol. 5(3), 2012, pp. 1-11.

[OST13] Ost, L. C.; Mandelli, M. G.; Almeida, G. M.; Moller, L. S.; Indrusiak, L. S.; Sassatelli, G.; Benoit, P.; Glesner, M.; Robert, M.; Moraes, F. G. “Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach”. ACM Transactions on Embedded Computing Systems, vol. 12(3), 2013, pp. 1 - 22.

[OVP13] OVP 2013, Available at: www.ovpworld.org/technology_ovpsim.php

[PAU06] Paulin, P.; Pilkington, C.; Langevin, M.; Bensoudane, E.; Lyonnard, D.; Benny, O.; Lavigueur, B.; Lo, D.; Beltrame, G.; Gagné, V.; Nicolescu, G. “Parallel Pragramming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia”. In: VLSI, 2006, pp.667-680.

[PET12] Petry, C.A.; Wachter, E.W.; de Castilhos, G.M.; Moraes, F.G.; Calazans, N. L V, "A spectrum of MPSoC models for design space exploration and its use," RSP, 2012, pp. 30-35.

[REK13] Rekik, W; Ben Said, M; Ben Amor, N. “Virtual Prototyping of Multiprocessor Architectures Using the Open Virtual Platform”. In: ICCAT, 2013, pp. 1-6.

[ROS13] Rosa, F.; Ost L.; Reis, R.; Sassatelli G. “Instruction-driven Timing CPU Model for Efficient Embedded Software Development using OVP”. In: ICECS, 2013, pp. 855-858.

[ROS14] Rosa, F., Ost, L., Raupp, T., Moraes, F. and Reis, R. “Fast energy evaluation of embedded applications for many-core systems”. In: PATMOS, 2014, pp. 1-6.

[ROT13] Roth, C.; Bucher, H.; Reder, S.; Buciuman, F.; Sander, O.; Becker, J., "A SystemC modeling and simulation methodology for fast and accurate parallel MPSoC simulation". In: SBCCI, 2013, pp.1-6

[SCH10] Schranzhofer, A.; Chen, J.-J.; Thiele, L. "Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms". IEEE Transactions on Industrial Informatics, vol. 6(4), 2010, pp. 692-707.

[SHA11] Shabbir, A.; Kumar, A.; Mesman, B.; Corporaal, H. “Distributed Resource Management for Concurrent Execution of Multimedia Applications on MPSoC Platforms”. In:SAMOS, 2011, pp. 132-139.

[SIM13] Simics, Available at : www.windriver.com/products/simics

[SIN09a] Singh, A.K.; Wu Jigang; Prakash, A.; Srikanthan, T. “Mapping Algorithms for NoC-based Heterogeneous MPSoC Platforms”. In: Euromicro, 2009, pp. 133-140.

[SIN09b] Singh, A.K. et al. “Eficient heuristics for minimizing communication overhead in NoC-based heterogeneous MPSoC platforms”. In: RSP, 2009, pp. 55-60.

[SIN10] Singh, A. K.; et al. "Communication-aware heuristics for runtime task mapping on NoC-based MPSoC platforms". *Journal of Systems Architecture: the EUROMICRO Journal*, vol. 56-7, Jul 2010, pp. 242-255.

[SIN13] Singh, A.; et al. “Mapping on multi/many-core systems: survey of current and emerging trends”. In: DAC, 2013, pp. 1-10.

[SMI05] Smit, L.T.; Hurink, J.L.; Smit, G.J.M. “Runtime mapping of applications to a heterogeneous SoC”. In: SoC, 2005, pp.78-81.

[TAN08] Tan, J.; Zhang, L.; Fresse, V.; Legrand, A.; Houzet, D. “A predictive and parametrized architecture for image analysis algorithm implementations on FPGA adapted to multispectral imaging”. In: IPTA, 2008, pp 1-8.

[TIA09] Tian, G.; Hammami, O. “Performance Measurements of Synchronization Mechanisms on 16PE NoC-based Multi-Core with Dedicated Synchronization and Data NoC”. In: ICECS, 2009, pp.988-991.

[TIL10] Tilera Corporation. “Tile-GX Processor Family”, 2010.

[TIW94] Tiwari. V.; et al. “Power analysis of embedded software: a first step towards software power minimization”. IEEE Transactions Very Large Scale Integration Systems, v.2(4), 1994, pp. 437–445.

[VEN10] Ventroux, N.; Guerre, A; Sassolas, T.; Moutaoukil, L.; Blanc, G.; Bechara, C.; David, R. "SESAM: An MPSoC Simulation Environment for Dynamic Application Processing". In: CIT, 2010, pp.1880-1886.

[VIL11] Villavieja, C; Etsion, Y.; Ramirez, A.; Navarro, N. “FELI: HW/SW Support for On-Chip Distributed Shared Memory in Multicores”. In: Euro-Par, 2011, pp. 282–294.

[WAN14] Wang, Z; et al. “System-level reliability exploration framework for heterogeneous MPSoC”. In: GLSVLSI, 2014, pp 9-14.

[WEI06] Wei Huang; et al. “HotSpot: a compact thermal modeling methodology for early-stage VLSI design”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, v.14(5), 2006, pp. 501-513.

[WEI11] Weichslgartner, A.; Wildermann, S.; Teich, J. "Dynamic Decentralized Mapping of Tree-Structured Applications on NoC Architectures". In: NoCs, 2011, pp. 201-209.

[WIL09] Wildermann, S.; Ziermann, T.; Teich, J. "Run time Mapping of Adaptive Applications onto Homogeneous NoC-based Reconfigurable Architectures". In: FPT, 2009, pp. 514 - 517.

[WOS07] Woszezenki, C. “Alocação de Tarefas e Comunicação entre Tarefas em MPSoCs”. Dissertação de Mestrado, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2007, 121p.

[WU11] Wu, Y-K; et al. "Distributed thermal management for embedded heterogeneous MPSoCs with dedicated hardware accelerators" In: ICCD, 2011, pp.183-189.

[YE02] Ye, T.; Benini, L.; De Micheli, G. "Analysis of Power Consumption on Switch Fabrics in Network Routers". In: DAC, 2002, pp. 524-529.

[YOU07] Yourst M.T. “PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator”. In: ISPASS, 2007, pp. 23–34.

[ZHA13] Zhang, D.; Zeng, X.; Wang, Z.; Wang, W.; Chen, X. "MCVP-NoC: Many-Core Virtual Platform with Networks-on-Chip support". In: ASICON, 2013, pp. 28-31.

APPENDIX A – WORKLOAD DISTRIBUTION

This appendix first complements the results of Section 5.5.1.1 (OVP platform) and 5.5.2.1 (SystemC platform), where the heuristics are evaluated concerning the workload distribution. Results presented in this appendix show the energy consumption per PE for scenarios B to F of Table 16. The L and LC heuristics have a better workload balancing compared to PREMAP-DN and LEC-DN heuristics in most scenarios. Such heuristics have a smaller number of SPs with a low energy consumption (green rectangles) and high energy consumption (red rectangles).

A.1 Results for OVP Platform

78.75	236.24	79.32	158.05	0.48	78.64	235.74	217.21	157.34	0.48
335.64	412.30	236.45	354.91	157.01	336.19	413.36	236.35	356.06	156.59
158.10	472.08	353.51	412.56	157.28	158.14	472.40	353.82	353.93	156.71
137.79	235.16	473.20	411.73	156.94	0.48	157.01	474.09	413.48	78.73
LMP	0.48	236.63	216.64	156.91	LMP	0.48	236.54	78.77	78.95
0.48	156.87	235.81	0.48	0.48	156.19	78.20	354.73	156.10	0.48
157.53	587.55	411.69	236.13	79.32	354.70	413.22	157.53	183.69	79.04
79.41	352.67	587.02	236.81	158.12	0.48	412.74	529.90	411.71	234.88
0.48	235.77	411.53	235.66	235.93	0.48	131.42	295.20	413.00	156.95
GMP	236.85	236.21	587.74	156.80	LMP	0.48	235.14	274.93	156.56

(a) PREMAP-DN

78.90	334.25	217.80	78.80	79.38	78.38	78.34	216.77	156.78	0.48
157.59	413.46	157.82	295.78	78.39	216.92	414.33	158.22	235.48	157.15
78.82	414.57	530.32	235.08	236.40	78.83	533.15	352.38	531.67	236.34
216.18	413.20	294.61	412.91	78.56	0.48	157.70	295.28	594.28	235.49
LMP	78.15	158.03	78.98	78.82	LMP	0.48	157.11	157.55	157.60
26.97	157.82	353.05	78.15	78.79	79.30	235.49	335.43	78.50	79.10
215.98	411.83	431.00	209.99	78.76	79.15	411.46	235.76	157.56	0.48
0.48	372.71	899.42	315.48	314.08	131.87	531.77	471.85	234.10	105.55
105.36	411.79	374.09	235.49	274.15	0.48	157.08	412.78	471.71	234.58
GMP	235.87	210.53	450.45	235.40	LMP	0.48	157.80	78.65	79.07

(b) LEC-DN

158.07	157.90	217.67	365.03	157.53	158.01	217.99	157.80	218.63	157.72
278.01	157.85	220.97	304.47	278.82	278.26	303.60	303.86	158.61	303.34
217.89	283.45	157.97	218.19	302.98	304.09	157.95	218.39	157.89	220.46
217.07	157.90	157.97	157.93	304.78	363.34	278.25	157.61	158.07	279.58
LMP	158.36	303.20	303.23	218.92	LMP	157.66	218.38	304.51	218.13
216.54	157.72	157.67	217.75	218.47	158.14	221.20	158.19	304.42	158.00
420.63	219.53	157.29	302.27	157.68	302.77	276.80	218.35	218.31	158.17
362.30	306.15	158.51	158.55	158.50	278.32	303.73	157.38	217.83	218.46
421.84	217.65	158.20	158.08	217.35	278.26	157.62	157.93	157.96	304.83
GMP	306.38	157.32	217.02	216.09	LMP	157.73	365.39	304.84	217.43

(c) L

78.65	305.19	422.47	209.21	449.65	422.93	156.82	27.01	210.16	506.38
182.52	184.82	158.23	185.68	183.86	157.61	185.42	184.18	271.66	208.95
421.37	158.83	79.88	156.91	217.75	164.36	183.53	542.17	184.73	53.49
244.99	302.49	131.57	132.24	362.26	216.99	185.35	183.39	185.20	157.29
LMP	157.25	282.66	157.35	278.17	LMP	339.58	130.63	157.01	423.10
333.51	158.72	190.06	269.47	445.66	421.23	297.54	276.56	278.72	217.36
215.96	158.19	156.42	183.64	210.88	181.89	236.16	218.46	132.61	157.04
156.48	132.81	333.65	217.60	132.62	219.30	216.28	131.10	129.53	219.17
250.91	133.99	216.83	243.07	182.93	361.24	218.95	157.67	218.57	277.34
GMP	254.17	131.43	209.47	504.30	LMP	218.07	216.07	157.36	277.49

(d) LC

Red rectangles: SPs consuming more than 400 mJ. Green rectangles: SPs consuming less than 100 mJ.

Figure 52 – Energy consumed per SP in **Scenario B** in the OVP platform.

0.48	0.48	44.23	15.62	3.44	0.48	20.17	6.46	45.35	3.53
0.48	56.12	48.98	41.18	21.36	15.49	13.38	59.66	51.39	5.92
44.38	48.38	60.16	63.91	33.61	45.11	46.64	56.52	44.55	8.81
15.39	44.31	62.92	56.15	47.52	55.77	44.65	46.35	56.94	44.06
LMP	15.52	44.80	0.48	0.48	LMP	0.48	0.48	43.80	0.48
0.48	17.72	33.34	16.35	11.38	0.48	5.91	0.48	0.48	0.48
49.70	106.66	95.65	66.35	31.07	0.48	41.00	42.93	34.99	4.89
0.48	47.32	102.08	60.06	32.37	0.48	52.53	101.64	47.28	8.53
0.48	45.67	108.20	99.86	64.01	0.48	43.17	58.52	10.53	19.81
GMP	13.74	49.76	0.48	0.48	LMP	0.48	42.76	8.47	3.08

(a) PREMAP-DN

1.38	6.74	45.18	0.48	0.48	0.48	15.61	6.21	89.84	0.48
0.48	64.47	58.15	38.04	6.40	43.41	15.20	92.38	103.04	22.32
44.57	61.40	59.69	34.95	21.01	45.66	51.30	71.39	57.60	13.42
0.48	44.71	72.83	60.54	6.58	43.23	30.04	46.34	20.43	5.91
LMP	15.77	45.38	0.48	0.48	LMP	0.48	0.48	0.48	0.48
46.81	40.28	48.92	49.62	5.76	0.48	0.48	8.36	26.34	13.17
48.07	64.79	63.77	69.58	32.65	0.48	30.47	45.21	24.81	18.90
47.85	55.66	69.64	47.61	5.14	45.48	100.46	50.35	52.70	6.15
21.65	46.27	48.79	72.23	49.90	0.48	45.32	45.35	56.64	31.54
GMP	10.46	0.48	45.64	0.48	LMP	0.48	0.48	46.24	0.48

(b) LEC-DN

27.37	26.87	33.63	41.81	29.36	29.09	16.74	28.99	16.36	28.98
32.99	46.31	44.52	33.31	24.35	29.36	17.39	29.26	17.57	29.15
32.76	20.56	33.09	25.64	24.36	23.74	17.87	23.85	16.92	22.91
32.86	21.02	28.50	18.39	26.03	44.86	46.75	45.34	23.42	43.70
LMP	32.15	19.54	26.67	45.98	LMP	23.30	29.22	23.35	28.88
33.33	23.97	33.05	32.76	30.51	24.02	37.34	33.53	18.48	28.58
34.24	26.51	32.94	41.87	54.64	36.86	33.84	33.12	33.21	27.83
41.73	24.83	50.76	52.28	25.47	44.96	30.98	18.10	30.17	29.66
57.20	32.56	36.99	27.75	33.41	32.77	18.21	45.72	33.66	30.40
GMP	34.69	40.87	30.56	26.69	LMP	39.21	49.67	46.78	40.52

(c) L

30.96	30.93	34.20	53.38	33.18	24.35	26.52	20.05	43.94	30.27
18.53	33.87	35.20	53.90	33.48	23.95	24.08	30.06	50.90	15.16
21.10	32.97	35.74	30.23	29.66	23.07	30.10	22.84	29.58	29.00
14.16	26.85	19.11	18.75	25.94	43.92	15.47	29.96	15.37	29.88
LMP	24.24	46.02	9.36	18.82	LMP	30.07	22.91	15.37	45.28
30.87	38.82	47.00	66.37	27.63	43.65	17.92	49.08	61.16	44.17
26.25	32.47	39.97	34.98	30.35	15.54	29.77	56.77	33.22	63.50
26.15	30.44	30.91	31.80	53.51	22.79	17.95	18.49	24.42	23.80
44.98	18.81	42.58	27.64	31.56	17.38	30.12	41.19	27.19	27.28
GMP	16.26	33.60	24.43	49.64	LMP	25.02	55.32	29.87	19.81

(d) LC

Red rectangles: SPs consuming more than 50 mJ. Green rectangles: SPs consuming less than 5 mJ.

Figure 53 – Energy consumed per SP in **Scenario C** in the OVP platform.

0.48	84.79	14.64	13.26	5.91	3.62	3.34	130.71	0.48	5.84
0.48	253.34	143.17	19.77	15.07	14.64	116.00	278.27	108.76	40.31
79.27	362.33	300.66	154.67	70.48	26.97	215.97	124.57	120.42	70.21
0.48	92.11	132.41	111.65	39.07	0.48	139.99	187.23	104.63	67.12
LMP	26.97	4.77	77.80	0.48	LMP	5.10	98.26	9.67	1.38
26.97	91.12	7.90	7.24	27.71	2.99	171.58	6.27	78.21	27.43
137.49	252.18	154.55	82.40	29.15	55.69	253.21	115.59	113.10	39.38
0.48	230.75	327.40	117.38	78.17	138.44	78.30	218.73	180.00	9.16
26.97	274.75	241.68	189.22	78.15	55.80	97.71	61.17	218.72	80.68
GMP	37.99	29.47	164.94	78.84	LMP	77.60	3.53	79.89	0.48

(a) PREMAP-DN

0.48	6.41	156.37	81.80	0.48	0.48	0.48	65.62	3.53	0.48
14.03	71.96	84.57	157.84	86.63	0.48	156.52	258.46	160.94	6.58
4.94	86.04	258.74	156.26	9.46	91.58	376.43	168.08	136.26	86.34
52.88	209.59	98.15	104.18	58.89	0.48	167.73	197.06	111.64	87.76
LMP	82.11	3.07	77.95	0.48	LMP	5.58	62.13	39.45	13.63
0.48	133.53	2.98	0.48	0.48	0.48	96.68	7.31	53.83	0.48
0.48	275.60	120.44	224.60	78.38	0.48	226.15	157.15	147.38	86.73
91.49	170.31	218.67	275.79	82.64	53.31	338.78	198.80	138.54	86.24
0.48	13.99	183.19	212.14	83.50	0.48	118.82	214.61	148.17	20.05
GMP	78.09	152.91	68.86	14.23	LMP	0.48	78.01	0.93	6.81

(b) LEC-DN

45.26	79.24	50.89	79.08	140.09	105.82	64.93	73.24	130.58	46.42
138.52	140.17	140.93	73.89	44.82	126.49	87.61	70.82	141.26	62.27
125.03	62.04	57.13	199.13	83.25	64.99	87.75	57.15	134.12	64.61
65.60	144.45	47.90	85.28	82.54	64.84	62.21	117.64	79.58	45.80
LMP	70.69	80.08	49.10	148.30	LMP	61.82	63.09	44.01	114.51
144.21	61.13	141.40	69.57	81.92	151.60	67.22	62.13	148.46	151.75
141.21	141.20	204.71	141.56	79.35	128.19	67.57	86.23	67.56	110.29
84.54	82.00	61.63	64.51	62.37	56.40	51.88	148.59	68.20	70.59
138.57	61.87	139.92	84.47	138.90	57.48	137.92	48.40	71.28	128.43
GMP	79.62	87.00	65.46	165.52	LMP	51.93	67.14	69.44	69.58

(c) L

165.13	226.19	89.92	41.25	127.99	136.06	68.61	57.43	103.58	193.40
78.57	83.21	82.82	140.02	79.32	52.92	44.96	43.64	85.79	157.40
18.79	21.46	59.84	98.94	64.56	99.77	40.09	48.45	63.52	85.46
75.44	68.42	44.77	83.15	82.62	80.69	109.39	111.58	45.03	21.92
LMP	114.51	32.97	170.48	168.06	LMP	108.52	290.27	69.71	18.00
227.57	90.37	14.32	64.33	231.27	145.19	145.29	48.03	78.97	113.91
87.58	85.21	35.45	87.92	87.81	62.30	65.70	65.00	38.89	140.17
81.57	35.59	104.89	73.54	99.15	88.25	69.50	122.28	56.77	58.64
139.99	142.70	38.99	45.02	149.35	210.24	94.38	39.65	64.42	90.10
GMP	36.11	70.90	64.18	133.02	LMP	74.43	22.72	87.68	234.36

(d) LC

Red rectangles: SPs consuming more than 200 mJ. Green rectangles: SPs consuming less than 20 mJ.

Figure 54 – Energy consumed per SP in **Scenario D** in the OVP platform.

1.39	139.44	143.04	78.38	0.93	0.48	7.03	138.12	78.38	0.48
44.06	190.47	133.60	137.87	78.20	47.81	98.11	79.43	62.66	5.91
0.48	183.14	129.53	93.10	13.43	87.50	186.40	239.00	171.34	104.59
0.48	14.79	45.21	196.56	78.17	47.99	79.98	78.75	113.27	32.18
LMP	13.82	5.08	78.90	0.48	LMP	5.61	8.86	77.80	0.48
0.48	131.89	26.97	53.41	0.48	0.48	44.53	76.27	63.11	36.22
78.95	307.57	147.31	255.71	155.51	0.48	124.30	160.89	51.03	6.81
52.63	144.60	104.88	57.35	6.21	46.89	199.29	236.50	130.43	22.67
77.78	151.51	186.49	112.98	16.70	0.48	46.43	83.06	101.05	59.19
GMP	33.65	53.56	138.56	78.40	LMP	0.48	0.48	88.28	0.48

(a) PREMAP-DN

0.93	155.44	0.48	0.48	0.48	0.48	88.68	4.37	77.79	32.97
31.81	212.72	186.81	138.14	78.21	77.81	241.20	120.51	93.29	43.98
0.48	57.79	282.73	119.70	0.93	0.48	96.56	242.98	122.07	36.62
53.42	98.19	129.89	198.89	78.16	0.48	30.26	78.63	114.04	47.67
LMP	77.60	0.93	78.91	0.48	LMP	0.48	0.48	77.80	0.48
0.48	79.02	26.97	0.48	0.48	46.48	45.93	60.71	46.12	0.48
78.95	195.93	73.18	216.61	78.38	0.48	61.02	160.61	50.07	47.73
2.80	87.27	84.36	195.77	83.92	89.80	87.38	249.45	164.58	91.77
70.93	140.65	91.35	118.25	19.22	0.48	61.06	87.56	258.01	125.80
GMP	77.61	32.20	145.82	81.55	LMP	0.48	0.48	124.44	58.09

(b) LEC-DN

50.93	40.83	133.51	50.41	41.04	96.89	68.47	57.18	126.65	62.00
49.62	41.55	135.42	71.71	30.25	66.63	68.53	48.05	28.33	213.77
100.56	31.01	30.38	80.14	31.38	150.08	57.25	39.88	68.63	39.58
43.77	41.30	116.26	44.66	31.49	139.28	62.07	68.46	44.00	62.37
LMP	30.48	30.23	45.15	50.65	LMP	153.39	63.64	61.15	57.98
53.36	49.61	70.39	139.05	79.60	50.65	41.65	128.03	170.12	42.85
53.57	50.32	70.15	143.40	44.57	139.20	45.01	41.62	130.65	67.50
98.01	51.18	81.53	79.20	163.54	134.14	67.60	67.78	88.91	67.98
113.72	128.25	165.19	150.06	138.75	142.31	44.85	44.91	53.03	50.07
GMP	49.33	133.20	44.47	149.09	LMP	89.36	131.77	76.08	50.37

(c) L

69.88	41.68	52.95	78.63	198.90	224.62	104.75	36.93	101.84	150.79
117.37	56.53	41.89	67.85	20.90	66.68	66.62	32.83	93.14	97.31
96.20	56.46	22.83	23.53	66.68	35.32	43.71	32.93	32.13	29.91
54.12	45.05	34.08	23.47	30.23	112.30	83.56	93.62	77.49	25.38
LMP	39.60	40.88	19.53	50.43	LMP	79.62	207.40	52.36	18.50
114.15	78.58	198.22	123.39	32.22	43.52	48.02	172.68	154.29	41.16
76.13	95.28	81.92	80.37	88.30	39.20	50.07	98.21	53.86	153.38
80.22	56.78	37.93	84.77	197.68	49.04	55.50	58.56	58.72	60.36
130.65	67.74	52.29	131.39	105.59	144.39	55.97	53.30	79.03	61.73
GMP	51.94	31.14	137.02	153.62	LMP	55.79	113.59	79.27	174.75

(d) LC

Red rectangles: SPs consuming more than 150 mJ. Green rectangles: SPs consuming less than 15 mJ.

Figure 55 – Energy consumed per SP in **Scenario E** in the OVP platform.

47.65	61.43	15.92	46.87	0.48	0.48	46.27	18.07	46.79	0.48
102.00	105.01	104.05	121.03	87.81	46.60	107.61	110.19	91.41	44.33
62.92	89.06	104.40	122.61	82.68	16.90	45.85	106.81	89.54	88.27
15.60	90.48	104.22	82.80	44.62	16.00	92.85	121.73	90.22	44.39
LMP	57.37	90.54	0.48	0.48	LMP	92.24	92.38	44.18	44.72
91.51	71.76	80.91	49.53	0.48	47.40	61.80	15.97	47.25	0.48
59.10	99.45	93.15	105.15	49.99	100.86	105.12	104.70	121.43	86.97
94.15	96.23	121.50	115.82	92.34	62.70	89.83	104.45	123.95	83.29
47.36	95.82	113.42	125.09	51.40	15.99	90.96	106.49	83.15	44.94
GMP	34.33	95.93	50.02	47.70	LMP	57.25	92.08	0.48	0.48

(a) PREMAP-DN

47.65	61.43	15.92	47.63	0.48	0.48	46.22	18.21	47.25	0.48
102.00	104.77	104.14	122.65	87.83	46.48	107.03	110.46	90.63	44.68
62.92	89.52	104.46	122.63	82.83	16.89	45.86	106.70	89.91	88.79
16.11	91.23	104.47	83.42	44.59	15.99	91.78	122.54	90.10	44.46
LMP	57.56	90.76	0.48	0.48	LMP	91.92	91.18	44.33	44.73
91.29	56.21	80.49	49.52	0.48	47.40	61.82	16.13	47.26	0.48
59.31	98.85	94.17	105.71	50.46	100.98	105.46	104.61	122.14	87.55
110.98	96.66	120.78	116.63	92.50	62.69	90.50	105.26	122.95	83.84
48.02	96.52	113.41	125.12	52.25	15.88	90.78	105.58	82.95	44.87
GMP	36.28	95.31	49.47	48.75	LMP	57.40	91.18	0.48	0.48

(b) LEC-DN

78.28	58.90	116.09	59.19	59.12	76.11	58.32	101.34	58.35	59.23
58.24	58.75	77.88	52.09	59.59	57.81	59.34	78.16	51.83	59.49
113.88	45.67	53.75	58.83	44.41	114.13	44.77	53.11	58.99	58.19
74.77	79.62	74.68	83.51	44.12	74.42	77.54	74.29	84.08	43.71
LMP	57.37	58.85	52.90	57.38	GMP	57.23	59.11	51.92	57.55
63.42	64.14	106.13	63.79	60.56	77.24	60.17	115.43	58.62	59.36
107.00	85.65	64.19	57.27	91.43	58.91	59.58	76.88	51.77	59.00
56.48	78.85	61.21	65.30	56.41	115.77	45.19	53.81	58.53	44.34
77.04	62.84	65.86	54.95	79.84	74.59	78.67	74.45	84.02	43.82
GMP	65.69	77.66	57.67	61.94	LMP	57.89	60.04	53.46	57.98

(c) L

51.99	52.59	74.10	117.29	86.84	52.02	65.72	71.89	87.98	102.43
74.13	52.71	72.87	108.47	59.38	87.12	52.44	72.89	113.61	72.90
81.90	45.00	66.57	66.41	57.88	66.85	59.33	78.87	65.83	72.23
75.58	44.45	44.89	44.33	45.02	44.92	44.64	44.52	43.98	44.67
LMP	30.03	52.54	52.12	102.46	LMP	44.23	66.44	37.69	59.72
46.08	60.94	72.85	57.23	109.82	73.90	67.24	59.57	74.71	85.76
63.92	45.38	59.58	55.52	59.94	51.46	66.89	91.00	109.78	59.51
71.71	46.97	30.07	62.12	79.01	81.83	59.76	66.54	66.18	58.68
75.13	62.44	72.03	74.94	90.70	74.44	44.78	44.48	44.83	44.24
GMP	74.80	55.92	60.97	119.22	LMP	30.16	52.25	51.71	103.45

(d) LC

Red rectangles: SPs consuming more than 100 mJ. Green rectangles: SPs consuming less than 25 mJ.

Figure 56 – Energy consumed per SP in **Scenario F** in the OVP platform.

A.2 Results for SystemC Platform

186.95	109.55	192.71	109.57	109.54	192.72	109.61	192.98	109.54	109.61
106.91	108.29	192.70	109.56	192.73	109.60	108.29	192.85	109.58	192.64
187.05	190.16	288.77	163.43	109.53	192.76	190.02	288.57	163.44	109.55
186.96	109.55	192.92	109.54	192.71	192.69	109.57	192.88	109.57	192.97
LMP	192.72	109.58	192.86	109.58	LMP	192.72	109.57	192.81	109.57
185.84	110.17	192.19	109.74	109.64	192.77	109.59	192.79	109.60	109.56
106.89	108.94	191.83	109.55	192.43	109.58	108.36	192.80	109.57	192.79
186.02	189.20	285.02	163.22	109.69	193.00	190.69	288.65	163.52	109.61
185.87	110.24	189.32	109.06	192.31	192.77	109.58	190.15	108.32	192.93
GMP	187.60	106.86	188.23	106.94	LMP	187.74	106.87	187.40	106.92

(a) PREMAP-DN

186.95	109.55	192.75	109.57	109.54	192.80	109.57	151.30	151.16	109.56
106.90	108.29	192.70	109.56	192.79	109.56	108.29	192.85	109.58	151.82
187.00	190.16	288.77	163.43	109.53	192.77	190.02	288.57	163.44	109.85
187.04	109.54	192.92	109.54	192.59	192.70	109.58	192.88	109.57	235.57
LMP	192.77	109.59	192.79	109.54	LMP	192.76	109.56	192.73	109.58
185.97	110.18	192.26	109.72	109.65	192.77	109.59	192.79	109.60	109.58
106.89	108.94	191.43	110.24	192.45	109.58	108.36	192.80	109.57	192.89
186.14	189.19	284.61	163.21	110.32	193.46	190.69	288.65	163.52	109.61
185.97	110.18	189.32	109.06	192.00	192.85	109.57	190.15	108.32	192.79
GMP	187.36	106.82	187.95	107.60	LMP	187.75	106.89	187.31	106.94

(b) LEC-DN

246.92	127.57	194.12	151.09	279.47	235.59	109.02	236.30	212.13	109.75
124.85	128.10	255.15	194.74	127.99	194.61	91.44	152.23	152.52	128.09
107.11	210.35	212.65	146.91	170.14	91.65	277.73	109.53	110.38	127.71
268.83	145.49	146.21	127.21	110.24	302.48	109.98	90.56	109.98	127.78
LMP	171.10	211.73	110.14	110.14	LMP	194.22	109.13	109.42	151.65
124.52	170.06	128.15	235.25	280.64	193.51	109.45	217.35	151.75	261.21
106.29	170.29	191.74	194.66	151.73	108.97	109.68	131.71	133.54	109.72
124.30	254.50	127.50	128.02	109.42	109.29	215.87	236.18	109.64	109.95
184.04	151.42	127.22	193.15	212.36	278.92	109.81	109.81	109.47	109.67
GMP	149.11	125.38	148.68	125.51	LMP	131.16	107.26	106.60	107.70

(c) L

223.63	128.59	150.14	109.86	236.13	274.03	128.89	150.50	110.09	193.18
182.98	128.67	150.57	193.31	109.81	188.22	128.96	150.77	193.47	109.93
188.41	109.94	193.94	110.13	192.91	194.14	109.90	194.05	110.19	192.76
186.23	108.52	151.92	110.94	150.34	149.58	108.73	151.83	110.69	109.22
LMP	151.38	152.10	152.04	193.92	LMP	151.55	152.33	194.55	236.76
187.87	110.47	191.59	109.73	151.89	189.40	128.14	150.61	109.93	236.42
147.21	235.80	109.40	236.65	110.43	230.52	128.71	150.50	193.17	109.91
205.33	126.58	151.43	152.07	109.78	237.18	109.61	193.79	110.18	192.97
218.85	127.98	110.49	151.22	151.14	149.75	108.59	152.11	110.25	150.64
GMP	125.30	147.29	187.81	186.07	LMP	147.71	147.95	146.48	185.80

(d) LC

Red rectangles: SPs consuming more than 200 mJ. Green rectangles: SPs consuming less than 100 mJ.

Figure 57 – Energy consumed per SP in **Scenario B** in the SystemC platform.

0.02	0.02	23.11	0.02	0.02	0.02	10.77	3.40	33.72	0.32
0.02	45.62	36.53	1.73	3.46	0.02	20.77	42.87	34.85	2.82
68.09	56.70	42.13	21.90	18.63	0.02	50.88	39.17	36.89	2.03
0.02	68.07	33.87	28.18	0.02	0.02	33.81	34.24	19.15	19.30
LMP	0.02	0.02	33.66	0.02	LMP	0.02	0.02	33.22	0.02
0.02	9.23	19.17	3.83	3.39	0.02	0.02	0.02	0.02	0.02
0.02	42.58	42.22	23.48	10.21	0.02	34.54	29.16	16.01	0.63
34.31	79.50	63.94	34.41	6.03	67.95	50.27	45.43	44.85	40.46
0.02	34.09	67.75	36.59	13.50	11.62	68.87	37.10	43.05	3.48
GMP	0.02	0.02	34.74	0.02	LMP	0.02	34.02	0.02	0.02

(a) PREMAP-DN

0.02	1.75	23.12	0.02	0.02	0.02	36.19	0.02	34.03	0.02
0.02	35.62	36.31	1.73	1.76	34.23	36.71	57.38	21.64	2.83
34.13	45.00	44.54	21.92	10.18	11.63	67.59	47.36	42.93	0.33
0.02	43.06	33.24	28.43	0.02	0.02	0.02	68.16	22.11	33.86
LMP	0.32	0.02	33.54	0.02	LMP	0.02	23.10	33.60	0.02
1.75	36.54	11.68	3.10	4.31	0.02	12.59	1.78	0.02	0.33
8.54	52.44	41.44	27.31	24.69	0.02	34.57	13.46	33.87	3.82
34.36	67.30	70.96	34.40	0.02	33.82	54.69	44.45	46.14	13.06
0.02	34.36	34.46	45.34	15.36	0.02	34.38	41.65	34.38	6.08
GMP	0.02	0.02	34.51	0.02	LMP	0.02	33.17	0.02	0.02

(b) LEC-DN

24.87	22.92	28.87	20.24	18.42	22.51	13.12	22.54	11.95	22.60
20.60	26.14	22.89	29.37	19.66	22.46	12.24	22.46	17.85	22.65
18.40	22.86	22.85	19.65	20.75	15.67	17.83	15.68	12.40	15.16
19.20	20.79	22.87	19.22	15.49	25.45	24.63	24.17	15.22	23.78
LMP	21.41	23.98	22.85	27.15	LMP	12.41	22.46	12.99	22.48
24.32	18.11	16.14	21.71	25.21	14.22	19.09	24.83	22.63	26.34
27.79	18.07	26.00	20.85	23.50	28.23	24.22	13.67	14.50	20.60
23.44	22.93	23.56	30.30	25.60	13.77	14.35	15.88	13.49	23.41
24.03	24.87	17.33	28.21	20.98	19.25	16.74	24.61	25.08	14.55
GMP	25.13	25.12	23.84	25.89	LMP	24.72	14.63	25.38	13.31

(c) L

4.19	10.53	21.40	19.46	26.49	12.40	15.39	13.30	23.06	23.19
21.19	20.13	19.69	20.09	18.34	18.31	12.89	23.22	25.55	11.62
21.19	8.57	24.03	17.75	14.64	15.14	22.27	16.69	22.67	22.65
23.98	23.17	23.16	45.58	36.16	23.85	11.99	23.08	12.89	22.48
LMP	15.07	24.21	45.15	28.48	LMP	23.22	15.87	18.33	23.85
38.73	29.03	19.69	15.57	16.73	24.02	38.02	23.70	22.95	32.42
40.81	36.77	20.26	15.31	25.06	28.12	31.28	29.86	23.77	24.76
29.46	30.39	23.36	17.19	25.20	18.45	13.28	5.56	20.16	16.00
24.52	19.61	23.78	15.03	23.34	24.19	13.65	8.79	15.56	9.81
GMP	22.42	19.97	13.71	14.77	LMP	12.92	9.30	10.95	5.20

(d) LC

Red rectangles: SPs consuming more than 50 mJ. Green rectangles: SPs consuming less than 5 mJ.

Figure 58 – Energy consumed per SP in **Scenario C** in the SystemC platform.

17.69	54.69	96.18	54.38	0.62	96.20	56.73	106.67	64.07	2.06
93.21	67.34	66.21	12.93	8.09	54.58	80.01	65.89	98.01	54.72
56.33	201.10	64.77	123.22	75.20	0.33	72.24	148.43	56.22	21.73
0.02	53.19	165.93	58.00	54.78	20.86	85.11	59.16	51.92	16.23
LMP	96.65	73.44	37.16	0.02	LMP	54.19	0.02	96.20	54.40
6.97	0.02	113.24	58.00	4.82	96.01	54.41	4.55	56.46	56.41
34.03	124.72	76.81	45.11	68.45	54.43	96.27	63.42	156.30	56.27
53.06	105.53	158.26	106.82	122.19	55.05	73.26	154.41	52.87	0.33
92.78	55.01	95.32	55.09	0.02	96.11	96.08	54.41	76.42	59.34
GMP	0.02	0.02	93.37	53.06	LMP	0.02	0.02	45.75	10.86

(a) PREMAP-DN

2.06	9.28	95.68	54.38	0.32	0.02	104.76	56.22	0.02	96.08
8.45	97.88	54.39	36.20	10.92	0.02	65.38	108.30	56.53	54.39
56.09	172.48	92.90	105.60	38.89	54.64	156.24	104.90	56.21	0.02
0.02	54.35	102.38	53.45	0.02	8.45	62.98	11.96	97.83	54.40
LMP	8.45	1.75	113.63	58.03	LMP	121.93	59.80	54.44	96.07
53.19	0.02	104.64	56.23	36.59	56.18	1.75	71.70	98.45	57.97
93.35	116.21	59.65	0.32	55.88	104.54	53.13	173.35	78.75	66.44
52.85	120.49	90.27	107.44	136.79	0.02	93.34	104.79	56.19	7.05
0.02	42.88	95.57	54.40	37.69	95.96	63.94	132.90	148.63	82.23
GMP	1.75	93.17	53.05	1.78	LMP	0.33	54.85	96.65	53.10

(b) LEC-DN

30.05	85.39	44.85	40.12	96.51	103.41	91.96	98.11	45.37	90.20
43.86	102.57	44.37	34.06	55.66	34.28	94.85	47.71	44.22	47.53
42.91	74.19	45.06	57.84	54.22	46.79	85.85	42.24	31.18	31.49
42.43	98.36	57.94	84.17	44.88	44.09	32.96	99.24	60.84	60.97
LMP	58.32	128.66	99.56	44.80	LMP	79.10	43.82	34.83	46.48
42.34	47.49	34.82	56.49	56.00	16.54	99.76	44.97	56.35	27.99
58.45	29.69	58.48	86.74	43.37	45.83	42.33	44.55	85.82	100.75
28.94	44.07	88.20	91.55	57.01	47.14	54.59	44.46	88.71	44.57
133.71	98.74	56.73	99.44	88.61	29.09	98.43	43.18	56.73	56.87
GMP	57.17	84.16	30.91	46.02	LMP	138.74	95.25	83.34	54.95

(c) L

39.48	142.04	62.92	102.78	98.79	134.23	64.08	36.23	95.73	96.29
90.86	62.14	57.85	62.26	63.09	55.00	63.18	57.54	42.53	43.15
28.26	30.26	40.91	40.38	41.63	32.65	35.73	94.69	47.91	17.82
39.36	42.58	86.36	44.91	86.15	12.09	45.03	90.32	59.32	33.28
LMP	85.50	26.32	55.16	79.74	LMP	43.72	82.81	61.29	89.79
77.26	56.62	88.03	107.12	50.85	99.85	104.83	28.20	39.53	78.04
36.68	43.50	44.93	56.74	102.25	56.63	56.74	85.42	54.68	44.89
24.01	23.63	53.95	97.71	98.43	67.74	37.45	43.66	48.10	78.99
107.12	53.82	48.53	57.33	63.36	37.07	38.23	39.09	58.27	72.47
GMP	94.70	87.08	26.70	17.82	LMP	43.18	136.57	45.15	109.85

(d) LC

Red rectangles: SPs consuming more than 100 mJ. Green rectangles: SPs consuming less than 10 mJ.

Figure 59 – Energy consumed per SP in **Scenario D** in the SystemC platform.

34.51	45.93	30.71	22.49	3.39	35.30	34.07	0.02	8.55	18.83
68.28	65.23	33.21	95.89	54.38	34.15	53.15	96.03	54.44	59.79
47.15	110.98	93.39	53.14	4.31	0.02	93.39	93.26	53.18	104.86
8.11	97.41	109.10	52.76	13.32	96.03	54.45	63.13	61.77	54.03
LMP	0.32	88.68	35.32	33.84	LMP	0.02	82.29	54.97	0.02
0.02	95.74	54.43	0.02	0.02	2.78	36.58	95.91	54.45	0.02
26.16	106.15	70.10	92.85	53.19	11.52	95.99	43.89	42.71	56.17
87.51	89.66	38.16	104.64	56.42	33.84	89.47	60.25	69.27	78.93
33.81	29.41	130.67	63.13	1.82	96.12	54.40	155.99	104.82	56.25
GMP	34.05	8.42	93.16	53.09	LMP	0.02	53.39	62.47	1.74

(a) PREMAP-DN

0.02	55.40	19.62	10.75	3.45	34.99	34.05	0.02	33.29	35.21
87.93	92.54	54.01	95.89	54.37	33.55	53.15	96.03	54.44	33.37
93.06	86.67	93.39	53.14	96.44	0.02	93.39	93.26	53.18	0.32
53.34	156.20	83.57	11.63	54.38	96.03	54.45	17.91	62.05	54.05
LMP	45.24	36.71	95.69	54.41	LMP	0.02	9.66	43.89	11.99
0.02	95.74	54.43	0.02	0.02	33.78	47.87	96.14	54.44	2.80
53.40	104.44	68.43	92.89	53.19	35.81	96.04	63.88	42.16	45.52
86.89	90.03	114.60	105.61	56.08	34.27	62.57	55.23	61.64	31.71
33.91	29.54	46.13	46.91	2.12	96.09	54.45	72.70	104.82	56.25
GMP	34.18	0.02	93.14	53.07	LMP	1.74	65.94	61.49	1.76

(b) LEC-DN

46.15	78.11	47.86	39.45	49.27	40.71	30.92	103.89	29.63	33.06
64.98	79.43	47.29	39.98	49.42	47.78	50.95	50.45	42.69	86.13
49.17	92.65	47.39	83.13	50.67	43.54	94.73	47.94	30.71	30.50
85.03	30.97	100.14	91.35	49.35	95.06	43.99	47.91	31.36	43.37
LMP	47.82	52.68	91.92	67.10	LMP	47.91	102.02	29.69	88.00
80.99	47.69	31.62	96.97	98.82	47.82	22.57	33.74	91.30	36.46
34.13	35.13	54.80	33.22	43.23	27.98	35.90	22.45	49.91	33.38
97.60	47.98	54.85	100.35	43.70	78.35	22.46	22.74	36.41	49.68
51.17	31.91	96.62	86.60	43.80	80.16	51.44	22.46	23.57	31.80
GMP	34.32	31.13	36.70	93.86	LMP	33.35	29.33	22.98	23.32

(c) L

135.87	54.02	86.76	55.29	139.06	90.74	62.67	76.32	49.47	91.44
55.04	63.00	43.18	61.75	54.82	31.84	59.76	58.91	54.67	47.51
20.85	48.14	46.16	36.16	27.72	43.49	47.90	85.18	40.31	24.51
73.18	30.16	20.06	40.80	42.28	76.11	46.93	23.09	49.50	42.01
LMP	31.02	22.17	46.98	86.89	LMP	29.27	17.86	37.17	103.37
23.65	46.07	97.45	85.59	25.57	36.69	23.20	23.20	48.20	83.88
37.18	90.86	55.77	54.45	37.86	23.30	26.66	26.64	32.27	49.09
93.15	25.46	59.86	37.42	96.15	26.80	23.26	23.58	54.98	67.38
44.64	48.23	60.17	54.99	55.34	30.78	29.51	49.31	41.35	96.86
GMP	24.37	84.64	42.41	134.93	LMP	83.40	48.47	74.22	44.84

(d) LC

Red rectangles: SPs consuming more than 100 mJ. Green rectangles: SPs consuming less than 10 mJ.

Figure 60 – Energy consumed per SP in **Scenario E** in the SystemC platform.

21.93	34.20	33.64	46.18	28.38	21.60	34.46	34.03	34.57	28.58
33.95	34.59	62.57	62.44	33.98	34.21	34.56	60.30	28.58	33.53
45.73	62.29	62.00	67.83	34.27	34.55	62.60	62.08	67.84	34.25
62.99	62.37	57.35	35.53	34.32	34.01	62.93	70.24	35.06	33.55
LMP	68.22	34.19	34.01	28.51	LMP	34.11	34.01	33.69	28.62
21.94	34.36	34.34	46.88	28.03	22.16	34.24	33.80	46.28	29.16
34.44	35.04	62.78	62.51	34.36	34.29	34.58	62.50	62.44	33.22
46.45	63.07	62.16	68.97	34.64	45.99	61.92	61.90	67.67	34.08
62.07	62.34	71.17	70.05	68.07	72.00	63.50	57.20	35.35	33.91
GMP	68.59	34.73	60.50	28.05	LMP	68.80	33.83	33.79	28.56

(a) PREMAP-DN

21.95	34.20	33.64	46.18	28.34	21.63	34.40	34.01	46.24	28.37
33.95	34.59	63.23	61.75	33.78	34.21	34.47	62.61	61.76	33.61
46.01	62.39	62.76	67.86	34.26	34.62	62.64	62.04	67.56	34.42
62.51	62.76	57.36	35.49	34.34	34.09	28.61	57.39	35.08	33.55
LMP	68.11	34.19	34.00	28.48	LMP	34.18	34.25	33.69	28.55
21.98	34.39	34.35	48.61	28.04	22.05	34.20	33.81	46.29	29.39
34.55	35.16	62.61	62.40	34.35	34.17	34.57	63.04	62.65	33.32
46.31	62.89	62.15	69.13	34.65	45.69	61.99	62.26	68.17	34.08
62.07	62.31	70.86	69.68	68.09	61.69	62.57	57.17	35.36	33.88
GMP	68.56	34.70	60.27	28.23	LMP	67.61	33.90	33.77	28.68

(b) LEC-DN

46.90	45.43	38.29	45.67	45.59	47.51	45.26	38.29	45.91	45.61
45.55	47.73	45.65	50.92	45.69	45.72	47.86	45.84	39.77	45.56
41.94	45.39	38.44	38.00	38.15	41.99	45.42	38.59	37.91	38.18
47.62	48.14	47.31	38.49	50.19	47.09	47.73	47.25	49.56	50.69
LMP	45.53	58.89	38.44	47.71	LMP	45.58	59.29	38.41	47.41
49.53	47.72	39.21	46.49	47.45	45.61	45.35	49.60	45.92	45.73
48.16	38.78	46.00	40.45	46.20	38.27	47.29	45.85	39.40	45.60
50.91	47.12	35.71	48.20	50.60	34.67	45.38	38.56	38.10	49.80
48.11	48.35	47.98	48.86	48.09	47.24	48.03	47.50	49.63	47.16
GMP	46.06	49.10	40.01	50.86	LMP	45.62	47.81	48.07	47.80

(c) L

46.88	38.18	38.19	47.46	36.84	35.48	38.09	37.67	47.08	48.08
40.36	46.21	45.83	58.92	46.09	38.68	34.46	46.17	60.71	46.17
38.25	45.59	33.93	45.83	57.00	38.20	45.56	34.09	45.73	45.33
47.44	46.21	45.77	45.76	45.55	59.17	46.07	45.79	45.71	45.75
LMP	46.15	44.24	38.16	58.88	LMP	56.99	44.60	38.16	59.06
35.79	54.47	33.41	47.70	36.57	35.35	38.25	37.66	46.90	48.18
38.92	46.08	47.94	48.43	34.64	38.11	34.44	46.14	60.94	46.26
40.90	46.25	51.90	46.07	46.16	38.22	45.60	35.77	45.59	45.49
47.42	41.14	58.79	37.83	57.03	58.73	46.18	45.77	45.58	45.84
GMP	46.31	52.81	50.51	62.49	LMP	57.22	44.95	38.13	59.19

(d) LC

Red rectangles: SPs consuming more than 60 mJ. Green rectangles: SPs consuming less than 30 mJ.

Figure 61 – Energy consumed per SP in **Scenario F** in the SystemC platform.

APPENDIX B –TEMPERATURE DISTRIBUTION

This appendix complements the results of Section 5.5.2.4, providing the temperature distribution at the end of execution time for scenarios B to F of Table 16. Such thermal maps show that L and LC heuristics reduce hotspots in most scenarios.

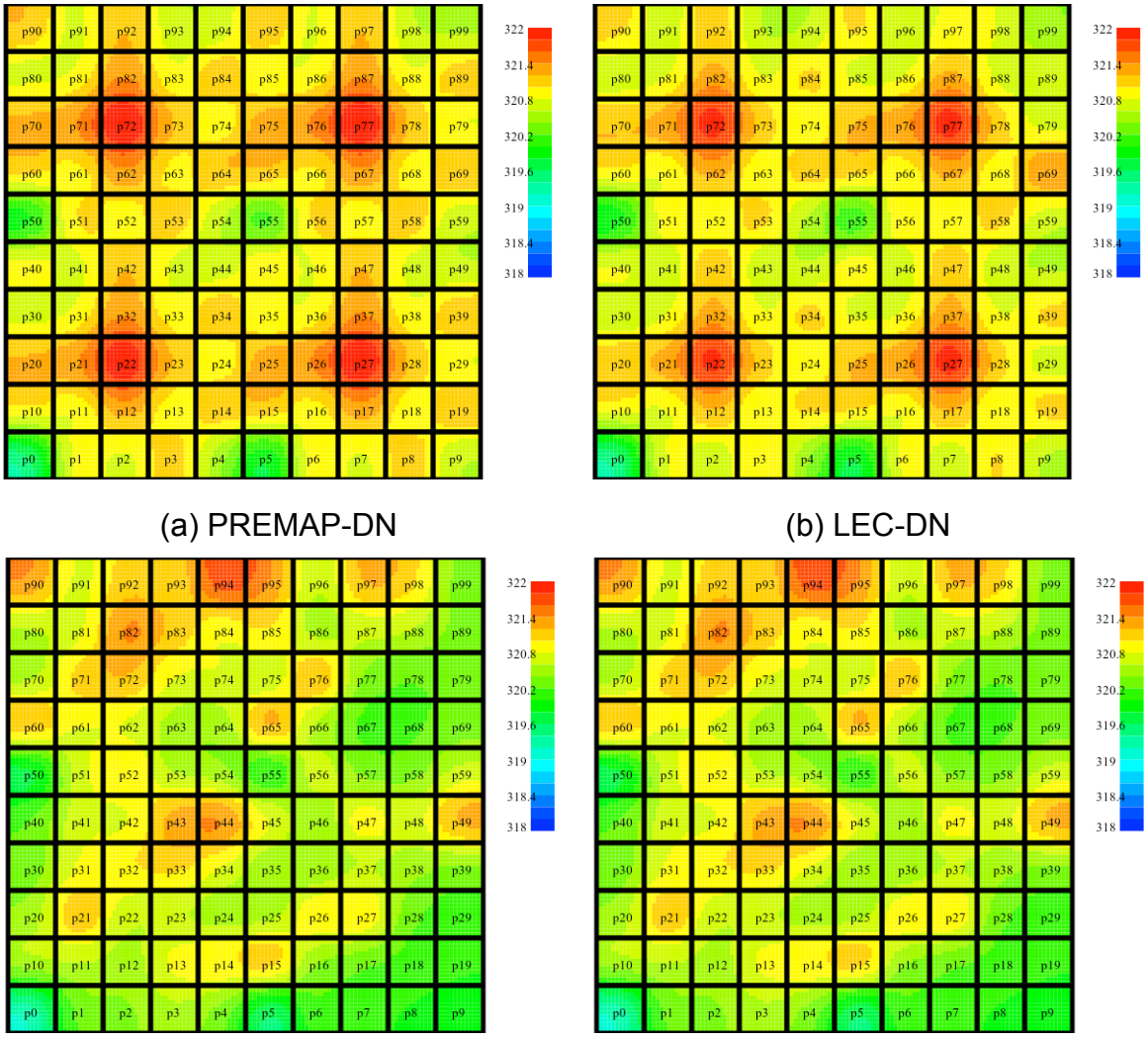
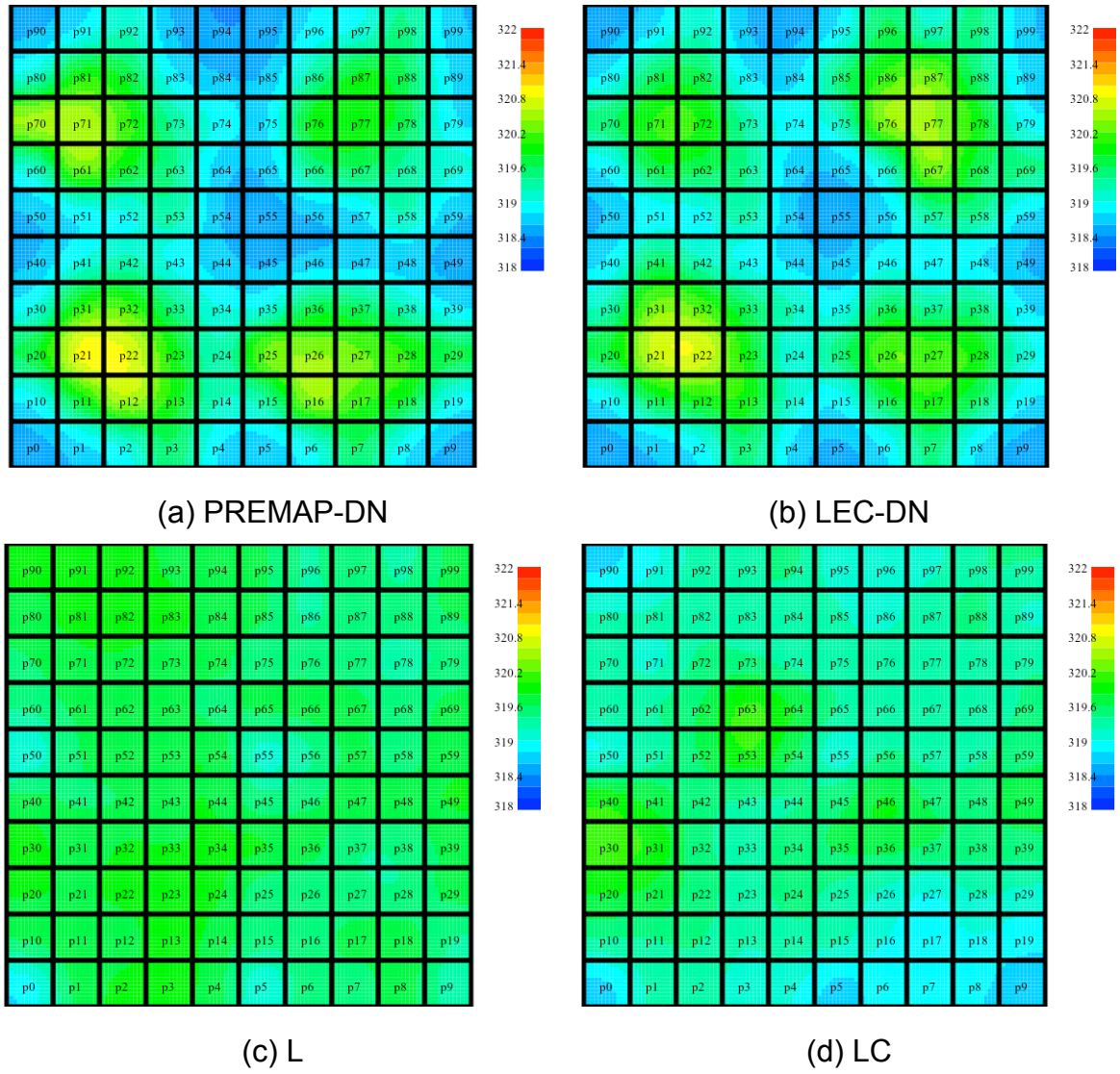


Figure 62 – Temperature distribution for Scenario B.



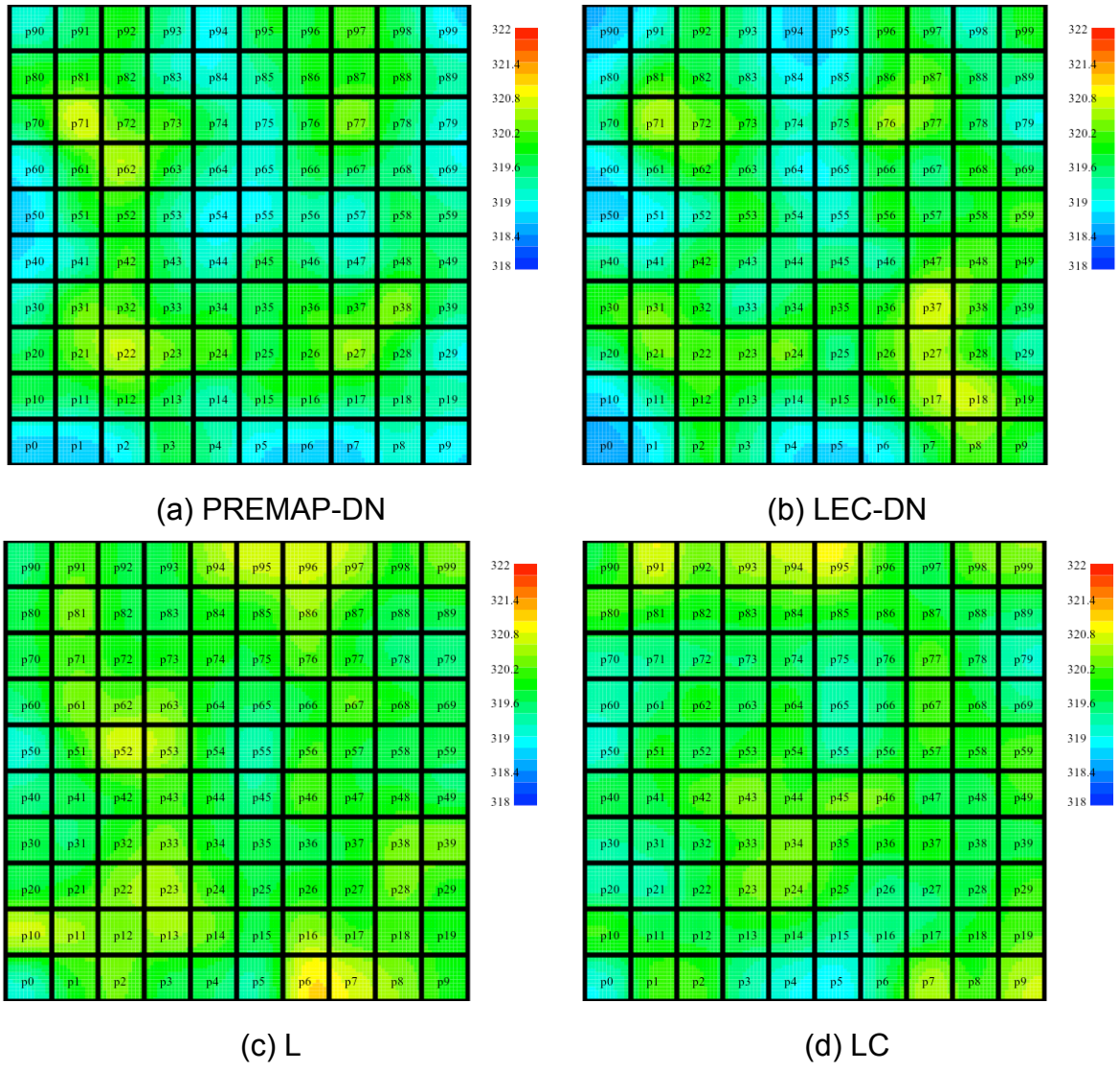
(a) PREMAP-DN

(b) LEC-DN

(c) L

(d) LC

Figure 63 – Temperature distribution for Scenario C.



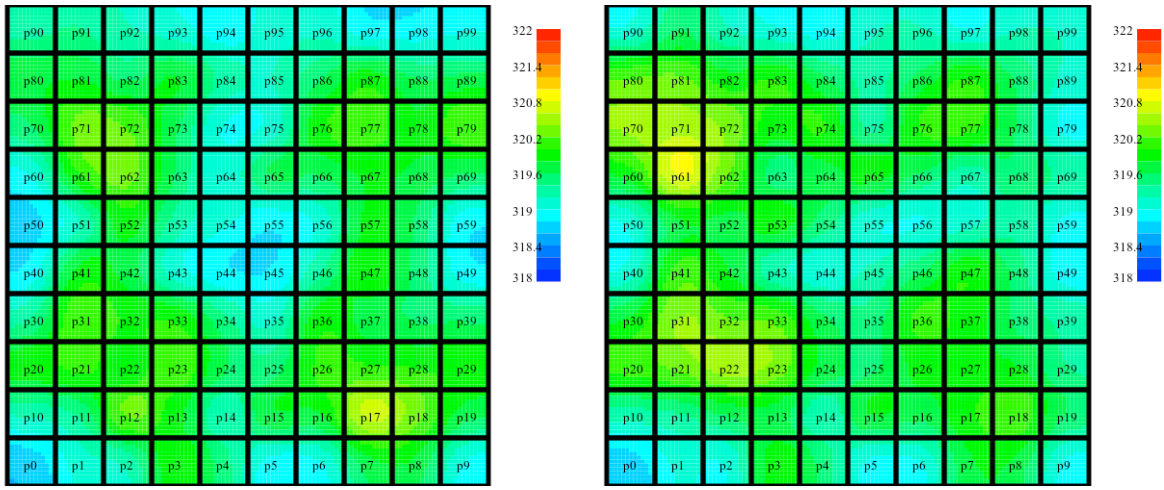
(a) PREMAP-DN

(b) LEC-DN

(c) L

(d) LC

Figure 64 – Temperature distribution for Scenario D.



(a) PREMAP-DN

(b) LEC-DN

(c) L

(d) LC

Figure 65 – Temperature distribution for Scenario E.

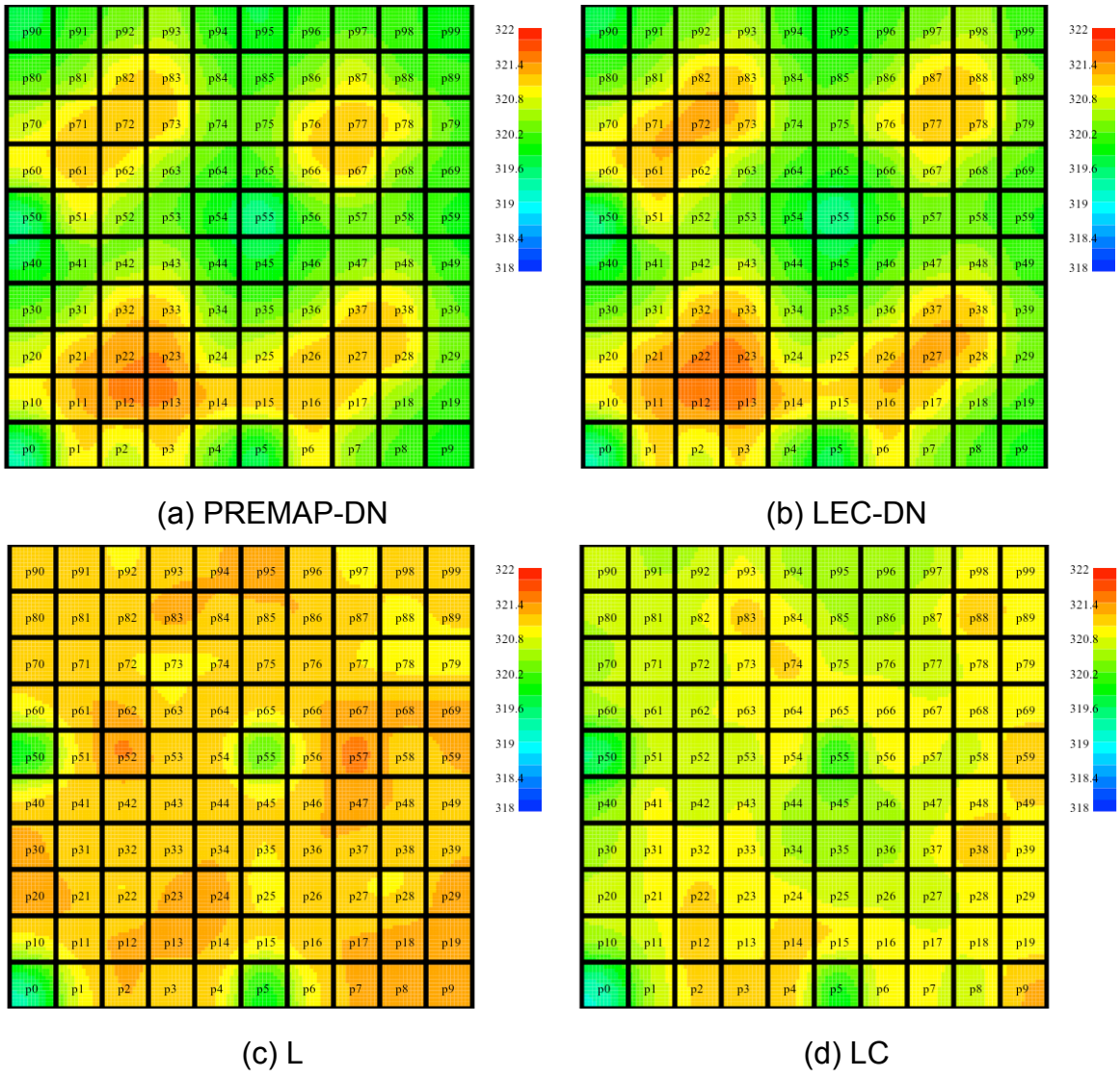
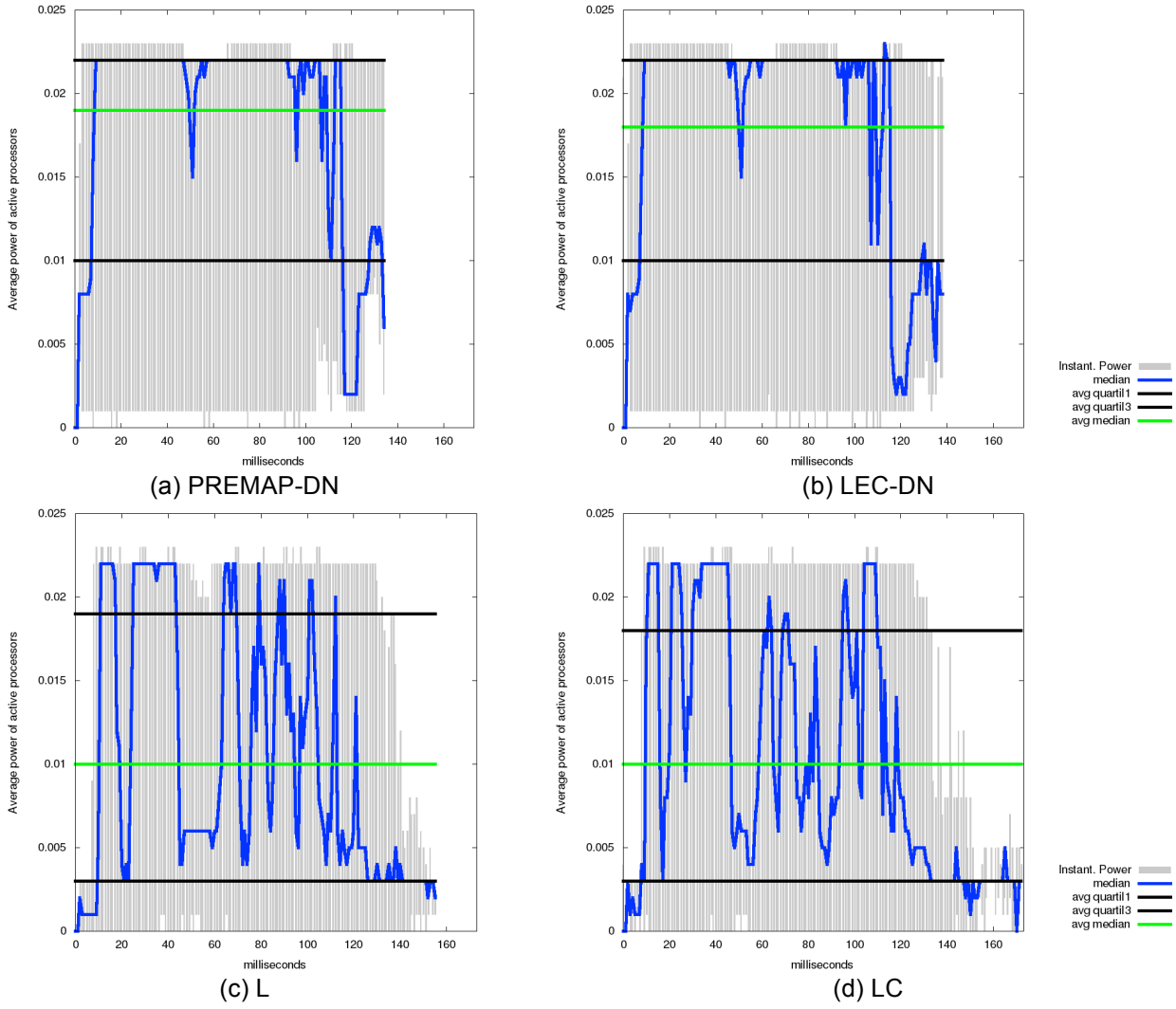


Figure 66 – Temperature distribution for Scenario F.

APPENDIX C – POWER TRACES

This appendix complements the results of Section 5.5.2.5, with the power traces for scenarios B to F of Table 16. Each graph presents the instantaneous SPs (slave PEs) power dissipation in the blue curves (median value). The X-axis corresponds to the execution time in milliseconds (only PEs executing tasks are considered) and the Y-axis the average power of active processors (W). Gray bars: 50% of the population, first to third quartiles. Black lines: average first and third quartiles. Green line: average median. Blue line: instantaneous median.

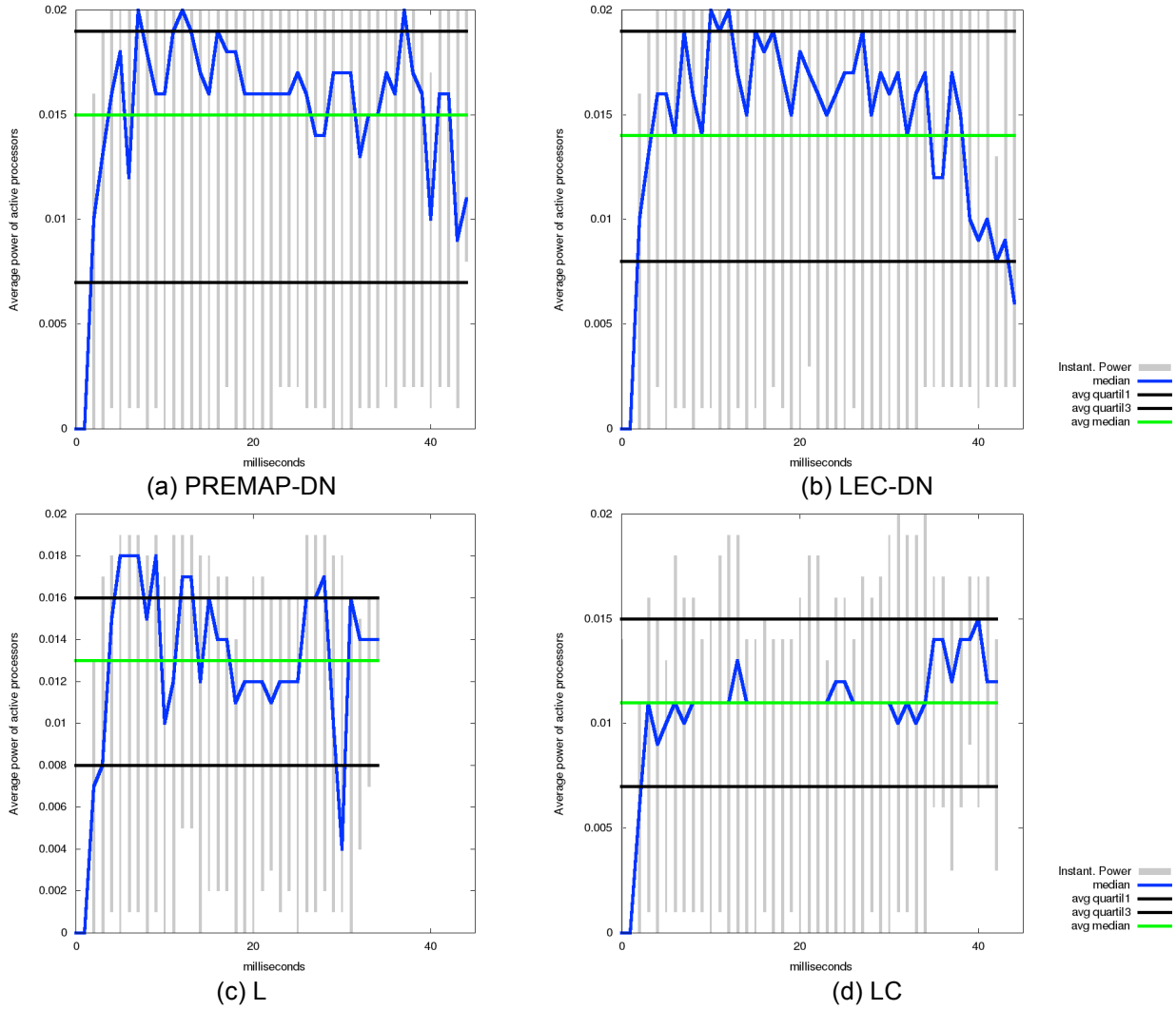
Such graphs show the better power distribution of the L and LC heuristics during the execution time, compared to PREMAP-DN and LEC-DN heuristics.



(a) PREMAP-DN (b) LEC-DN

(c) L (d) LC

Figure 67 – Power traces for scenario B.



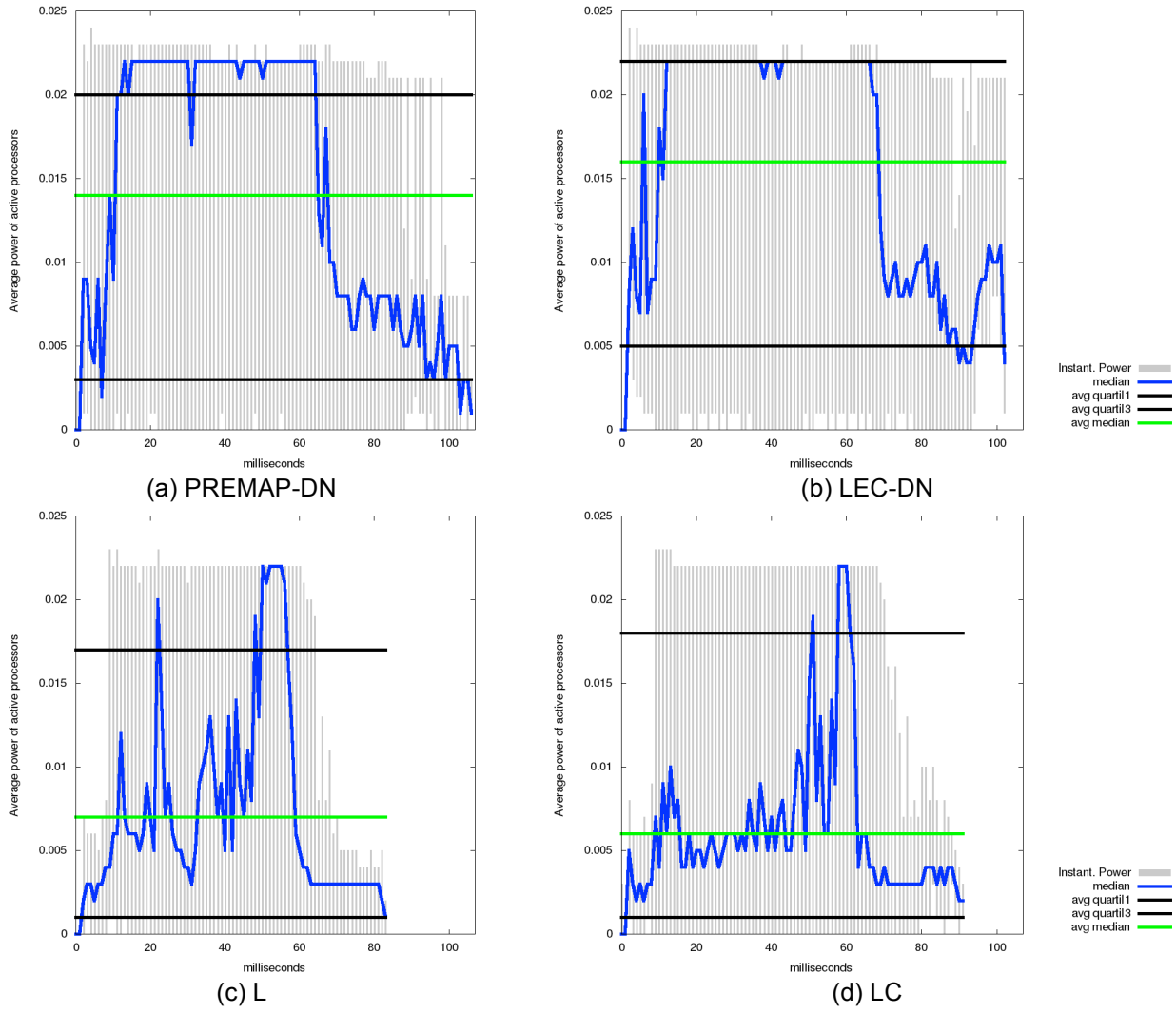
(a) PREMAP-DN

(b) LEC-DN

(c) L

(d) LC

Figure 68 – Power traces for scenario C.



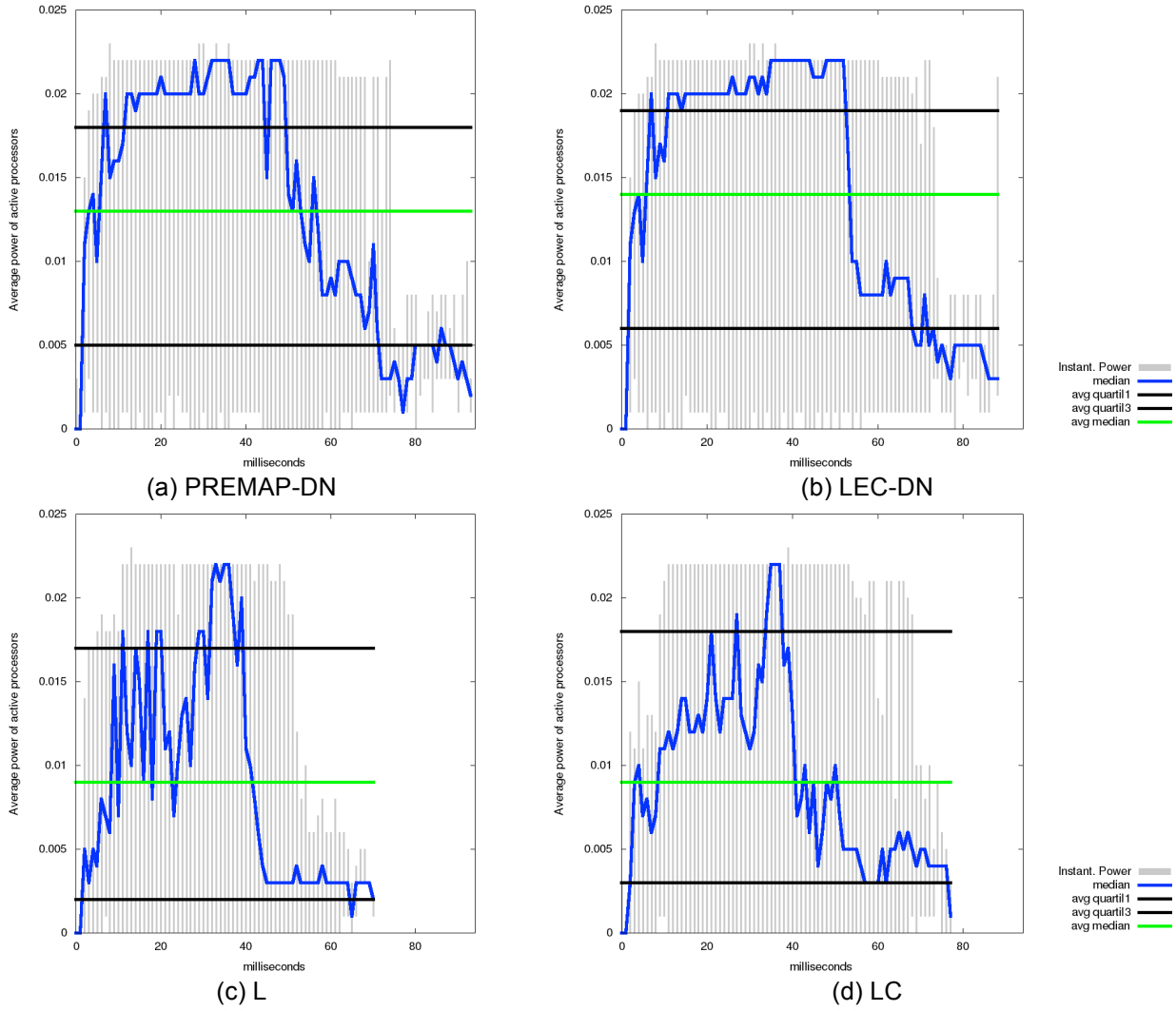
(a) PREMAP-DN

(b) LEC-DN

(c) L

(d) LC

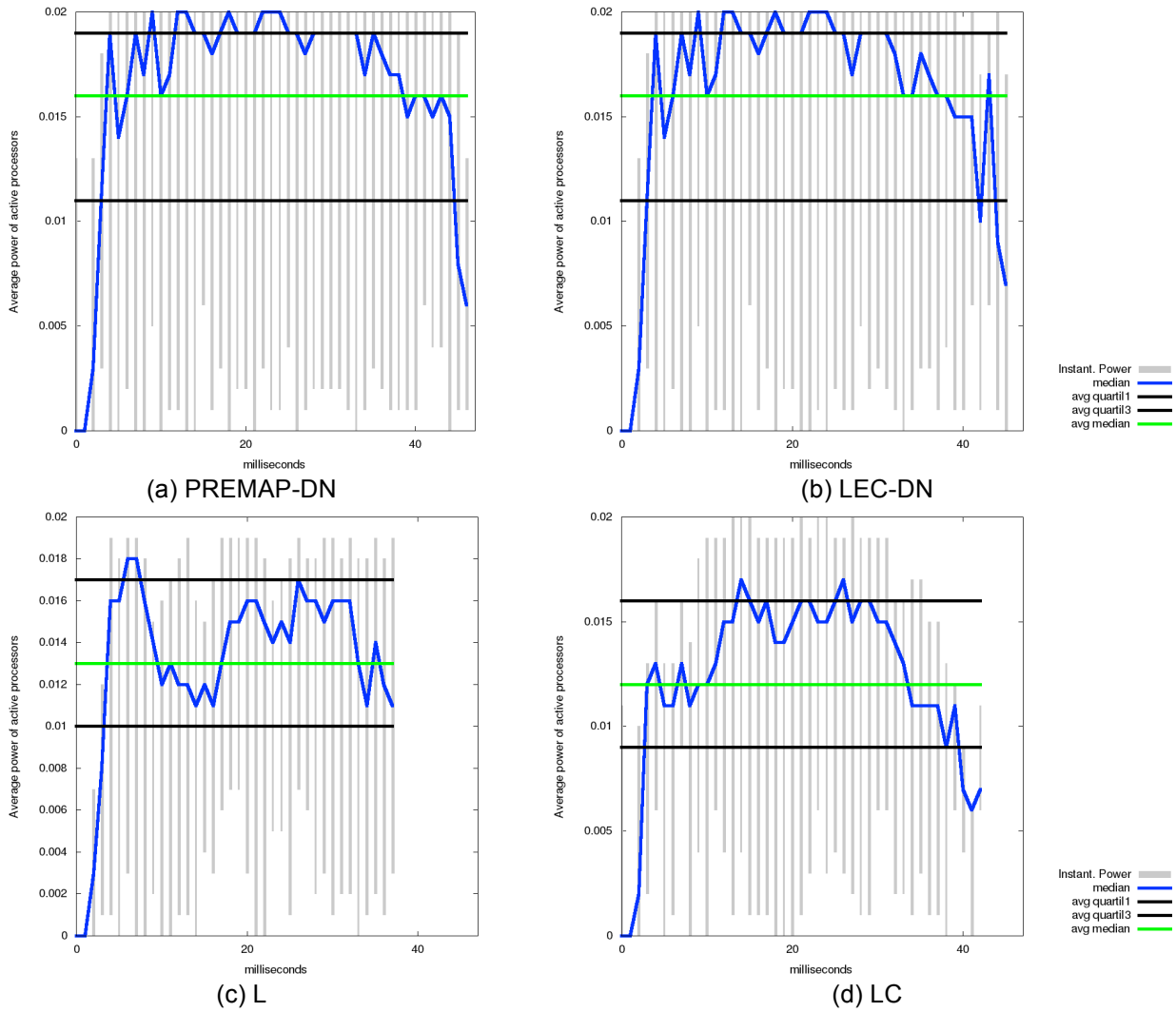
Figure 69 – Power traces for scenario D.



(a) PREMAP-DN (b) LEC-DN

(c) L (d) LC

Figure 70 – Power traces for scenario E.



(a) PREMAP-DN (b) LEC-DN

(c) L (d) LC

Figure 71 – Power traces for scenario F.

APPENDIX D – PUBLICATIONS OF THE AUTHOR

Table 34 presents the set of publications of the author. The publications 1, 2, and 3 were written during the Master's degree. Other publications were developed during the PhD studies period. The description column links the paper to this work, when applicable, or to the main theme of the publication.

Table 34 – Publications during the PhD period.

	Publication	Description
1	**Energy-aware dynamic task mapping for NoC-based MPSoCs** MANDELLI, M. G.; OST, L. C.; CARARA, E. A.; GUINDANI, G. M.; ROSA, T.; MEDEIROS, G.; MORAES, F. G. In: ISCAS, 2011 [MAN11a]	Reference mapping heuristic used in Chapter 5
2	**Multi-Task Dynamic Mapping onto NoC-based MPSoCs** MANDELLI, M. G.; OST, L. C.; AMORY, A. M.; MORAES, F. G. In: SBCCI, 2011 [MAN11b]	Reference mapping heuristic used in Chapter 5
3	**Exploring dynamic mapping impact on NoC-based MPSoCs performance using a model-based framework** OST, L. C.; MANDELLI, M. G.; ALMEIDA, G. M.; INDRUSIAK, L. S.; MOLLER, L. S.; GLESNER, M.; SASSATELLI, G.; ROBERT, M.; MORAES, F. G. In: SBCCI, 2011 [OST11a]	Integration of the task mapping heuristics proposed in publications 1 and 2 in a high-level MPSoC model [OST13]. Chapter 3.
4	**Model-based design flow for NoC-based MPSoCs** OST, L. C.; INDRUSIAK, L. S.; MAATTA, S.; MANDELLI, M. G.; NURMI, J.; MORAES, F. G. In: ICECS, 2010 [OST10]	
5	**Exploring Heterogeneous NoC-based MPSoCs: from FPGA to High-Level Modeling** OST, L. C.; ALMEIDA, G. M.; MANDELLI, M. G.; WACHTER, E.; VARYANI, S.; INDRUSIAK, L. S.; SASSATELLI, G.; ROBERT, M.; MORAES, F. G. In: RECOSOC, 2011 [OST11b]	
6	**Exploring Adaptive Techniques in Heterogeneous MPSoCs based on Virtualization** OST, L. C.; VARYANI, S.; MANDELLI, M. G.; WACHTER, E.; ALMEIDA, G. M.; INDRUSIAK, L. S.; SASSATELLI, G.; MORAES, F. G. In: ACM Transactions on Reconfigurable Technology and Systems, vol. 5(3), pp. 1 - 11, 2012. [OST12]	
7	**Power-aware dynamic mapping heuristics for NoC-based MPSoCs using a unified model-based approach** OST, L. C.; MANDELLI, M. G.; ALMEIDA, G. M.; MOLLER, L. S.; INDRUSIAK, L. S.; SASSATELLI, G.; BENOIT, P.; GLESNER, M.; ROBERT, M.; MORAES, F. G. In: ACM Transactions on Embedded Computing Systems, vol. 12(3), pp. 1 - 22, 2013 [OST13]	
8	**Enhancing Performance of MPSoCs through Distributed Resource Management** MANDELLI, M.; CASTILHOS, G. M.; MORAES, F In: ICECS, 2012 [MAN12]	Chapter 4
9	**Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes** CASTILHOS, G. M.; MANDELLI, M.; MADALOZZO, G. A.; MORAES, F In: ISVLSI 2013 [CAS13]	Chapter 4
10	**MPSoC Modeling for Reducing Software Development** MANDELLI, M.; ROSA, F.; OST, L.; SASSATELLI, G.; MORAES, F. G. In: ICECS, 2013 [MAN13]	Chapter 3
11	**Trading-off system load and communication in mapping heuristics for improving NoC-based MPSoCs reliability** Mandelli, M.; Ost, L.; Sassatelli, G.; Moraes, F. In: ISQED, 2015 [MAN15]	Chapter 5