PUCRS

ESCOLA POLITÉCNICA
PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ LUÍS DEL MESTRE MARTINS

MULTI-OBJECTIVE RESOURCE MANAGEMENT FOR MANY-CORE SYSTEMS

Porto Alegre
2018

PÓS-GRADUAÇÃO - *STRICTO SENSU*

Pontifícia Universidade Católica
do Rio Grande do Sul

PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
FACULTY OF INFORMATICS
COMPUTER SCIENCE GRADUATE PROGRAM

MULTI-OBJECTIVE RESOURCE MANAGEMENT FOR MANY-CORE SYSTEMS

ANDRE LUIS DEL MESTRE MARTINS

Dissertation submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fullfillment of the requirements for the degree of Ph. D. in Computer Science.

Advisor: Prof. Fernando Gehm Moraes

Porto Alegre
2018

Ficha Catalográfica

M386m Martins, André Luís Del Mestre

Multi-Objective Resource Management for Many-Core Systems / André Luís Del Mestre Martins . – 2018.
147.
Tese (Doutorado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Fernando Gehm Moraes.

1. Many-core. 2. Resource Management. 3. Energy Optimization. 4. DVFS. 5. Multi-objective Optimization. I. Moraes, Fernando Gehm. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

André Luís Del Mestre Martins

Multi-Objective Resource Management for Many-core Systems

Tese apresentada como requisito parcial para obtenção do grau de Doutor em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação, Escola Politécnica da Pontifícia Universidade Católica do Rio Grande do Sul.

Aprovado em 19 de março de 2018.

BANCA EXAMINADORA:

Prof. Dr. Everton Carara -Avaliador (UFSM)

Prof. Dr. Sergio Bampi - Avaliador (UFRGS)

Prof. Dr. Ney Calazans - Avaliador (PUCRS)

Prof. Dr. Fernando Gehm Moraes (PPGCC/PUCRS - Orientador)

ACKNOWLEDGMENTS

Gostaria de agradecer ao Professor Fernando Moraes pela orientação e dedicação durante o desenvolvimento desta Tese. Ainda, agradeço por ter me aceitado como seu aluno sem me conhecer e pela compreensão durante o período anterior ao afastamento.

Agradeço ao Prof. Nikil Dutt e ao Prof. Amir Rahmani por me acolherem e me orietarem no período sanduíche nos Estados Unidos. Agradeço também ao pessoal do DRG - Dutt Research Group - pelo apoio técnico e companheirismo.

Agradeço aos colegas do GAPH pelo apoio técnico. Nominalmente, agradeço ao Marcelo Ruaro, por manter a HeMPS atualizada e dar o necessário suporte no desenvolvimento. Ainda, agradeço ao Guilherme Heck e Leandro Heck por manterem a infraestrutura do laboratório sempre funcionando. Obrigado ao Luciano Caimi por organizar todos os eventos. Finalmente, agradeço aos que colaboraram tecnicamente de forma direta para desenvolvimento do meu trabalho: Anderson Sant'Ana e Douglas Silva.

Agradeço à minha família e amigos que me apoiam desde sempre e torcem por mim. No doutorado os amigos que participaram de maneira decisiva no desenvolvimento da Tese são: Tiago Paes, Bruno Zatt e família, Lisa Lee, e Thao Turong. Além destes, agradeço minha namorada Kellin Gauze.

Agradeço ao IFSul por me proporcionar a oportunidade de me dedicar exclusivamente ao Doutorado por 3 anos.

GERENCIAMENTO DE RECURSOS MULTI-OBJETIVO PARA SISTEMAS MANY-CORE

RESUMO

Sistemas *many-core* integram múltiplos *cores* em um chip, fornecendo alto desempenho para vários segmentos de mercado. Novas tecnologias introduzem restrições de potencia conhecidos como *utilization-wall* ou *dark-silicon*, onde a dissipação de potência no chip impede que todos os PEs sejam utilizados simultaneamente em máximo desempenho. A carga de trabalho (*workload*) em sistemas many-core inclui aplicações tempo real (RT), com restrições de vazão e temporização. Além disso, *workloads* típicos geram vales e picos de utilização de recursos ao longo do tempo. Este cenário, sistemas complexos de alto desempenho sujeitos a restrições de potência e utilização, exigem um gerenciamento de recursos (RM) multi-objetivos capaz de adaptar dinamicamente os objetivos do sistema, respeitando as restrições impostas. Os trabalhos relacionados que tratam aplicações RT aplicam uma análise em tempo de projeto com o *workload* esperado, para atender às restrições de vazão e temporização. Para abordar esta limitação do estado-da-arte, decisões em tempo de projeto, esta Tese propõe um gerenciamento hierárquico de energia (REM), sendo o primeiro trabalho que considera a execução de aplicações RT e gerência de recursos sujeitos a restrições de potência, sem uma análise prévia do conjunto de aplicações. REM emprega diferentes heurísticas de mapeamento e de DVFS para reduzir o consumo de energia. Além de não incluir as aplicações RT, os trabalhos relacionados não consideram um *workload* dinâmico, propondo RMs com um único objetivo a otimizar. Para tratar esta segunda limitação do estado-da-arte, RMs com objetivo único a otimizar, esta Tese apresenta um gerenciamento de recursos multi-objetivos adaptativo e hierárquico (MORM) para sistemas many-core com restrições de potência, considerando *workloads* dinâmicos com picos e vales de utilização. MORM pode mudar dinamicamente os objetivos, priorizando energia ou desempenho, de acordo com o comportamento do *workload*. Ambos RMs (REM e MORM) são abordagens multi-objetivos. Esta Tese emprega o paradigma Observar-Decidir-Atuar (ODA) como método de projeto para implementar REM e MORM. A *Observação* consiste em caracterizar os *cores* e integrar monitores de hardware para fornecer informações precisas e rápidas relacionadas à energia. A *Atuação* configura os atuadores do sistema em tempo de execução para permitir que os RMs atendam às decisões multi-objetivos. A *Decisão* corresponde à implementação do REM e do MORM, os quais compartilham os métodos de Observação e Atuação. REM e MORM destacam-se dos trabalhos relacionados devido às suas características de escalabilidade, abrangência e estimativa de potência e energia precisas. As avaliações utilizando REM em many-cores com até 144 *cores* reduzem o consumo de energia entre 15% e 28%, mantendo as violações de temporização abaixo de 2,5%. Resultados mostram que MORM pode atender dinamicamente a objetivos distintos. Comparado MORM com um RM estado-da-arte, MORM otimiza o desempenho em vales de *workload* em 11,56% e em picos *workload* em até 49%.

Palavras-Chave: Many-core, gerência de recursos, otimização de energia, DVFS, multi-objetivo.

MULTI-OBJECTIVE RESOURCE MANAGEMENT FOR MANY-CORE SYSTEMS

ABSTRACT

Many-core systems integrate several cores in a single die to provide high-performance computing in multiple market segments. The newest technology nodes introduce restricted power caps so that results in the utilization-wall (also known as *dark silicon*), i.e., the on-chip power dissipation prevents the use of all resources at full performance simultaneously. The workload of many-core systems includes real-time (RT) applications, which bring the application throughput as another constraint to meet. Also, dynamic workloads generate valleys and peaks of resources utilization over the time. This scenario, complex high-performance systems subject to power and performance constraints, creates the need for multi-objective resource management (RM) able to dynamically adapt the system goals while respecting the constraints. Concerning RT applications, related works apply a design-time analysis of the expected workload to ensure throughput constraints. To cover this limitation, design-time decisions, this Thesis proposes a hierarchical Runtime Energy Management (REM) for RT applications as the first work to link the execution of RT applications and RM under a power cap without design-time analysis of the application set. REM employs different mapping and DVFS (Dynamic Voltage-Frequency Scaling) heuristics for RT and non-RT tasks to save energy. Besides not considering RT applications, related works do not consider the workload variation and propose single-objective RMs. To tackle this second limitation, single-objective RMs, this Thesis presents a hierarchical adaptive multi-objective resource management (MORM) for many-core systems under a power cap. MORM addresses dynamic workloads with peaks and valleys of resources utilization. MORM can dynamically shift the goals to prioritize energy or performance according to the workload behavior. Both RMs (REM and MORM), are multi-objective approaches. This Thesis employs the Observe-Decide-Act (ODA) paradigm as the design methodology to implement REM and MORM. The *Observing* consists on characterizing the cores and on integrating hardware monitors to provide accurate and fast power-related information for an efficient RM. The *Actuation* configures the system actuators at runtime to enable the RMs to follow the multi-objective decisions. The *Decision* corresponds to REM and MORM, which share the *Observing* and *Actuation* infrastructure. REM and MORM stand out from related works regarding scalability, comprehensiveness, and accurate power and energy estimation. Concerning REM, evaluations on many-core systems up to 144 cores show energy savings from 15% to 28% while keeping timing violations below 2.5%. Regarding MORM, results show it can drive applications to dynamically follow distinct objectives. Compared to a *state-of-the-art* RM targeting performance, MORM speeds up the workload valley by 11.56% and the workload peak by up to 49%.

Keywords: Many-core, resource management, energy optimization, DVFS, multi-objective.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AA – Application Admission

ABUC – Applications Buffer Utilization Calculator

AES – Advanced Encryption Standard

AIRC – Application Injection Rate Calculator

ALU – Arithmetic Logic Unit

APC – Application Power Calculator

APUC – Application Processor Utilization Calculator

ARM – Advanced RISC Machines

BE – Best Effort

CG – Clock-Gating

CMP – Cluster Manager Processor

CM – Cluster Manager

CPI – Cycles per Instruction

DMNI – Direct Memory Network Interface

DOP – Degree of Parallelism

DSAPM – Dark Silicon-aware Power Management

DTW – Dynamic Time Warping

DVFS – Dynamic Voltage and Frequency Scaling

EDP – Energy-delay Product

FIFO – First In, First Out

FPGA – Field-Programmable Gate Array

GM – Global Manager

HI – High Injection

HPC – High Performance Computing

HPM – Hierarchical Power Management

ILP – Instruction Level Parallelism

LTI – Long Time Interval

LU – Low Utilization

MORM – Multi-Objective Resource Management

MPEG – Moving Picture Experts Group

MPI – Message Passing Interface

NOC – Network on Chip

NVT – Near-threshold Voltage

ODA – Observe-Decide-Act

OS – Operating System

PE – Processing Element

PID – Proportional-Integral-Derivative Control

PM – Power Management

REM – Runtime Energy Management

RM – Resource management

RTL – Register Transfer Level

RT – Real Time

SDF – Standard Delay Format

SOC – System-on-Chip

SOI – Silicon-on-Insulator

SPEC – Standard Performance Evaluation Corporation

SP – Slave Processor

TA – Task Allocation

TCF – Toggle Count Format

TDP – Thermal Design Power

TM – Task Migration

TMAP – Task Mapping

TR – Task Remapping

TSP – Thermal Safe Power

VCD – Value Change Dump

VF – Voltage-Frequency

VHDL – VHSIC Hardware Description Language

VHSIC – Very High Speed Integrated Circuits

XML – Extensible Markup Language

XY – XY Routing Algorithm

CONTENTS

1. INTRODUCTION

The high density of transistors allowed the development of a broad set of Systems-on-Chips (SoC) for different purposes. A trend in SoC design is *many-core* systems. A many-core system may be classified as heterogeneous or homogeneous (symmetric and asymmetric). Heterogeneous many-cores employ cores with distinct architectures and organizations (like general-purpose processors, graphics processing unit, dedicated hardware accelerators, among others) [EBA+12]. Symmetric homogenous many-cores correspond to systems with all cores having the same architecture and organization. Asymmetric many-cores are a particular case of homogeneous systems, where the cores share the same architecture (Instruction-Set Architecture - ISA), but not the organization [ARM13].

Examples of industrial many-core systems include Intel SCC (48 cores) [Int10], Mellanox Tile-GX (72 cores) [Mel17], Kalray array (256 cores) [DDVAPL14], and KiloCore chip (1,000 cores) [BSP+16]. Recently, even complex architectures follow the many-core trend, such as Intel Xeon i9 (18 x86-processors) [Int17].

1.1 Power Management in Many-cores

Many-core systems provide high-performance computing for distinct market segments, such as mobi, desktop computers, and servers. A key element in many-core systems design is to develop run-time management techniques for dynamically trading power, performance, energy and others conflicting parameters to achieve system and application requirements. Since each core usually allows dynamic settings of their control knobs, the management of a many-core system becomes challenging due to the amount of possible operating points.

Despite the many-core advantages, a certain number of cores must remain off (*dark*) during the applications execution due to power constraints. This restriction, called Dark Silicon [EBA+12], is more pronounced in recent technology nodes due to the higher number of integrated cores. Without respecting the power constraints, the system becomes vulnerable or unreliable to several problems such as cooling, faults from thermal issues, and fast aging effects.

The widely used power constraint is the *Thermal Design Power* (*TDP*), which is an upper bound of power globally defined to the system. *Power cap* and *power budget* can also refer to power constraints with the same meaning of TDP. Thermal constraints can also be employed instead of power ones [KPSH15]. More sophisticated approaches can consider the temperature and the activity of each core to maximize the utilization of

the system resources [PKM+14]. Quality of Service (QoS) can also constrain the system regarding performance, but it is not related to Dark Silicon [PAC+12].

1.2 Motivational Example

Besides the challenges related to controlling the core knobs and dealing with power cap, an additional challenge is the management of the dynamic workload. Dynamic workload refers to the unknown applications' set that will execute in the system. Even if the applications set is known, applications enter and leave the system at different moments, creating an unpredictable scenario of resource sharing. Thus, the system must monitor its resources to avoid undesirable situations, as excessive power dissipation.

Figure 1.1 shows the utilization and the power behavior of a typical workload in many-core systems. This workload presents peaks and valleys of system utilization and, consequently, power. In the example of Figure 1.1, the arrow labeling 38 tasks corresponds to a peak of utilization as well as the arrow labeling 17 tasks is a valley. The red line in the power graph corresponds to the power cap.

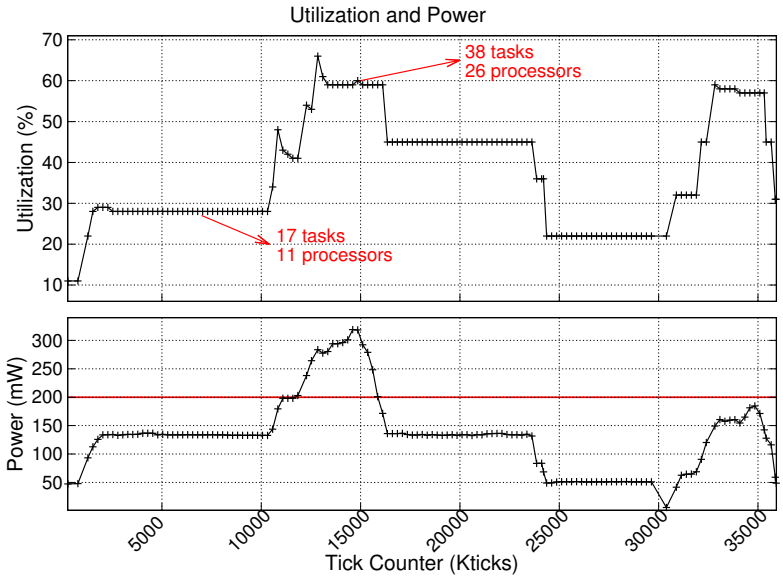


Figure 1.1 – Utilization and power for a 6x6 many-core system running a dynamic workload.

Due to the 200mW power capping, the system may not admit power peaks on a high utilization phase. For instance, the resource management (RM) should decide which applications speed up and/or down, creating a resource sprinting situation. Meanwhile, the low workload may be an opportunity for boosting applications and perhaps finish some of them before a peak since there is a power slack available. Another strategy for the low workload, the RM can also activate a low power mode to save energy. Mobile systems reproduce similar utilization behavior, presenting low workload when in standby mode and

high workload when the user is handling the system [NVI11b]. Even in active mode, the system utilization is frequently varying according to the number of executing tasks [NVI11a]. Servers for cloud computing are another example of a variation on the workload demand, but the time scale of the variation can be hours or days [Liu11]. *Thus, a multi-objective RM requires adaptive mechanisms according to the workload behavior.*

1.3 Problem Definition

The challenge of executing dynamic workloads on many-core systems under certain constraints has motivated researchers in several areas to employ RM to achieve one or more specific objectives. Due to several reasons related to technology, environment, and workload variation (such as thermal issues, security, performance, and energy efficiency), current many-core systems demand multi-objective RM [SDMI17]. The following definitions aim to clarify the *multi-objective* meaning according to Rahmani et al. [RJD17]:

Definition 1. *Goal* - a high-level result or plan for the system.

Definition 2. *Objective* - specific goal that the system is supposed to achieve or fulfill under at least one constraint.

Definition 3. *Multi-objective* - a system function that combines multiple objectives.

The goal can be selected, prioritized or optimized jointly with another goal. However, the objectives can conflict with each other. For example, a performance-driven RM does not consider other requirements like aging or temperature. In case of conflicting objectives, the RM can opportunistically select some goals to track along the time and dynamically switch between the goals according to certain conditions to satisfy the multiple objectives. Therefore, fixed-objective RM covers specific cases chosen according to narrow contexts.

Adaptability (see below) is an essential feature of runtime RM [DJS15] to support the unpredictable behavior of dynamic workloads and to enable the dynamic switches of objectives priorities.

Definition 4. *Adaptability* - the ability of replanning or updating system settings at runtime.

Executing dynamic workloads on many-core systems include several challenges to an RM. At the application level, the RM evaluates if the system has enough power and resources for new incoming applications [KP15]. Once the RM allows the application to execute, task mapping algorithms find the most suitable area to place application tasks. Further, task migration [SSKH13] and Dynamic Voltage Frequency Scaling (DVFS) may adapt the system and applications settings according to the system status [PKS+17] or goal switching. To support adaptability, the RM manages the control knobs available in the system at runtime

to meet the goals embedded in the management algorithms. Therefore, a comprehensive and adaptive management for many-core systems should include several mechanisms, like application admission, task mapping, task migration, and DVFS.

According to the taxonomy of RM methods [SSKH13], RMs can be centralized, distributed, or hierarchical (combination of centralized and distributed). In such complex systems, a centralized RM compromises the system performance because a single processor concentrates the management and affects negatively the scalability [SDMI17] by inducing network congestion and hotspots. With distributed RMs the many-core is divided into independent clusters in such a way that one core is in charge of managing the cores of its cluster. Distributed and hierarchical RM can guarantee scalability for current many-core systems [CMMM13, QP16].

The *fundamental problem* is how to control the set of actuators of a many-core-system at runtime to provide the required adaptability to enable the development of resource management running dynamic workloads.

1.4 Thesis Hypothesis

As stated before the target of interest in this Thesis are many-core systems running dynamic workloads. The final goal to achieve is to optimize many-cores by using multi-objective functions. The hypothesis to be demonstrated along with this Thesis is that the resource management hierarchically organized of the target systems is beneficial. The benefits are the coordination of the actuators and the distribution of the control complexity throughout the hierarchy levels.

1.5 Thesis Goal

The strategic goal of this Thesis is the proposition of a comprehensive set of methods for resource management adopting a hierarchical multi-objective approach, targeting many-core systems executing dynamic workloads under restricted power constraints.

1.5.1 Thesis Specific Goals.

The specific goals of the Thesis are the following:

- review the state-of-art related to RM proposals;

- present a general method to characterize the hardware of the reference many-core system to provide accurate per-core power and performance measurements;
- create a hierarchical observing infrastructure, using the characterization data, to support the RM decisions;
- evaluate available RM control techniques;
- propose a multi-objective RM targeting energy-efficiency under power and timing constraints in the context of real-time applications;
- propose a power-constrained multi-objective RM that opportunistically chooses between two objectives according to the workload behavior: energy and performance;
- compare the proposals with related works.

1.6 Reference Platform

Figure 1.2 illustrates the baseline many-core platform – HeMPS [CMMM13]. It is a homogeneous many-core, with a NoC (Network-on-Chip) interconnecting the set of Processing Elements (PEs). Each PE contains a processor (32 bits MIPS-like), a Direct Memory Network Interface (DMNI) [RLMM16], a local dual-port memory, and a router. This platform adopts a simple memory organization, without caches and shared memories. The local memory acts as a scratchpad memory, storing the tasks code and data. The memory is also organized in equally sized pages. Such simple memory organization simplifies the task mapping and task migration heuristics because any task may be assigned to any memory page. The methods proposed herein are not limited to the features of this platform. The reason to adopt this platform is to enable the validation of the proposed methods with clock-cycle accuracy.

The main NoC features include: (*i*) 2D mesh topology; (*ii*) input buffering for temporary flit storage; (*iii*) wormhole packet-switching; (*iv*) credit-based flow control; (*v*) routers with 5 bi-directional ports; (*vi*) round-robin arbitration; and (*vii*) XY routing algorithm.

Despite the same hardware of all PEs, they have different roles in the system. The Operating System (OS) running on the PE defines its role in the hierarchical organization. A PE can assume the role of cluster manager (CM), global manager (GM), or slave (SP):

- **CM**: resource manager at the *cluster* level (Definition 5). It executes the decision management regarding task mapping and DVFS;
- **GM**: resource manager at the system level. It receives application execution requests via the external interface (application repository) and decides which clusters execute the applications. The GM also acts as a CM, managing the PEs belonging to its cluster.

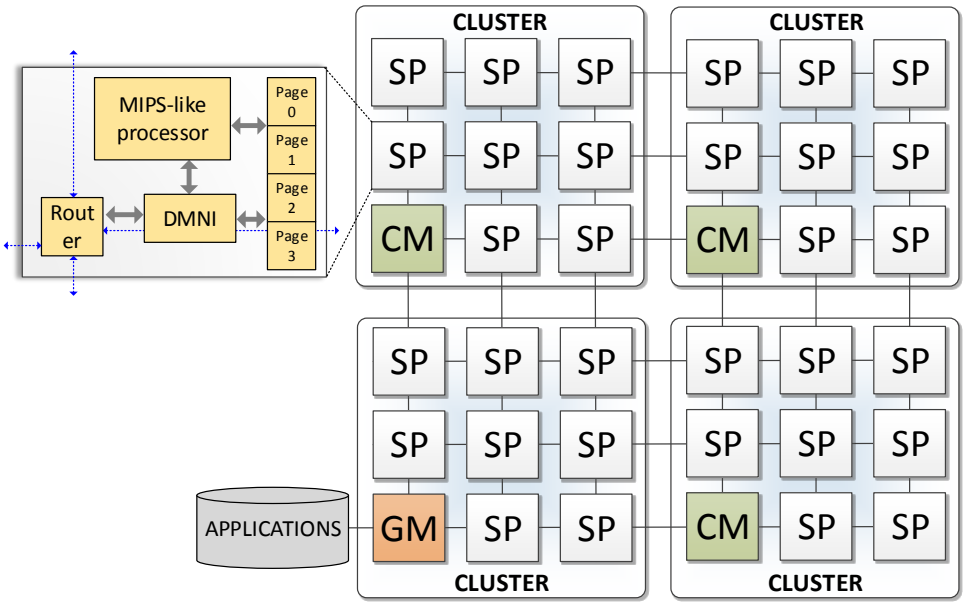


Figure 1.2 – A 6x6 instance of the homogeneous many-core reference platform, organized in four 3x3 clusters.

- **SP**: slave PEs, execute the applications tasks. Each SP executes a multi-task OS, enabling the concurrent execution of tasks.

Definition 5. *Cluster* - virtual region of the many-core, with a set of SPs and a manager PE (CM or GM). The cluster size is a design-time parameter, but a cluster can borrow resources from SPs of other clusters at runtime when there are no available PEs in the cluster to execute a given application. The protocol to modify at runtime the cluster size is named reclustering [CMMM13].

Figure 1.3 overviews the hierarchical organization by highlighting the hierarchy levels and the communication pattern. The exchanged messages related to the system management may be intra- or inter-cluster. SPs belonging to a cluster communicates with the manager PE of its cluster, characterizing an intra-cluster communication. Similarly, the inter-cluster communication occurs when CMs communicate with the GM. Note that, intra-cluster communication also occurs in the cluster managed by the GM because the GM acts as a CM in this case.

Applications are modeled as directed acyclic task graphs, $A = (T, E)$, where the vertex $t_i \in T$ is a task and the directed edge $e_{ij} \in E$ is the communication between tasks t_i and t_j. The adopted communication model is message passing. Figure 1.4 presents two examples of applications modeled as task graphs. In the current Thesis, applications are described in C language and they use MPI-like communication primitives.

Considering the paged memory organization and the application model, a memory page is a *resource* (Definition 6) to execute one task.

Definition 6. *Resource* - available page in the PE memory able to execute a task. Each PE can execute a set of tasks simultaneously (multitasking).

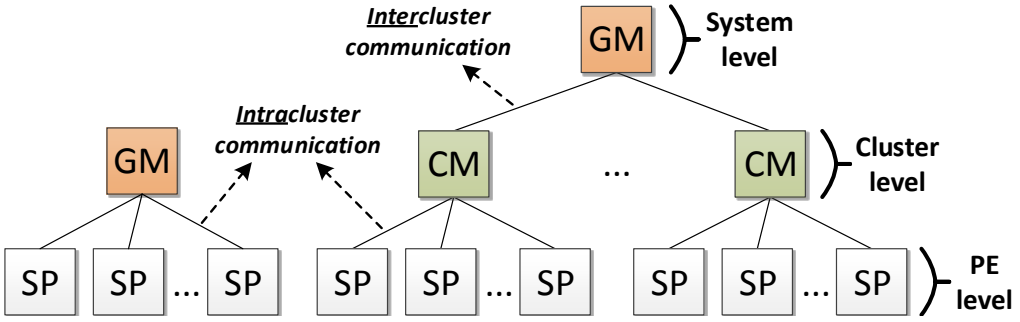


Figure 1.3 – Hierarchical organization of the PEs. The GM manages the system, the CMs manage a set of SPs, and SPs execute applications tasks.

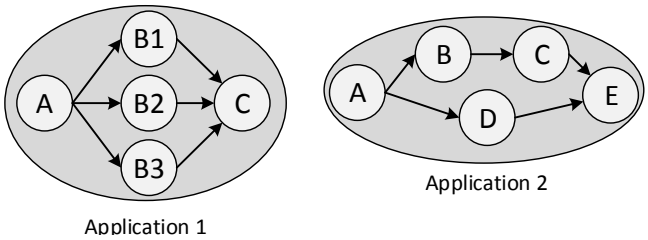


Figure 1.4 – Examples of applications modeled as task graphs.

1.7 Methodology

Develop an RM able to deal with conflicting objectives in many-core systems requires monitoring the communication and computation of each PE at runtime [SDMI17], and also requires the development and modeling of an actuation framework to enable the system to meet the RM decisions. The RMs proposed in this Thesis adopt the Observe-Decide-Act (ODA) paradigm to connect the monitoring to the actuation infrastructure [DJS16]. Figure 1.5 shows an ODA control loop. Each layer of the system (hardware and software layers) generates data for the decision algorithms from multi-layered sensors to enable system observing. According to the control policies implemented by the decision algorithms, actuation adapts the system layers to track the decisions.

The ODA paradigm requires self-awareness and self-configuration from the system to enable the adaptation policies [DJS15]. In the context of this Thesis, self-awareness means that the system can *observe* itself through virtual or physical sensors, and self-configuration is the ability of *actuation* on the available system actuators to update system settings. Further, the Observe state provides a multi-layer system sensing and the Act one is split into a multi-layer actuation. The multi-layer observing and actuation allow an individual control of each layer by enabling the hierarchical approach under ODA paradigm. Once *observe* and *act* are known and defined, the controller can adequately take *decisions* according to heuristics, PID-controller, or learning-based algorithms to meet the multi-objective purposes [SLR+17].

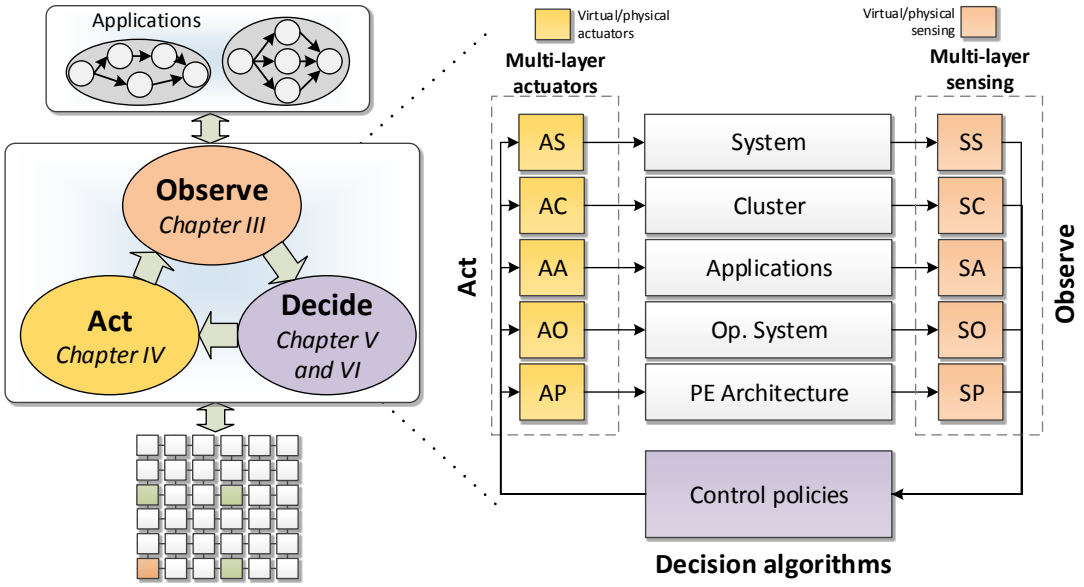


Figure 1.5 – Observe-Decide-Act paradigm is the methodology for the development of the Thesis goals applied to the reference platform. Adapted from [DJS16].

This Thesis proposes two approaches of multi-objective RM in large many-cores systems running dynamic workloads. Both RMs share the *observe* and *act* approaches, so that the *decide* one distinguishes the proposed RMs. The hierarchy approach of RMs is applied regarding layers in the ODA paradigm (Figure 1.5). Since *observing* is essential to provide an adequate measurement infrastructure, the *observing* design is introduced first. Following, the *actuation* mechanisms, such as DVFS and task allocation, modeled to meet the RM goals are presented. Next, the first RM proposal includes *decisions* concerning Quality-of-Service for real-time applications and energy consumption. The second RM proposal is a *decision* to control conflicting objectives like energy and performance under power capping.

1.8 Thesis Contributions and Originality

The main original *contributions* of this Thesis include:

1. a characterization method to accurately estimate power and energy of NoC-based many-cores by using RTL descriptions considering all PE components;
2. design and implementation of lightweight and scalable observing infrastructure for power, performance, and communication;

3. a comprehensive approach for RM that includes algorithms and actuators for application admission, task mapping, task migration, DVFS, and power-gating to make a trade-off between conflicting goals;
4. an energy-efficient RM to run soft real-time and best-effort applications with distinct heuristics while preventing the need of design-time analysis of the application set;
5. an RM approach that can dynamically adapt the system to the frequent changes of system goals due to the workload variation.

The main Thesis *originality* is the proposition and design of RMs addressing power, energy, and performance concomitantly, considering communication, computation and scalability issues. The hierarchical organization and the multi-layer ODA paradigm are the key enablers of the proposed heuristics.

1.9 Document Structure

Chapter 2 reviews related works and makes a qualitative evaluation of the literature with the proposed Thesis. Figure 1.5 presents the Chapters according to the ODA paradigm:

- Chapter 3 presents the *Observe* state. The Observe Chapter details the monitoring infrastructure and corresponds to the first and the second contributions previously mentioned.
- Chapter 4 describes the *Act* state (third contribution). The Act Chapter highlights are the hierarchical management, task migration, and the DVFS model and design.
- Chapters 5 and 6 present to the *Decide* state, which brings the major scientific contributions. The decision Chapters share some features, like the hierarchical organization and the multi-objective approach.
  - Chapter 5 (fourth contribution) presents a multi-objective RM for soft real-time applications in such a way of exploring the slack time of RT applications to obtain energy savings. The management of best-effort applications uses thresholds.
  - Chapter 6 describes an adaptive RM that can dynamically shift the system goals according to the dynamic changes of the workload behavior (fifth contribution).

Chapter 7 concludes this Thesis, and draws directions for future works.

2. RELATED WORKS

This Chapter reviews and discusses related works in Resource Management (RM). The Chapter finishes with a comparison between the key features found in the state-of-the-art regarding to RM.

2.1 Thannirmalai Somu Muthukaruppan et al.

Muthukaruppan et al. [MPV+13] work proposes their RM as hierarchical power management (HPM) integrated into a real platform (ARM.big.LITTLE running Linux OS). The big.LITTLE used has one cluster with two Cortex-A15 cores and one cluster with three Cortex-A7 cores. Both clusters execute the same ISA.

In the modeling step, the Authors perform experiments to evaluate the target platform according to the following criteria: (*i*) power-performance tradeoff; (*ii*) DVFS impact; (*iii*) impact of active processors on cluster power; (*iv*) task migration cost. The main conclusions from the modeling step are: (*i*) the A15 cluster dissipates more power and has better performance; (*ii*) mapping should balance the workload between the cores of the same cluster; (*iii*) migration of tasks between clusters takes longer than migrating tasks between cores in the same cluster.

Figure 2.1 shows an overview of HPM developed after the modeling step. A *Per-task QoS controller* ($QoSCtrl_i$) determines the range (maximum and minimum) of the application performance, and the shrinking of QoS (Quality-of-Service) of applications when the power is greater than the cap. A *Per-cluster DVFS controller* ($ClusterCtrl_i$) defines at run-time the minimum frequency to meet the QoS requirements. The *Chip-level power allocator* has the highest priority on the system and triggers new values for application performance and frequency when a TDP emergency occurs. *Per-task resource controller* ($ResShareCtrl_m$) keeps the applications performance within the QoS range defined by *QoS Ctrl*. *Balancer* and *Migrator* ensure the load balance of the workload at the cluster level, avoiding idle cores. The migration between clusters occurs as an exception when a task is demanding the maximum frequency of the A7 cluster or a task is running at the minimum frequency in A15 cluster.

Sensors to measure frequency, voltage, power, and energy of each cluster are available in the target platform. Experiments employ five applications from PARSEC, Vision, and SPEC. The platform provides a native scheduler, which maps tasks in the A15 cluster preferably and triggers power switching to respect TDP constraint.

Results demonstrate the HPM benefits from the asymmetric architecture and its RM is more efficient than the native scheduler. HPM achieves significant power consumption

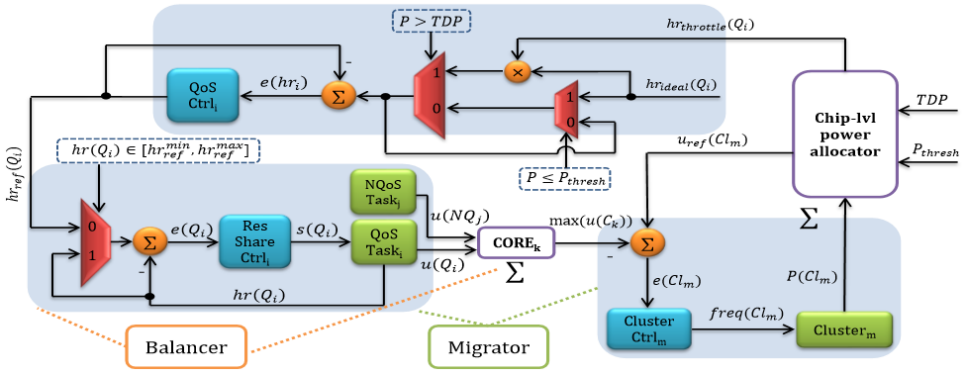


Figure 2.1 – RM overview of the Muthukaruppan et al. proposal [MPV+13].

reduction (∼69%) while has worst performance compared to the native scheduler (∼43%). The HPM maintains the power stable, near the TDP reference, even when the TDP changes dynamically during simulation.

2.2 Bharathwaj Raghunathan et al.

Raghunathan et al. [RTGM13] proposal explores process variation to schedule applications in a homogeneous many-core. The management can benefit from a variability model to choose the best subset of processors to maximize the performance under a power budget. Authors call this process to select a given processor as cherry-picking. They present a statistic model to represent the process variations between the processors. The cherry-picking management includes mapping, scheduling, power gating and frequency scaling.

Authors propose a polynomial time algorithm for optimally picking a subset of cores based on the variability parameters, mapping threads to cores in this subset and assigning operating frequencies to each core to maximize performance under a power budget.

Since the cherry-picking management knows the variability of the cores, the algorithm assigns for an application: (i) the cherry-pick core to execute the sequential phase, (ii) the cherry-pick core to execute the parallel phase, (iii) the dark cores, (iv) the frequency of each core running a thread. The frequency of the cores can be scaled after the mapping to respect the TDP. Each processor runs only one thread.

Figure 2.2 shows a process variation map for a 4x4 many-core with twelve cherry-picked cores and four dark cores. The applications used (SPLASH and PARSEC benchmarks) have one sequential thread and a set of parallel threads synchronized on a barrier (Figure 2.2), i.e., the processor begins a new execution when all threads of the parallel phase finish.

Three multi-processors containing 16, 24 and 32 cores run the same 16 threads, i.e., for the 16-core processor there are no dark cores, for the 24-core processor 8 cores are

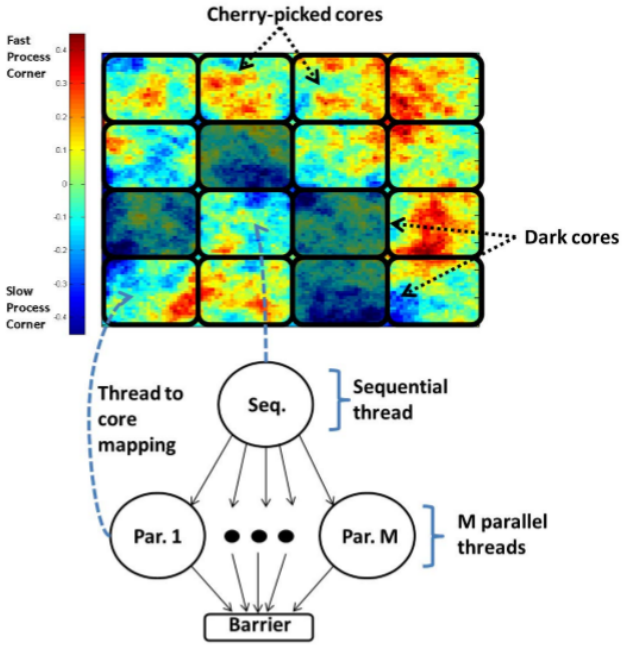


Figure 2.2 – Process variation map in Raghunathan et al. work [RTGM13]. Black borders identify the cores. Shadow cores are power gated. Each core runs a thread. The graph shows the barrier synchronization scheme.

dark, and 16 cores are dark in the 32-core processor . The results show the 24-core and the 32-core processors achieve 22% and 30% of performance improvement, respectively, compared to the 16-core processor by using the cherry-picking method. Results are a function of the process variation and the standard deviation used for calculating the variation in the experiments.

2.3 Heng Yu et al.

Heng Yu et al. [YSH14] work presents an RM concerning temperature for heterogeneous many-cores running adaptive workloads. Adaptive applications have the capacity of modifying their executing quality according to execution status to maximize Quality of Service (QoS) dynamically, i.e., the more cycles the application can execute in a given time interval, the bigger the number of deadline misses reduction.

The high-level system model includes: (*i*) adaptive task model; (*ii*) power model to execute the DVFS (the model assumes the processors operate under a finite set of voltage-frequency pairs), and (*iii*) thermal model adapted from HotSpot [HGV+06].

Considering the behavior of the target workloads, a monitor determines the state of the task as "hot" or "cool". If the task is "hot", the frequency scaling algorithm may decrease the task frequency. Similalry, the frequency can increase for "cool" tasks. The actuation

heuristics are constantly adapting the frequency to reach the dynamic QoS from adaptive applications.

Nine heterogeneous cores are prototyped on a Virtex-6 FPGA board. There are no details about the heterogeneity of the cores. An in-house tool randomly generates a synthetic workload. The software tool available on FPGA reports the power results. Thermal results are estimated.

On the first experiment, Authors show the management can avoid up 90% of thermal deadlines on the system for adaptive workloads, but the workload requests no significant cycle gains. On a second experiment, the number of cycles increases the management with DVFS running adaptive workload under thermal constraints (65^o, 70^o and 75^o) in 31,5%, 9,5%, and 3% respectively compared to one without DVFS.

2.4 Santiago Pagani et al.

Pagani et al. [PKM+14] work presents a new power budget for many-core systems to maximize the power efficiency as an alternative to Thermal Design Power (TDP). TDP is a global constant power value used to avoid problems caused by high temperatures. However, TDP leads to underutilization of the system when the mapping creates hotspots on the system. Instead of considering the power of the whole system, the new metric, called Thermal Safe Power (TSP), is a function of the temperature of each core and of the floorplanning (mapping) of active/dark cores.

Figure 2.3 shows two mappings of six active cores in a 4x4 many-core, for which the TSP is 80^oC. In the worst-case mapping, the maximum power of cores is higher than best-case due to the temperature of the neighborhood. The pattern mapping of the best-case mapping allows the active cores to work with higher power values by avoiding the neighborhood effect.

Two RM are developed considering TSP as a temperature constraint. One uses TSP online measures to map active cores and another defines mapping patterns at design-time. Both algorithms are described as a function of power but the model abstracts the technique (DVFS, for example) used to vary the power.

Experiments use Gem5, McPAT [LAS+09], and HotSpot [HGV+06] tools. The hardware platform is an 8x8 many-core, and the processor is an Alpha21264 in 22nm technology. The TSP online and TSP offline are compared to RMs with power constraints per-core and per-chip. The results show the TSP management achieves higher performance compared to other approaches when the number of active cores is high by avoiding the management triggers due to thermal constraints. For a small number of active cores, the performance of both approaches is similar.

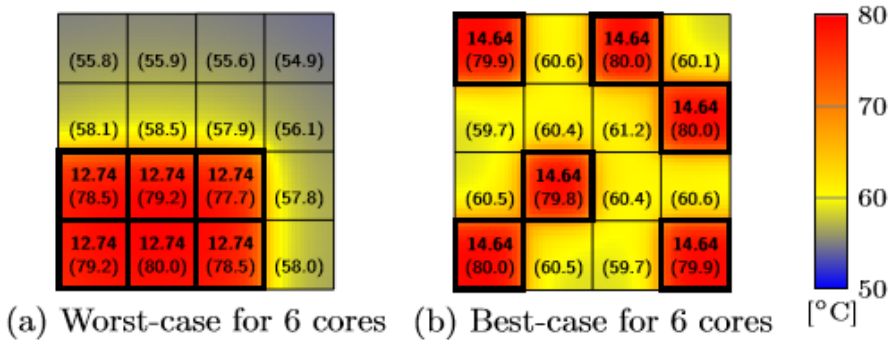


Figure 2.3 – Example of worst-case and best-case mappings concerning thermal constraint. The top bold numbers are the power (Watts) of the active cores. Bottom numbers in parenthesis are the temperature in the center of cores [PKM+14].

2.5 Vinay Hanumaiah et al.

The primary goal of the RM presented in Hanumaiah et al. [HV14] work is to maximize energy efficiency for heterogeneous many-core systems. The three specific control policies modeled operate in the system fan speed, voltage and frequency scaling per core, and task-to-core assignment (mapping/migration). Each core executes one task at a time. The heuristic of the management model considers the time scale of each control police (Figure 2.4). When a control technique executes, the heuristic blocks other control techniques until the end of the expected delay. For each time window of the control technique latency, the heuristic looks for new values to maximize the energy efficiency, e.g., as the fan speed has 1-3 seconds of latency, the heuristic looks for better values every 1-3 seconds.

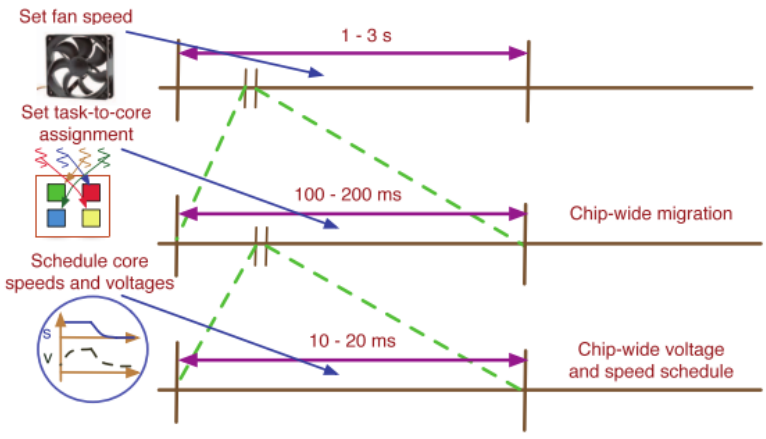


Figure 2.4 – Delay and time scales of the control techniques used in Hanumaiah et al. work [HV14].

Authors have an in-house tool, called Magma, to simulate their RM in many-core systems using a power model (adapted from PTScalar) and a thermal model (HotSpot) in

the background. Details about the system architecture (heterogeneity, topology, etc.) are not presented.

Authors execute experiments to evaluate performance, power, temperature, and PPW (performance per Watt - a measure of energy efficiency) in three scenarios: maximum performance, minimum energy, and maximum PPW. In the maximum performance scenario, the frequency is set to the highest value, in such a way to reach also the maximum allowed temperature. The results show the maximum PPW scenario offers the best tradeoff between performance and energy. For all scenarios, energy efficiency decreases when a task terminates because the idle cores are not power gated.

The DVFS of the RM is adapted for running on a real system (quad-core Intel Sandy Bridge processor) while task scheduling and fan speed control are out of this experiment. The DVFS improves the energy efficiency of the system by at least 37%. The results could be better if the power and thermal model were available or extracted in a modeling step (as presented in Muthukaruppan et al. work [HV14]). Authors report that the processor supports DVFS at the system level (DVFS modeled at the core level), and the sensors have low sampling rate and are noisy.

2.6 Paul Bogdan et al.

Paul Bogdan et al. [BMJ13] proposal is an RM using a fractal control approach for NoC-based systems. According to Authors, classical control theory cannot correctly model the characteristics of real applications (such as burst workloads), and many-core systems have no steady state due to the frequent and unpredictable disturbances (mapping of a task, for instance), so the fractal control approach represents more adequately the behavior of real systems.

Figure 2.5 is an overview of the Bogdan et al. [BMJ13] approach. From the set of *applications* and *architecture* data, the *fractal modeling* reads *NoC workload* measures as network queue utilization, arrival time process (task) and departure time process. Afterwards, the *parameter identification* derives a fractal model by estimating the *fractional order* of the time *derivative* and NoC parameters. The parameters required by the system control are calculated from the *parameter identification*. Finally, the *optimal controller design* determines frequency and voltage of the routers and the processors according to *constraints*. Router and processor of the same core can run at different frequencies while the voltage is the same for both.

The experiments are performed in a 4x4-mesh NoC system running Apache server and Oracle database applications. Some details about the experimental setup are missing (simulation tools). Authors derive classic control (PID) for the RM of the fractal model for the experiments. The results show the PID controller cannot avoid peaks of power (power

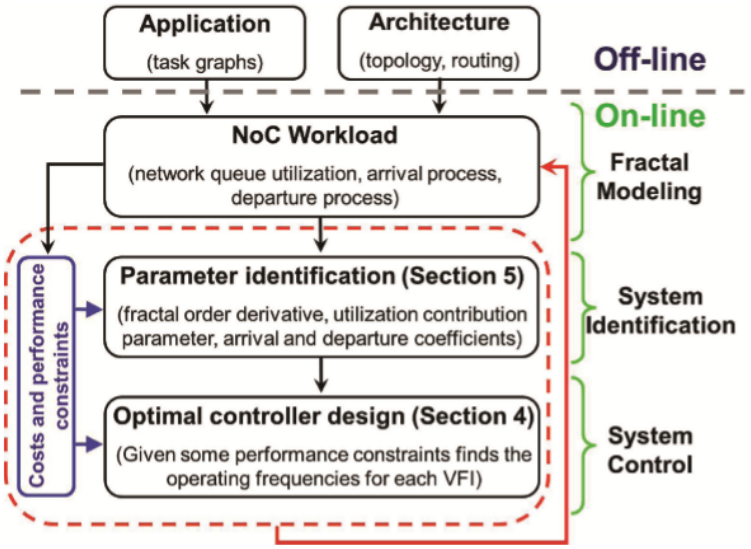


Figure 2.5 – Methodology overview description of Bogdan et al. work [BMJ13].

overcomes the reference), and the power consumption is 30% worst compared to fractal approach. According to the Authors, the RM proposed is not scalable. An instance of the RM (without many-core integration) is synthesized in FPGA to justify the practical purpose of their proposal.

2.7 Amir-Mohammad Rahmani et al.

Rahmani et al. [RHK+15] work proposes an RM for NoC-based many-core systems running dynamic workload - called MOC (Figure 2.6). The method distinguishes real-time applications and non-real-time applications, and models the applications as task graphs.

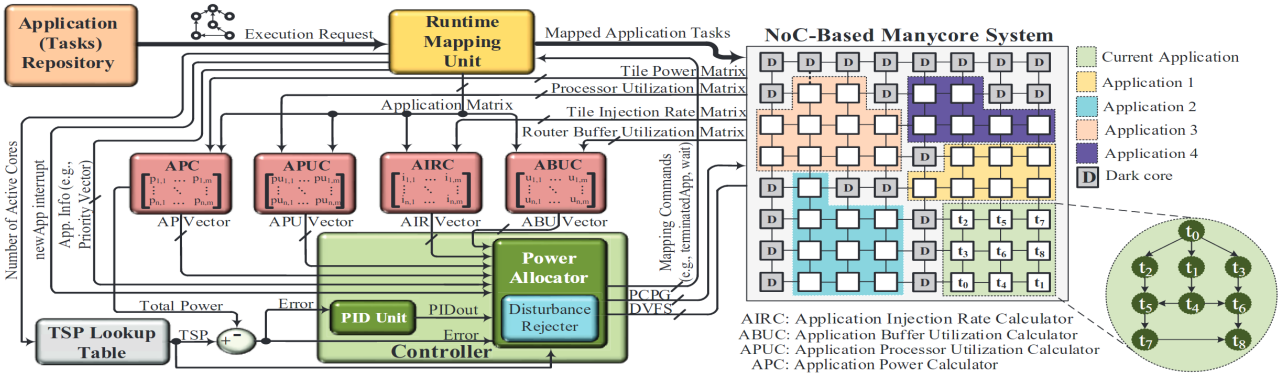


Figure 2.6 – System overview of Rahmani et al. proposal [RHK+15].

Figure 2.6 is an overview of Rahmani et al. [RHK+15] work. Runtime Mapping Unit embodies the SHiC method [FDLP13] to select the first node to map tasks using the CoNA mapping [FRD+12] algorithm. CoNA and SHiC are described in specific papers.

The Tile Power Matrix provides the power consumption of all cores (Authors assume each core has a power sensor) to the Application Power Calculator (APC), which calculates the power consumption of each application. Similarly, Applications Buffer Utilization Calculator (ABUC) provides the congestion of the applications. Likewise, APC and ABUC, the Application Processor Utilization Calculator (APUC) and the Application Injection Rate Calculator (AIRC) contain respectively information about CPU utilization and the network latency from the router injection rate. These matrices consider computation and communication aspects to manage the system.

The matrices size is the same as the system size. Since the TDP is the reference power for the controller, APC calculates the total power of the system comparing it to the TDP. The *Power Allocator* adjusts power and frequency settings for each core. The *PID Unit* stabilizes the system when a power violation occurs, i.e., it is a reactive actuation. *Disturbance Rejector* is a proactive actuation to handle power overshoots caused mainly when an application begins or finishes. Simulations in Matlab define the PID (Proportional-Integral-Derivative) controller gains. Whether the DVFS is not enough to maintain the power cap, *Power Allocator* can terminate or pause the application with the lowest congestion. TSP lookup table is the power budget.

The Authors perform the experiments on a System-C many-core platform. The Lumos framework [WS12] provides physical parameters for the model. MPEG4, VOPD, and UAV are the real-time applications and the non-real-time applications are synthetic.

Three experiments are performed for a 12x12 NoC-based many-core. The PE in the homogeneous system is a Niagara2-like core from McPAT [LAS+09]. For comparison purposes, three other RM are simulated: (*i*) DSAPM -– dark silicon-aware power management, power management with APUC, AIRC, and Disturbance Rejector; (*ii*) PGCapping — Power-aware Mapping, power management where only power-performance ratio is the feedback; (*iii*) no TDP constraint, no power management. The results show that without RM, the system stays too much time above TDP reference while DSAPM and PGCapping can keep the power below TDP most of the time. Comparing DSAPM and PGCapping, the data throughput of MOC is 15% and 29% better, respectively. The power graphs show a better utilization of power for MOC as well as the lowest power overshooting due to the proactive actuation. The Authors do not detail if the management compromises the real-time tasks nor report other results.

2.8 Shoumik Maiti et al.

The power management proposed by Maiti et al. [MKP15] considers the process variation inherent to the new technologies nodes affected by the utilization wall (or dark silicon). The supported actuation techniques are mapping and DVFS.

At design-time, process variation parameters are captured (Monte Carlo Spice simulation) for the range of frequencies used in DVFS at nominal voltage and temperature. The parameters of the power model for each core derive from the process variation model. Process variation model, power model and instruction per cycle (IPC) of all applications are the *inputs* for the power management (Figure 2.7).

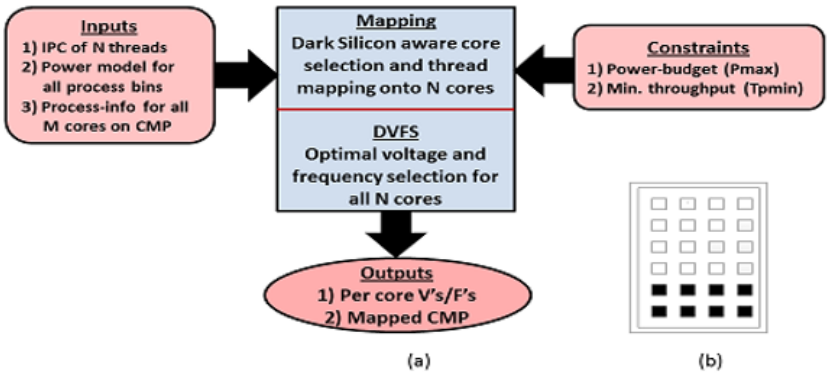


Figure 2.7 – (a) Power management overview of Maiti et al. [MKP15] proposal and (b) 4x4 selected cores (white squares) from 6x4 cores. The *outputs* define the voltage and frequency of active cores.

Mapping in this work means selecting a contiguous rectangular region of cores to activate while the other cores are clock gated. The mapping heuristic uses the process variation data for selecting the active cores according to two schemes: a contiguous rectangular region with (*i*) the minimum leakage and (*ii*) the maximum performance. The management chooses the adequate scheme according to throughput required (IPC) by the application.

The system can target minimum energy or maximum performance while respecting the system *constraints*. To respect these different system requirements without power gating, the *DVFS* has an extended range of voltage and frequency, which varies from the near-threshold voltage (NVT) up to the turbo boost voltage (a voltage value higher than the nominal one). The voltage/frequency of all cores is updated at every epoch (term used by the Author to determine a time window). The Authors consider the DVFS latency as 9 ns and the control epoch as 1 ms.

The experiment setup uses Sniper [CHE11] as the architectural simulator. The target architecture is a 24-core NoC-based system, with x86-processors for a 16 nm technology. The McPAT tool [LAS+09] provides power traces. One hundred random 24-core die profiles with distinct process variations are generated to run the PARSEC benchmark.

A comparison between the process variation-aware power management and a similar version unaware of variation information evaluates the process variation model. Results show a 3.7% improvement in energy minimization under throughput constraints, and an 11.9% gain in maximum throughput under power budget, on average, by considering the process variation on management. Moreover, a simplified version of the power management with nominal range DVFS (without NVT and boosting) without process variation information

is compared to the proposed one. The proposed power management results in 15% energy gains and 14.6% throughput improvement over the nominal DVFS version.

2.9 Nishit Kapadia et al.

Nishit Kapadia et al. [KP15] work integrates reliability and variation models in a runtime variable DoP (Degree of Parallelism) application scheduling methodology for power constrained many-core systems, named VARSHA. VARSHA includes runtime application mapping, dynamic application scheduling of DoP, and chip-wide voltage scaling. All features are aware of reliability and variation status of the system in the algorithms.

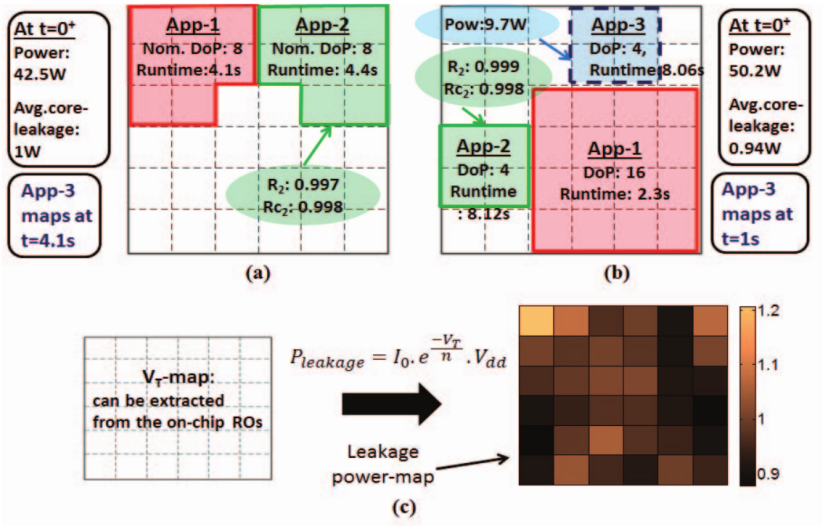


Figure 2.8 – VARSHA approach overview [KP15]. (a) When an incoming application arrives, (b) VARSHA reviews the DoP of all applications to optimize the power utilization (difference between TDP and power dissipation). (c) The position of the applications and the voltage settings rely on the leakage variability.

Figure 2.8 gives an overview of VARSHA approach. Figure 2.8(a) shows a 6x6 many-core running two applications and a third application, which is about to exceed the 50W TDP, requiring allocation as an example of the starting point. The VARSHA approach changes the DoP of all applications and returns a new mapping to optimally use the available power (Figure 2.8(b)). The variability of leakage is considered in this new application scheduling (Figure 2.8(c)). When an application arrives or departures, VARSHA first reviews the voltage settings and after it performs the application scheduling (DoP and mapping) of all running applications. Assuming the same priority for all applications, the application schedule consists of (*i*) defining the DoP for each application, and (*ii*) mapping the adequate task graph in a contiguous square shape area.

The benchmark used for the experiments includes 14 parallel applications for DoPs of 4, 8, and 16. The experiments are conducted with the gem5 simulator for a 10x10 homo-

geneous many-core where each core is an ARM processor. The power values for the links and router come from the ORION tool. It is assumed five voltage levels and a TDP of 100W. VARSHA is compared to other two prior works, and the results show savings of 11-13% in energy. Also, VARSHA avoids all reliability violations due to the ability of dynamically change the DoP while other works suffer up to 11% of violations.

2.10 Huazhe Zhang et al.

Huazhe Zhang et al. [ZH16] proposal introduces a RM under power cap using software and hardware actuation. To justify the need for a hybrid (software and hardware) approach, two power properties are discussed: *timeliness* – the speed to set the power; *efficiency* – the performance under the power cap. Authors discuss that software-only approaches can be timing consuming and hardware-only approaches do not consider the performance. The proposed RM, called PUPiL, combines software efficiency and hardware timeliness to maximize the performance under the power cap.

PUPiL also employs the ODA paradigm, integrated into a decision framework. The steps of PUPiL decision framework can be summarized as follows: (*i*) at design-time, sort all software resources concerning *efficiency* in descending order. For example, turn on a core has more efficiency impact than a new thread. At runtime, (*ii*) start with minimal configuration of all resources; (*iii*) remove software control from hardware actuation, like DVFS; (*iv*) for each software resource, do a binary search between the possible configurations starting from the maximum value to improve the system performance while fullfilling the cap.

A dual-socket chip server with eight x86 cores each is the platform for the experiments. The benchmark includes 20 multi-thread applications running on Linux for five power caps. PUPiL is compared to a hardware-only approach (native from the experiment platform) and two software-only approaches: a DVFS set by software, and PUPiL with hardware settings hardened.

Regarding *efficiency* results, PUPiL outperforms hardware-only approaches up to 32% in performance in single application workloads and up to 2.43 times for specific multi-application workloads. On the other hand, hardware-only can achieve better results than PUPiL when applications have a good scale of parallelism. Concerning *timeliness* results, PUPiL is orders of magnitude smaller than software-only approaches.

2.11 Anup Das et al.

The runtime manager (named RTM) proposed by Anup Das et al. [DAHM16] simultaneously addresses two goals: energy and temperature. Three thermal aspects are

considered: peak temperature, average temperature, and thermal cycling. RTM is in charge of allocating the threads to cores and selecting the core frequencies. To deal with the high complexity of the problem, hierarchical management approach is the choice to minimize the learning overheads and provide scalability.

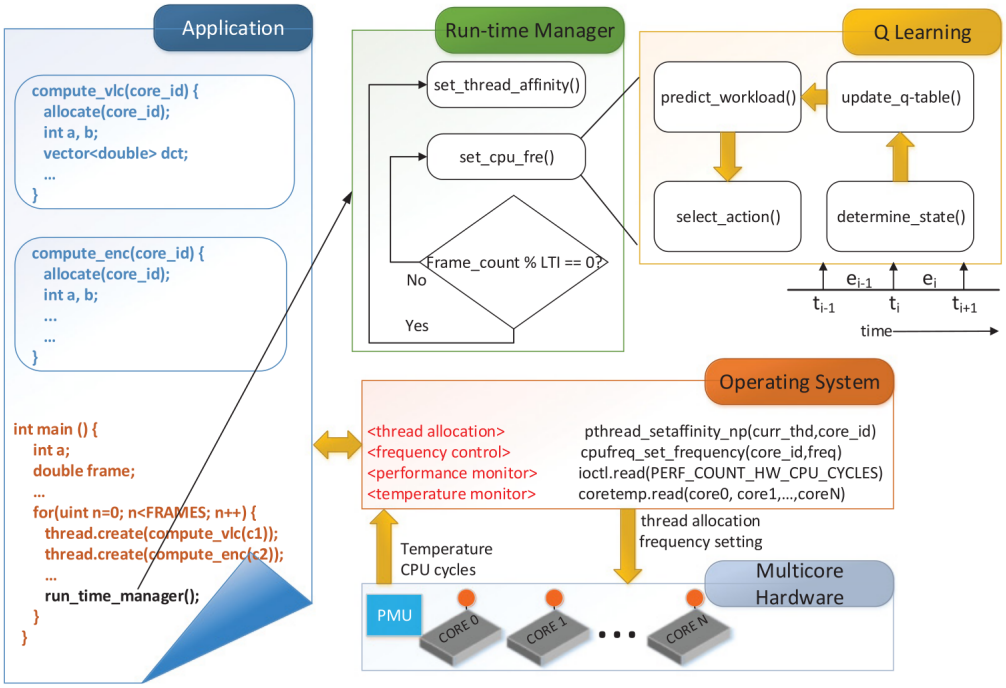


Figure 2.9 – Hierarchical runtime manager proposed in Das et al. [DAHM16] work.

Figure 2.9 overviews the method. The application code (blue box) calls the *run_time_manager* (RTM – green box). In the RTM, at the end of a *long time interval* (*LTI*, corresponding to the epoch), the *set_thread_affinity()* heuristic is invoked to assign threads to cores, and it is responsible for managing the thermal cycling aspect. Otherwise, at the end of short time intervals, the Q-learning module (yellow box – learning-based frequency selection) selects the frequency considering the average temperature, peak temperature, and energy consumption. The application interfaces with the operating system by using the drivers shown in the orange box. RTM actuates proactively to prevent thermal emergencies.

The experimental set-up is a real platform with a quad-core ARM with chip-level DVFS running Linux kernel. The multithreaded applications are periodic with deadlines to meet. RTM is compared to native Linux governors and an Energy-only approach. The results regarding performance show linux Governors ignore the latency required by periodic applications leading to underperformance or overperformance. Energy-only approach achieves a sligthly better results in performance (no timing violations), power (9% smaller than RTM), and energy (9% of savings). On the other hand, RTM outperforms Energy-only in all aspects related to temperature: reduction of 13.15% and 18% in average temperature and peak temperature, respectivelly, and 2x less thermal cycles than Energy-only approach.

2.12 Daniel Olsen et al.

Daniel Olsen et al. [OA17] proposal is a resource management framework to carry out a thread to core mapping on many-core systems (Figure 2.10). At design-time, the framework performs profiling for the various threads of each multi-thread application targeting the maximization of the resources utilization. Next, the application profiling classifies the threads based on throughput, power, and temperature. At run-time, the resource management looks for hot spots of temperature to identify the regions of the system that benefit or suffer from a change in system mapping. Next, the RM finds (*i*) the appropriate number of threads for the incoming application, and (*ii*) the thread-to-core mapping regarding performance that satisfies the application constraints.

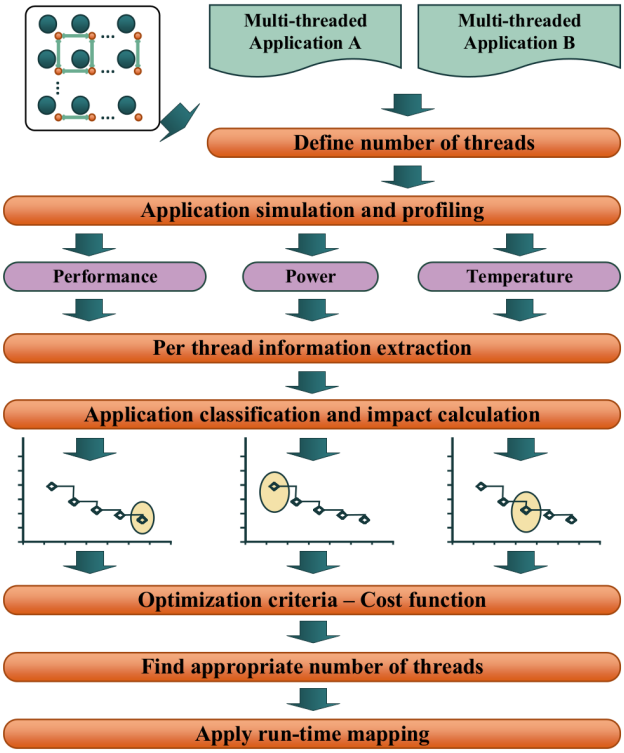


Figure 2.10 – Resource management methodology of Daniel Olsen et al. work [OA17].

Regarding the experimental setup, the Sniper simulator is the tool to run the proposal and generate the profiles. Power consumption and thermal profiles are extracted from McPAT and HotSpot tools, respectively. Five versions of parallel applications are derived to 1, 2, 4, 8, and 16 threads to compose the benchmark. It is assumed a 16-core system for the experiments and the distribution of applications arrival intervals follows two scenarios: large interval and small interval. The large interval covers a lightweight workload situation, and the small one simulates an amount of applications requiring admission in such a way that no resources are available for serving them all.

Results demonstrate that Olsen et al. approach saves 23% in average applications execution time and creates up to 34% less thermal hotspots compared to related works in large interval scenarios. In small interval scenarios, Olsen et al. approach reduces 19% the applications execution time in average and generates 21% fewer hotspots.

2.13 Anuj Pathania et al.

Anuj Pathania et al. [PKS+17] work proposes an RM as a probabilistic power budgeting for many-core systems - *ProGov*. The primary argument is that monitoring the many-core is not required when a large number of tasks is running on the PEs. Assuming a global power budget, the average power consumption is stable with small standard deviation, despite the fact that task phases leverage frequent variations in PE power.

The main *ProGov* advantage is to reduce the overheads to manage the system like the monitoring. ProGov assumes a task-to-processor mapping and two *vf*-levels (*vf* refers to voltage and frequency settings) for each PE: *Low* and *High*. A probabilistic phase profile generates a histogram of DVFS speedup for each task at design-time (Figure 2.11). Speedup is the metric for measuring the performance variation due to DVFS. For each task, a *strategy* is associated to represent the speedup threshold. For example, Figure 2.11 shows the probability of assigning *High* DVFS for a task when a speedup of 2.5 times is defined as the strategy. Task probabilities are the starting point for proposing the *Probabilistic Performance Model*, *Probabilistic Power Consumption Model*, and *Probabilistic TDP Model*. Since ProGov is a probabilistic approach, no TDP violation is guaranteed, and no adaptability such as task migration and *vf* resetting is allowed.

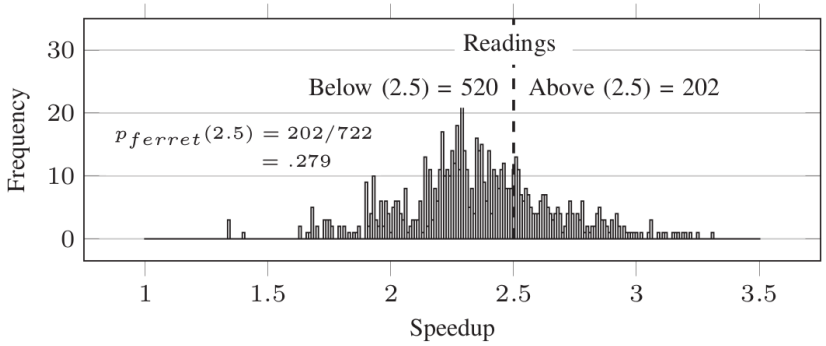


Figure 2.11 – DVFS speedup in the single-threaded application (called *ferret*). The strategy is 2.5, so the histogram illustrates the calculation of $p_{ferret}(2.5)$

The experimental set-up consists in a two-stage simulator of a homogeneous many-core embedded with the Alpha ISA. The first stage, a cycle-accurate simulator (gem5 and McPAT tools) of eight cores generates the traces files of the benchmark without ProGov. The output traces in the first step are the input for the second one. The second step is an

in-house trace based Simulator with ProGov, which generates final traces to 1024 cores. A workload of 256 tasks (1024 threads) is simulated with a target of boosting 75% of tasks.

The first set of results illustrates ProGov models can accurately predict TDP violations, power consumption, and DVFS settings distribution. In particular, ProGov provides minimal risk of TDP violations, but the accuracy decreases if the number of tasks shrinks. The second set of results is a comparison to a scalable non-probabilistic approach (SortedWS). The comparison shows similar throughput between ProGov and SortedWS. Concerning scalability, the main motivation of Anuj Pathania et al. [PKS+17] work, ProGov reduces up to 97.13% the time to allocate the power budget.

2.14 Navonil Chatterjee et al.

Navonil Chatterjee et al. [CPMC17] proposal introduces dynamic task mapping and scheduling (called DEAMS) of periodic, sporadic and aperiodic applications in NoC-based many-cores. Periodic and sporadic applications have timing constraints to meet, but the interval of sporadic ones is variable at runtime. The goal is energy minimization and deadline meeting.

In the homogeneous many-core system, a manager processor is in charge of performing resource management, task mapping, and task scheduling. Remaining processors are only responsible for running tasks. Besides, the edges in the task graph application have attached the communication cost. The heuristic looks for a continuous region for mapping the application while avoiding overlapping regions with other applications. To minimize the communication energy, multitask mapping is supported, and a task is preferably mapped onto the processor containing a communicating task with the highest communication load. The task with the highest sum of communication costs is the first to be mapped while the order of remaining tasks can follow filters that consider the processing time and the deadline constraints.

An 8x8 many-core with a centralized manager is considered for the experiments. An in-house C++ simulator carries out the experiment. The energy of a generic 5 port router is derived from the ORION power model. The test cases vary the scheduling difficulty for the application set by creating three categories of scheduling: urgent (30% of the task execution time), moderate, and relaxed (60% of the task execution time). For urgent applications, DEAMS is not able to satisfy the deadlines. For moderate and relaxed ones, DEAMS meets 75% and 90% of deadlines. The Authors argue the communication-bound tasks are more likely to miss the deadline. Concerning energy results, DEAMS reduces 28% of communication energy compared to related works.

2.15 Related Work Analysis

Table 2.1 summarizes the reviewed works according to the classification chosen for Resource Management (RM) comparison. The first column contains the Authors and the corresponding reference. The second column presents the architecture (symmetric/asymmetric homogeneous, heterogeneous) and the core counting given by the number of processor elements or the NoC size (if NoC-based many-core). The third column lists the techniques to evaluate the comprehensiveness for controlling the system. The fourth column concerns the abstraction level of the system modeling which results are produced according to the following standard proposed in Matthew et al. work [WDH+16]: top-down or bottom-up. The fifth column classifies whether RM is the multi-objective approach. The sixth column highlights if the RM algorithms depend on design-time evaluation of the applications set. The seventh column highlights qualitatively the proposals scalability. The last column shows when RM supports multitasking.

Table 2.1 – State-of-art in RM for Many-core Systems running under a power-related cap.

Proposal	Arch., # PEs	Actuation set	Modelling	Multi-objective?	Dynamic workload?	Scalable?	Multiple tasks per core?
Muthukaruppan et al. [MPV+13]	Asym., 5	TA, TM, DVFS, PG	Top-down (ARM-based embedded system)	✗	✓	✗	✓
Raghunathan et al. [RTGM13]	Homo., 16 up to 32	TA, DVFS, PG	Bottom-up (Sniper, McPAT)	✗	✗	✗	✗
Yu et al. [YSH14]	Hetero., 9	DVFS, TM	Top-down (FPGA simulation)	✓	✓	✗	✗
Pagani et al. [PKM+14]	Homo., up to 64	TA, PG	Bottom-up (gem5, HotSpot, McPAT)	✗	✓	✗	✗
Hanumaiah et al. [HV14]	Asym., up to 64	TM, DVFS	Bottom-up (validation in x86 system)	✗	✓	✗	✗
Bogdan et al. [BMJ13]	Homo., 4x4	DVFS, PG	Top-down	✗	✓	✗	✗
Rahmani et al. [RHK+15]	Homo., 12x12	TA, DVFS, PG	Bottom-up (in-house tool, McPAT, Lumos)	✓	✓	✓	✗
Maiti et al. [MKP15]	Homo., 6x4	TA, DVFS	Bottom-up (McPAT)	✓	✗	✗	✗
Kapadia et al. [KP15]	Homo., 10x10	AA, TA, DVFS	Bottom-up (in-house tool, ORION)	✗	✗	✓	✗
Zhang et al. [ZH16]	Homo., 2x8	TA, TM, DVFS	Top-down (x86 system)	✗	✓	✗	✓
Das et al. [DAHM16]	Homo. 4	TA, TM, DVFS	Top-down (ARM-based embedded system)	✓	✓	✓	✓
Olsen et al. [OA17]	Homo., 16	AA, TA	Bottom-up (Sniper, McPAT, HotSpot)	✗	✗	✗	✗
Pathania et al. [PKS+17]	Homo., 1024	TA, DVFS	Bottom-up (gem5, McPAT)	✗	✗	✓	✗
Chatterjee et al. [CPMC17]	Homo., 8x8	TA, TM, DVFS	Bottom-up (in-house tool, ORION)	✓	✓	✗	✓
This Thesis	Homo., 3x3 up to 12x12	AA, TA, TM, DVFS, PG	Top-down	✓	✓	✓	✓

AA: Application Allocation; TA: Task Allocation; TM: Task Migration; PG: Power-Gating

Homo.: Symmetric Homogeneous; Asym.: Asymmetric Homogeneous; Hetero.: Heterogeneous;

The literature presents distinct RM approaches for many-core systems. Most works in Table 2.1 adopt symmetric homogeneous architecture (2^{nd} column), but the management of asymmetric and heterogeneous architectures are more complex than the symmetric ones.

On the other hand, the core count for symmetric homogeneous many-cores is larger than asymmetric and heterogeneous ones.

The actuation set column (3^{rd} column) exhibits the comprehensiveness of each approach. The RM complexity and comprehensiveness is a function of the number of actuators. Zhang et al. [ZH16] demonstrate that jointly setting of various actuation steps leads to better results in general. The challenge of a comprehensive RM is to coordinate the actuators set to follow the same goals while avoiding overlapping between them [RJD17]. The ODA paradigm applied in comprehensive RM approaches evidence the coordination of the different actuation layers in a holistic RM view. This Thesis and Zhang et al. [ZH16] are RMs using the ODA paradigm. Das et al. [DAHM16] deploy the RM in the application layer to guarantee the management coordination. The remaining works focus on the decision and abstract details about observing and actuation.

Regarding the system modeling classification (4^{th} column), bottom-up approaches generate the results (time, area, power, among others) from a generic estimation of the system features or based on a target ISA. Top-down approaches include cycle-accurate simulators (like the one assumed in this Thesis), and real systems that report results from a chip. Bottom-up approaches allow fast evaluation of the design space, but lead to incompatibility between the model and hardware by making assumptions related to the hardware characteristics. For instance, Hanumaiah et al. [HV14] report problems to embed their resource management because the target platform does not support all their power techniques, just a system-level DVFS (a per-core DVFS is originally modeled). Another example, DVFS, and power gating are the power actuation techniques to cap the system. However, the overhead of these techniques is omitted. For example, the latency of both power techniques is on the order of milliseconds for real systems [Int10, LLW+13, AMP+15]. In this Thesis, a realistic DVFS model includes latency and energy overheads while the power gating model mirrors related work model for fair comparison purposes (Section 4.2).

The characteristics of the new technologies nodes introduce challenges and opportunities for exploration of the design space in the context of temperature, reliability and variability [SGHM14]. Therefore, the works targeting these performance figures [RTGM13, PKM+14, MKP15] are proposed only at top-down approaches. However, the mapping patterns needed for the system benefit of the temperature constraint and variability increase the hop distance between the tasks/threads of the same application. There is no discussion about performance losses or traffic growing that these patterns generate.

RM for multiple objectives (5^{th} column) is a trend for many-core systems since it is necessary to consider other metrics besides performance such as reliability, security, and energy [SDMI17, RJD17]. The RM that address timing-constrained applications [YSH14, RHK+15, MKP15, DAHM16, CPMC17] are multi-objective if the RM optimizes the system for another goal besides performance. If the applications do not have time constraints, the RM can penalize the performance to optimize a conflicting metric such as energy, as reported in

Hanumaiah et al. [HV14]. The RM presented in Das et al. [DAHM16] optimize energy and temperature simultaneously. This Thesis presents two multi-objective RMs. The first one is an energy efficient RM for timing-constrained applications. The second one can adapt many-core clusters to run the applications prioritizing energy or performance.

Another issue is the use of dynamic workloads (6^{th} column) where applications with unknown requirements can arrive and leave any time in the system. Although design-time evaluation of the application set returns better decisions than runtime one, design-time heuristics are not suitable for large systems and dynamic workloads [SSKH13]. Run-time RM also needs to provide the adaptability to deal with unpredictable events such as power violations, and network traffic. Nevertheless, some related works develop heuristics for RM according to a design-time analysis of the application set [RTGM13, MKP15, KP15, DAHM16, OA17]. Moreover, multi-objective RM increases the design search space, so that if RM requires a design-time evaluation of the application set, this evaluation for various objectives may lead to the pruning of the design space to maintain a low complexity [SDMI17].

Regarding system scalability (7^{th} column), top-down RMs of [MPV+13, YSH14, ZH16] present a small number of cores, ten cores or less, and no scalability evaluation. The top-down approach of Das et al. [DAHM16] implement the RM at the application layer to guarantee scalability. The clock-accurate model of Bogdan et al. [BMJ13] is not scalable. This Thesis targets many-cores with dozens of cores and scalability. Rahmani et al. [RHK+15] work is close to this Thesis, but large system sizes require cores grouped into clusters and distributed or hierarchical managed [CMMM13] as well as low-level aspects like observing and power data generation are ignored. Unlikely Pathania et al. approach [PKS+17], which requires no observing in the Probabilistic RM, the bottom-up approaches do not address the overheads related to the traffic congestion in large systems.

The allocation of multiple tasks/threads per core (8^{th} column) reduces the energy in energy efficient RM [MOSM15, CPMC17] due to the reduction of network traffic and the number of cores running tasks. Besides, the task-to-core mapping shrinks the computational complexity of mapping algorithms by bypassing the possibility of mapping multiple tasks in the same PE. Furthermore, multi-task enables more resources (Definition 6) in the many-core, expanding the number of tasks the system can run simultaneously.

The last row of Table 2.1 summarizes the contributions of this Thesis. The comprehensive approach is an advantage compared to related works because of the complexity of coordinating the actuation according to heuristics and because multiple actuations have more potential to lead to better results [ZJG16]. The top-down modeling assumed in this Thesis allows the evaluation of large systems without abstracting low-level aspects or overheads inherent to observing and actuation. The last four columns in Table 2.1 groups together the upcoming trends [SDMI17] in the RM for dynamic workloads: hierarchical approach (related to scalability), multi-objective decisions, and consideration of communication and computation loads (multi-task is used with this purpose).

3. OBSERVING

Many-core management requires accurate, stable, and reliable system information at runtime. Create a self-awareness system by observing the multiple layers of the system stack is essential for an effective Resource Management (RM) because the decision algorithms rely on the quality of the input information as illustrated in Figure 1.5. Besides that, a reliable observing infrastructure is crucial for system design and exploration.

This Chapter describes the development of the Observing infrastructure adopted for this Thesis. Due to dark silicon issues, any RM needs to be aware of power-related data, so it is necessary a characterization step to integrate a power model into the observing scheme, even for real hardware [WDH+17].

The observing methodology presented herein requires two descriptions of a many-core with the same functionality: (*i*) synthesizable VHDL for characterization purposes; (*ii*) RTL (Register Transfer Level) SystemC, enabling the simulation of systems with dozens of PEs (Processing Elements).

The contributions of this Chapter include:

- A power characterization method for the processor and the NoC (Network-on-Chip) to consider the static and dynamic power dissipation (the logic synthesis of the hardware components guarantee the accuracy of the model);
- Integration of the processor and NoC power/energy model along with a memory power/energy model to enable the complete PE characterization;
- Comparison of the proposed characterization model with a state-of-the-art power model;
- The design of hierarchical observing from the characterization flow;
- The multi-layered sensing infrastructure.

Section 3.1 reviews observing approaches. Section 3.2 presents the method to characterize power and energy of NoC-based MPSoCs from an RTL description. Next, Section 3.3 describes how to integrate the characterization measurements into high-level models to generate observing data for all hierarchy levels with low hardware and traffic overheads.

3.1 Overview of Observing Methods for Many-core Systems

According to Matthew et al. [WDH+16], power modeling can adopt two approaches. *Bottom-up* approaches use theoretical features of technology and transistors to model the

power (time and area as well). *Top-down* approaches require either a real hardware or cycle-accurate description of the system to model the power. *Bottom-up* power estimators are generic but they lack accuracy due to specification errors and abstractions. *Top-down* power estimators are more accurate and can be validated, but they target a specific hardware implementation. Besides that, *top-down* approaches are not suitable for large simulations due to long simulation times.

Even though developing a fast and accurate system characterization is still considered an open problem [BEG+15], some works try to address it. A learning-based power and performance estimation relies on a design-time phase for training the predictor model [ZJG16]. Another strategy to cope with the power modeling is an automated statistical methodology for power modeling [WDH+17]. Despite these approaches are very fast and accurate as well as validated for real hardware, they are not suitable for NoC-based many-core systems because they are both based on task phases, so they do not consider the traffic disturbances on the applications. An RM with statistical power capping claims observing task phases is not required when a large number of tasks are running on many-core systems [PKS+17]. Despite the fact that the communication between tasks and manager is just for initializing and finishing, no observing structure prevents both adaptability and any task-driven actuation such as task migration and multitask.

In the context of RM, another gap in the literature concerning observing approach are the *top-down* power models. Usually, *top-down* RMs abstract how the power-related data is obtained, computed and transmitted at runtime [RTGM13, KRS+14], or it is assumed the power-related data is ready to be used [HRW+14, YSH14, BMJ13]. The assumption that observing data is available on *top-down* models abstracts the complexity of the observation in real systems.

In *bottom-up* approaches, the system manager applies a given actuation policy when the observing data is available, but this information (when available) in real systems is performed at system level [HV14] or restricted at the cluster level [MPV+13], so a previous characterization step is required. As a consequence, the power-related information can be available in a different scheme on real systems than expected by the model, as reported in [HV14]. The lack of characterization limits the management or adds steps of remodeling the model parameters.

For cycle-accurate simulators using RTL descriptions, counters can be included to provide the data for power-related calculation. The manager should consider the observing epoch and the time required for actuation to take effect to keep the real behavior close to expected by the model [HV14]. Specifically to NoC-based many-cores, the epoch is a relevant property to avoid unnecessary congestion due to observing transmission.

3.2 Power Characterization and Energy Estimation

This Section presents a general method to characterize the energy and power parameters for the main PE modules. The power due the DMNI is not considered because it is a small module compared to the processor, router, and memory. The method to estimate power and energy is general because it is based on a calibration process to define the energy/power values. The characterization flow employs the synthesizable VHDL description of the reference platform.

3.2.1 Processor Characterization

The process of characterizing the processor for energy and power relies on calibration. Initially, the instruction set is divided into classes [TMW94]. The goal is to obtain measures of *energy per instruction* and *power per instruction* for all classes, as well as presenting these values in parcels of leakage and dynamic power. The processor power characterization comprises five steps [MSC+14]:

1. Group the instruction set into classes. For each instruction class, an assembly program is written with the instructions of the class in such a way to use all processor registers. The goal of the assembly programs is to maximize the switching activity of the processor modules (e.g. ALU, registers) by generating an important Hamming distance between the results (Figure 3.1).

2. RTL simulations for each assembly program count and trace the number of executed instructions and the number of clock cycles to execute the programs. Table 3.1 shows the instruction classification as well as the data obtained from the RTL simulations. The generated code for branches and jumps present a smaller percentage of instructions belonging to these classes, due to the insertion of *nop* instructions. The influence of the *nop* instructions is accounted to compute the energy for these two classes.

3. Logic synthesis of the processor for a given technology generates a gate level description and an SDF file (Standard Delay Format - annotated delay data and timing checks file obtained after logic synthesis). The timing constraints (the frequency of PE is supposed to run) are defined at this step.

4. Switching activity annotation from netlist simulation. The netlist obtained in step 3 is simulated using the same assembly programs of step 1. This simulation generates the switching activity at the gate level, in a VCD (Value Change Dump) or TCF (Toggle Count Format) file, as well as traces for functional validation with step 2.

5. Power analysis of the switching activity files provides accurate and reliable measurement of dynamic and static power since the results come from a netlist synthesized for a given technology. Table 3.2 shows the power and energy obtained from the netlist simulation.

```
int main(){
   int i;
   for(i=0;i<600;i++)
   {
     asm volatile
        ("   addi $8, $0,  0xffff\n"
         "\t addi $9, $0,  0x0001\n"
         "\t lui  $1, 0xA5A5\n"
         "\t addi $10,$1, 0xA5A5\n"
         "\t addi $1, 0x5A5A\n"
         "\t addi $11,$1, 0x5A5A\n"
         "\t lui  $12, 0xF0F0\n"
         "\t addi $12,$1, 0xF0F0\n"
         "\t lui  $13, 0x0F0F\n"
         ....
         "\t addu $17,$26,$17\n"
         "\t sub $17,$26,$17\n"
         "\t subu $17,$26,$17\n"
         "\t addiu  $24,$25,0x0810\n"
       );
    }
}
```

Figure 3.1 – Assembly code snippet used for processor characterization.

Table 3.1 – RTL simulation results obtained from instruction set classes.

Class	# of executed instructions	% of executed instructions	Cycles per instruction
arithmetic	110,456	98.88	1
logical	108,656	99.40	1
shift	46,430	99.50	1
move	77,030	97.36	1
nop	50,940	99.69	1
branches	220,030	47.72	1
jumps	220,030	45.45	1
load-store	60,058	95.91	2

A set of codes with minimal switching activity was written to better evaluate the effect of the Hamming distance in the assembly codes. It was observed that the switching activity could induce up to 30% of average power concerning to the results obtained in Table 3.2. Thus, assembly codes with maximum Hamming distance are employed because the characterization must be general (i.e., used for any benchmark), and it does not capture

the Hamming distance between operations belonging to different instruction classes. As the assembly programs consider instructions belonging to the same class, a set of bits (6 for the MIPS processor) does not switch.

Table 3.2 presents the characterization results for each instruction class, considering the Plasma processor (MIPS architecture), in a 65-nm technology, 1.1V@4ns.

Table 3.2 – Power characterization results and energy estimation for each instruction class of the processor. Library CORE65GPSVT (65nm), 1.1V, 25ºC (T=4ns).

Class	Avg. Power (mW)		Energy per inst. (pJ)	
	Leakage	Dynamic	Leakage	Dynamic
arithmetic	0.452	5.894	1.808	23.58
logical		5.176		20.70
shift		4.940		19.76
move		4.768		19.07
nop		3.331		13.32
branches		5.723		31.70
jumps		4.175		18.56
load-store		5.507	3.616	43.15

The fifth step of the characterization flow provides the power values while the energy is given as follows:

$$E_{class} = P_{class} * CPI * T \tag{3.1}$$

where: P_{class} is the average power for a given instruction class, CPI is the number of cycles per instruction (last column of Table 3.1), and T is the clock period (4 ns). Equation 3.1 works for both static and dynamic power.

From the energy per instruction class, the total energy consumption and, consequently, the power dissipation can be estimated for the processor as follows:

$$E_{processor} = \sum_{i=0}^{n_{class}} n_{instructions_i} * E_{class_i} \tag{3.2}$$

$$P_{processor} = \sum_{i=0}^{n_{class}} n_{instructions_i} * P_{class_i} \tag{3.3}$$

where: $n_{instructions}$ is the number of executed instructions for a given class, E_{class_i} is the energy per instruction for a given class, CPI_{class} is the CPI for a given class.

It is important to mention that if the processor has no task to execute or is waiting for data from another task in another processor, the processor enters in the hold state while consuming only static power.

Finally, the simulation of different benchmarks enables the validation of the calibration process. Table 3.3 summarizes the validation of the processor calibration for seven benchmarks. The energy and power *estimated* results come from applying equations 3.2 and 3.3 in a trace file generated by an RTL simulation. The *measured* results are extracted from power reports of netlist simulation. The errors presented on latest columns are mainly due to switching activity (Hamming distance), and pipeline stalls due to dependence between operators.

Table 3.3 – Processor Energy Estimation Error.

Benchmark	Estimated Results		Measured Results		Error (%)	
	Energy (nJ)	Power (mW)	Energy (nJ)	Power (mW)	Energy	Power
binarySearch	1749.79	5.46	1739.986	5.41	1%	1%
bubble	1783.13	5.77	1761.630	5.64	1%	2%
compress	4555.17	5.51	4213.906	4.99	8%	10%
crc	2965.24	5.68	3257.885	6.24	-9%	-9%
fft	775.92	5.21	841.601	5.1	-8%	2%
switchCase	5689.21	5.20	5546.044	5.06	3%	3%
usqrt	2032.50	5.52	1985.088	5.39	2%	2%

The data presented in this Section was obtained at room temperature, 25^{o}C. It is worth mentioning that the temperature mainly affects the static consumption due to the leakage current which increases exponentially with the temperature (up to 9 times at 125^{o}C). The effect of the temperature on the dynamic power is smaller than the leakage power (+3% at 125^{o}C) because it corresponds to the charge and discharge of capacitors. Using the method herein described it is possible to characterize the components for other temperatures.

3.2.2 Router Characterization

The main router internal components include input buffers, crossbar, and control logic (responsible for arbitration and routing). Therefore, the router characterization requires excite all internal components, and provide a payload with an important Hamming distance between flits to induce a large switching activity in the router logic gates. The characterization process of the router is similar to the processor characterization. The router power/energy characterization comprises four steps [MSC+14]:

1. Traffic generation for maximizing the switching activity of all input buffers. The power dissipation of a router is a function of the reception rate in the input buffers [OGI+09]. The reception rate is the relationship between the traffic rate and the available bandwidth of the physical link. Figure 3.2 presents the traffic flows to characterize the 5-port central router in a 3x3 NoC. Each traffic flow source injects 1,000 32-flit packets, with a

Hamming distance between flits superior to 80%. The method herein proposed creates 6 test cases. The injection rates vary from 0% (idle) to 50% of the link bandwidth. For example, for an injection rate equal to 50%, for a 32-flit packet, each packet is injected at 64 clock cycles.

2. Logic synthesis of the instance of a 5-port router to generate a netlist and an SDF file. To account the energy consumed in the links (wires), the output load settings in the timing constraints file consider the wire capacitance between two routers (1 mm long, metal 5, 200fF).

3. Simulation of a 3x3 NoC with the replacement of the RTL description of the central router by the netlist obtained in step 2 (Figure 3.2). The simulations of each traffic scenario from step 1 produce the switching activity at the gate level (VCD or TCF files) for a given injection rate.

4. Power analysis of the switching activity file. A particular feature of the PE is the injection/reception of the packets in burst mode. The DMNI makes the bridge between the router and the memory. Thus, packets are transmitted with an injection rate equal to 100% without blocking the processor. Despite several injections rates used in the characterization method, the characterization adopts two rates: 100% - active mode, and 0% - idle mode. The 100% rate derives from the extrapolation of the available injection rates.

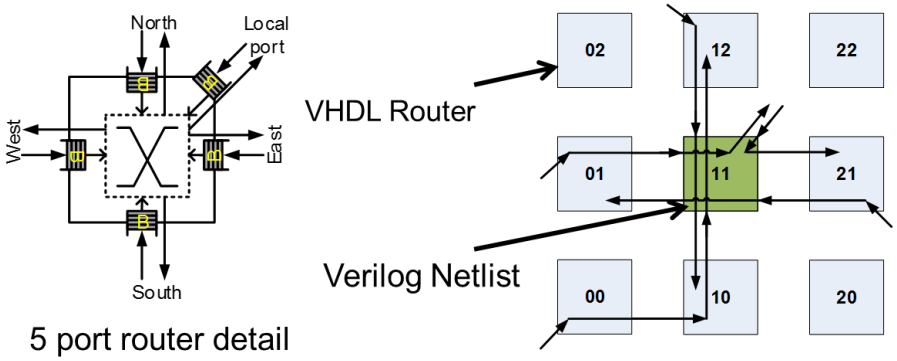


Figure 3.2 – A 3x3 NoC traffic scenario for the 5-port router characterization.

Table 3.4 presents the power characterization of the router for the two traffic rates. The second column presents the dynamic average power consumption for one buffer. The third column, combinational logic, corresponds to the remaining parts of the 5-port router ($P_{leak_{router}}(n_{ports})$). The last Table column presents the router leakage power.

Let E_{active} be the dynamic active energy to receive one flit (with one active buffer) and E_{idle} be the spent dynamic energy in idle mode, which are derived from Table 3.4 as follows:

$$E_{active} = [(n_{ports} - 1) * P^{idle}_{buffer} + P^{active}_{buffer} + P^{active}_{comb}] * T \quad (3.4)$$

Table 3.4 – Router Average Power. Library CORE65GPSVT (65nm), 1.1V@4ns, 25ºC.

Traffic Rate	One buffer	Combinational Logic	$P_{leak_{router}}(n_{ports} = 5)$
0% - idle	364.64 µW	575.64 µW	223.08 µW
100 % - active	755.56 µW	2655.25 µW	

$$E_{idle} = [(n_{ports}) * P^{idle}_{buffer} + P^{idle}_{comb}] * T \quad (3.5)$$

where: n_{ports} is the number of ports of the router, $P^{idle}_{component}$ is the average idle power of a given component, $P^{active}_{component}$ is the average active power, and T is the period used to characterize the router.

3.2.3 Memory Characterization

In general, a memory generator tool provides memories as black boxes in the technology design kit without an RTL model. The CACTI-P [LCA+11] tool models distinct memory types and generates estimations like access time, silicon area and power. This tool also supports Dynamic Voltage Scaling (DVS). CACTI-P allows the characterization of the energy consumption of the PE local memory, configured as a 64kB scratchpad memory, with two ports. Table 3.5 presents the characterization data produced by CACTI-P. CACTI-P tool allows some technology options (like cell and peripheral circuits) that are different from the industrial libraries of standard cells previously deployed. The technology settings are calibrated to generate consistent result when comparing with processor and memory. In Table 3.5, the access time corresponds to the period used to characterize the processor and the router (4 ns), $P_{leak_{memory}}$ is the leakage power, E_{load} is the dynamic read energy per access, and E_{store} is the dynamic write energy per access.

Table 3.5 – CACTI-P Report for a Scratchpad Memory (65nm, 1.1V, 25ºC).

Access time	$P_{leak_{memory}}$	E_{load}	E_{store}
3.98 ns	0.66 mW	67 pJ	38 pJ

3.2.4 Comparison with State-of-the-Art Tool

Among the bottom-up approaches for power estimation, McPAT [LAS+09] is the mostly used [HRW+14, MKP15, RTGM13, PKM+14]. McPAT is power, area, and timing modeling framework for many-core systems. The main advantages of McPAT are the fast design

space exploration and the XML interface. The user can specify systems using an XML template as well as the integration with others simulators such as HotSpot [HGV+06]. A set of experiments with benchmarks was carried-out with the goal of evaluating if McPAT could be adopted to model the processor power and replaces the method previously described.

Figure 3.3 illustrates the power characterization results of the processor for McPAT and the proposed approach (Section 3.2.1). The X-axis corresponds to the benchmark, and the Y-axis the power. The orange curve represents results for the proposed method, the black multiplies the orange values by 100, and the blue curve corresponds to the McPAT result. Two main considerations from these results are as follows:

- Power results obtained with McPAT are, on average, 50 times larger than the ones obtained from the proposed characterization methods;

- The trend of the curves is not the same (compare the black and blue curves).

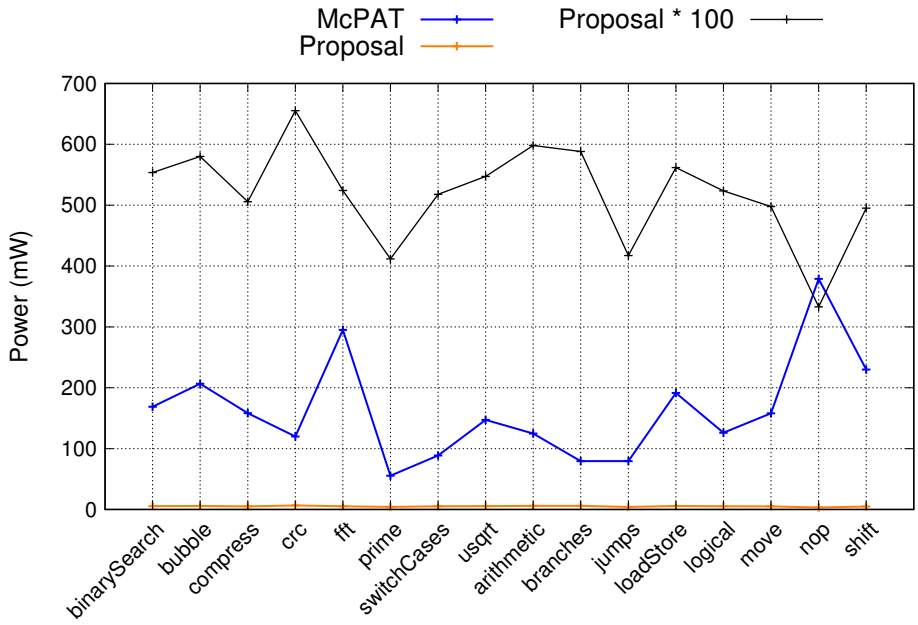


Figure 3.3 – Proposed top-down power characterization versus McPAT bottom-up power characterization.

Concluding, McPAT is inappropriate to be used the present Thesis because McPAT cannot capture the different instruction classes and the Hamming distance between instructions due to McPAT generates different results from the proposed characterization method in scale and trend. In the Authors opinion, this tool may be used to evaluate abstract models, in early design stages, where accuracy is not the main goal.

3.3 Observing

The characterization flow (Section 3.2) enables the estimation of the average power and the energy consumption at runtime. This Section details the hierarchical observing infrastructure with the focus on the energy consumption (Section 3.3.1), PE utilization (Section 3.3.2) and RT constraints (Section 3.3.3). Section 3.3.4 discusses the impact of the observing messages according to the periodicity to transmit them to the manager PEs.

Figure 3.4 overviews the 3-level hierarchical observing. The SPs implement the lowest level of the observing scheme. Each SP senses the observing data at the PE level and sends the *sample per PE* periodically to the CM of its cluster. At the *cluster level*, the CM receives the observing data of its corresponding SPs and updates look-up tables with the received samples to compute the cluster data. The intra-cluster look-up tables size at the *cluster level* corresponds to the number of SPs per cluster. In particular, the SPs of the GM cluster send the *sample per PE* straight to the GM. Finally, the CMs send to the GM the *sample per cluster* when the CM received all samples from the SPs. Similarly to the *sample per cluster*, as the GM receives energy data from all CMs and SPs of its cluster, the GM observes the system by updating inter-cluster look-up tables. Accordingly, the inter-cluster look-up tables size at the *system level* corresponds to the number of clusters.

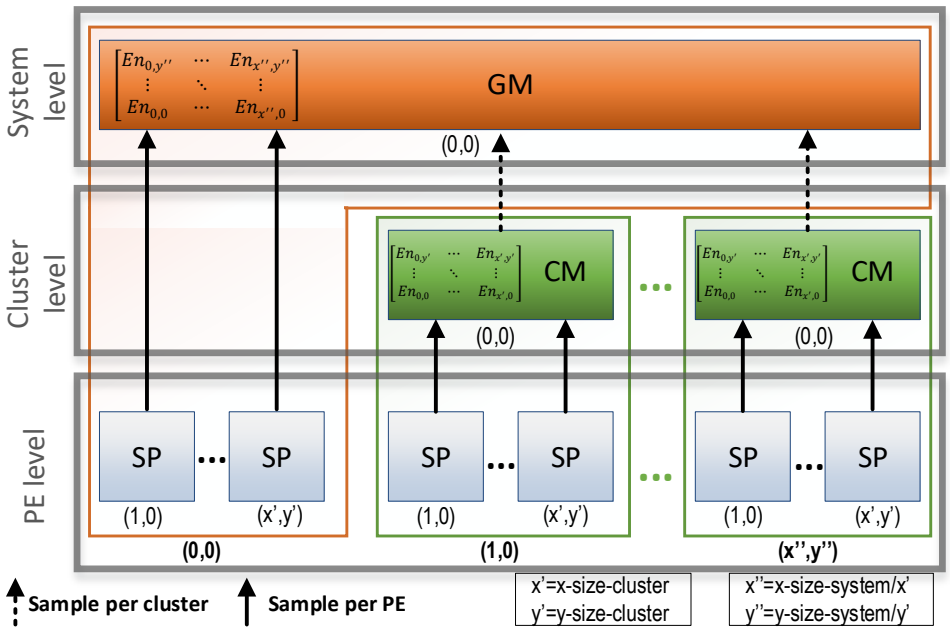


Figure 3.4 – Hierarchical observing scheme for many-core systems [MRM15].

3.3.1 Energy and Power Observing

The hardware requirements to enable observing comprise the addition of a set of registers into the PE for counting events: (*i*) n_{class}, number of executed instructions for a given instruction class; (*ii*) $cycles_{active}$, number of clock cycles the router transmit flits; (*iii*) $cycles_{total}$, number of clock cycles executed by the PE. At the end of an epoch (Definition 7), the CPU is interrupted, and all counters are reset to zero.

Definition 7. *Epoch* - an execution time interval defined by a periodic hardware interruption where SPs report intra-cluster observing data to its CM, or CMs report inter-cluster observing data.

The software requirements to enable observing include functions to read the counters at the end of the epoch, and functions to estimate the PE energy. An estimation function executes periodically in the OS (Operating System) of the SP to compute power and energy of the PE into the epoch (average power in the epoch multiplied by the epoch time). Let E_{proc} be the dynamic processor energy, E_{mem} be the dynamic read/write energy per access, E_{router} be the dynamic router energy, and E_{leak} be the energy from leakage power of the whole PE, which is defined as follows:

$$E_{proc} = \sum_{i=0}^{n_{classes}} n_{instructions_i} * E_{dynclass_i} \tag{3.6}$$

$$E_{mem} = n_{instructions_{load}} * E_{load} + n_{instructions_{store}} * E_{store} \tag{3.7}$$

$$E_{router} = E_{idle_{router}} * (cycles_{total} - cycles_{active}) + E_{active_{router}} * cycles_{active} \tag{3.8}$$

$$E_{leak} = [P_{leak_{proc}} + P_{leak_{mem}} + P_{leak_{router}}(n_{ports})] * cycles_{total} * T \tag{3.9}$$

$$E_{PE} = E_{proc} + E_{mem} + E_{router} + E_{leak} \tag{3.10}$$

GM and CMs receive from their SPs the *energy per PE*, E_{PE}, through an observing message and then update the look-up tables.

Figure 3.5 presents power and energy data obtained from the hierarchical observing for a set of tasks executing simultaneously. The first row of the graphs presents average power results. The second row of the graphs shows the accumulated energy, i.e., the sum of all past instant energy data (Equation 3.10). The setup used on the graphs is a 4x4 many-core divided into four 2x2 clusters, running twelve applications. This size of many-core is chosen to make the graphs readable. The intra-cluster epoch is a hardware interruption set to 50,000 clock cycles. Meanwhile, the CMs send inter-cluster samples when receiving all intra-cluster samples.

The first column of the Figure 3.5 presents the power and energy of an SP at the *PE level* – **PE 0x1**. The red vertical lines signalize when a task is allocated while the blue vertical lines indicate a terminated task. The task events highlight the disturbances on the power curve in the second half of the simulation and indicate, for this example, a near steady power throughout the first half of the simulation. The *energy per PE* graph shows that the SP spent most of the energy at the first half of the simulation because the SP is executing a workload near to 100%. After some task events at the second half of the simulation, the proposed observing can sense those variations since the energy increases slower. At the end of the simulation, the SP spent 127 uJ.

The second column of the Figure 3.5 shows the observing at the *cluster level*. The power graph gives the CM view of the power in all intra-cluster SPs. For instance, all SPs at cluster 1x1 have a higher average power at the first half of the simulation. If a decision algorithm was running, an actuation policy could avoid all SPs working at high power simultaneously.

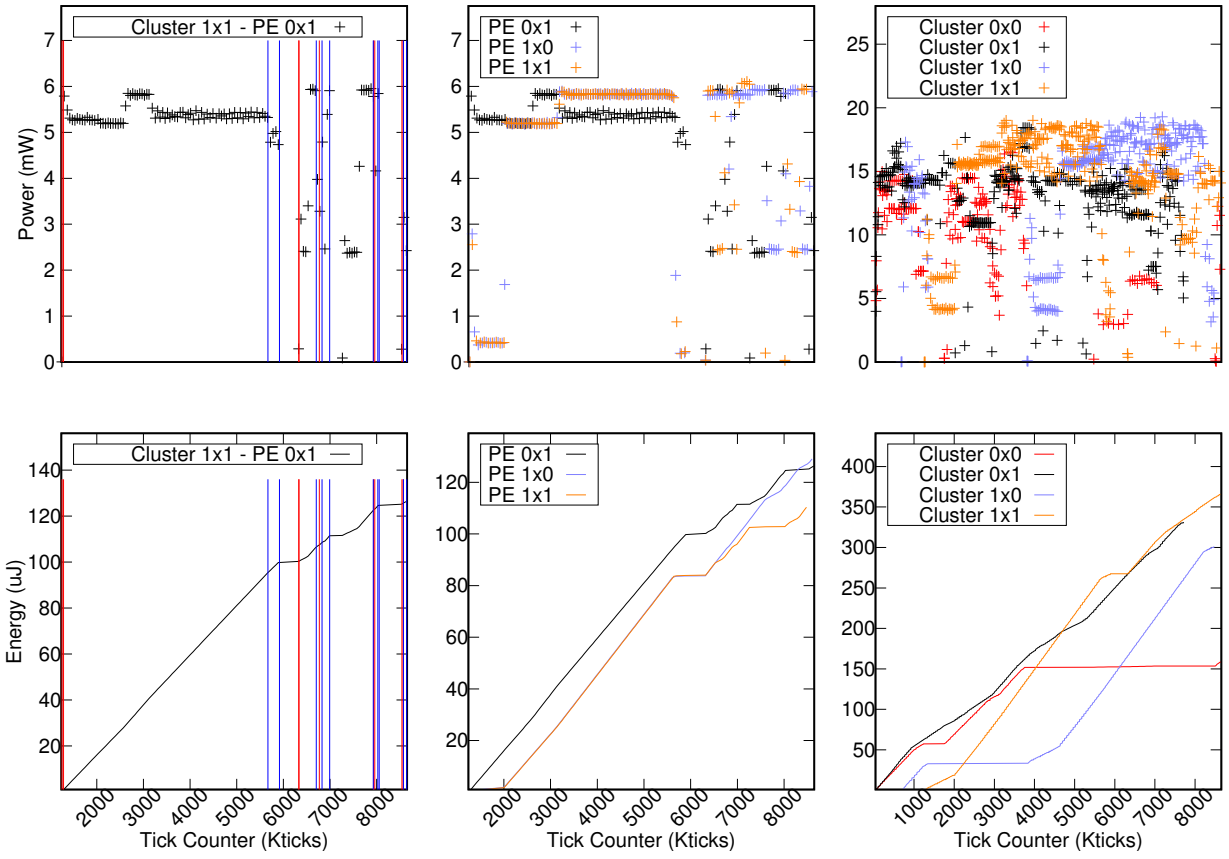


Figure 3.5 – Hierarchical observing of the many-core.

The *system level* view is shown in the last column of the graph. At the system level, it is possible to check that the 4x4 system takes around 90,000 Kticks (36 ms) to run all the tasks. Moreover, all clusters receive tasks to execute but details such as task events, resource, or which SPs are currently running, are abstracted on the system level.

The energy between the clusters is not balanced but, for example, if a decision heuristics was running the cluster 0x0 could execute more tasks to balance the energy.

It is worth to note that, the proposed observing scheme allows a complete sensing of the many-core in any level of the hierarchy. Figure 3.5 only depicts graphs for one SP, but similar graphs are generated for all SPs. Similarly, graphs at cluster level for all clusters can also be depicted. The observing herein proposed allows a full exploration of design space for RM. All graphs presented along this Thesis are generated from this observing scheme.

3.3.2 PE Utilization Observing

Besides the energy and power information, sensing the processor and router utilization is also a desirable feature of the observing scheme. For example, it allows identifying CPU-intensive and communication-intensive PEs, tasks or applications. Figure 3.6 shows an example of processor utilization view of a PE. The ratio of the registers $cycles_{active}$ and $cycles_{total}$ derives the processor utilization.

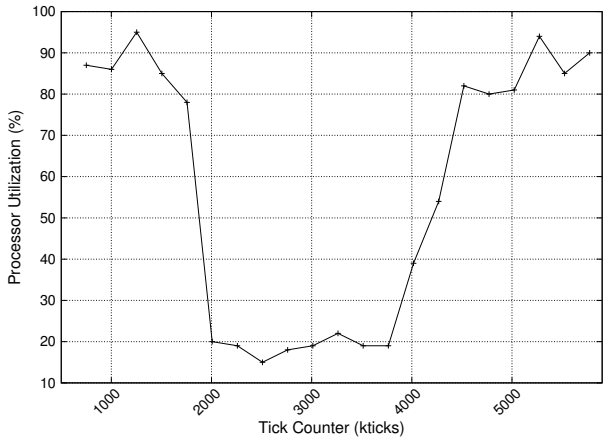


Figure 3.6 – Processor utilization observing.

Besides the processor, some RMs take decisions based on NoC traffic profiles [RHK+15, CPMC17]. These traffic-aware RMs usually look for network bottlenecks by identifying routers under high congestion and/or high injection effect. The observing of the input buffers (Figure 3.2) is the method for sensing the traffic effect in the routers [RHK+15]. The hardware requirement is a buffer utilization counter that measures the average message flow for each link. The buffer utilization measure of the local port corresponds to the *router injection* information as well as the buffer utilization measure of the non-local ports (i.e., north, south, east, and west) corresponds to the *router congestion* information. Figure 3.7 shows an example of the router injection and router congestion generated in an SP.

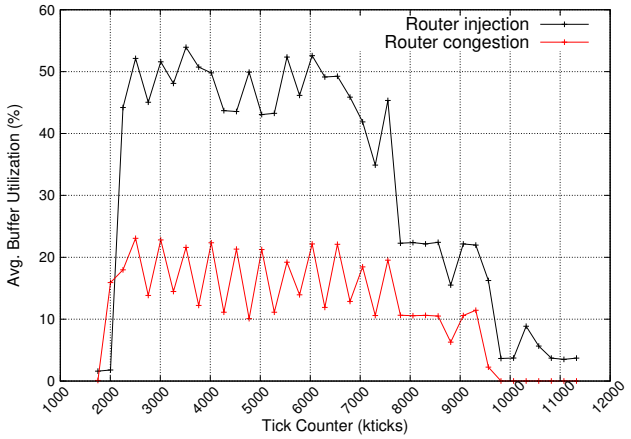


Figure 3.7 – Router injection observing and router congestion observing from the input buffers utilization.

3.3.3 Application Observing

Up to this Subsection, the observing is restricted to the system, cluster, router and PE layers and implies in reading some available counters to generate the sensing information. However, observation at the application layer requires software routines into the user applications to generate and transmit the observing data. The observation at the application layer allows the decisions and actuation concerning an application. In the context of soft real-time (RT) applications, observing the timing properties is essential to guarantee Quality of Service [RM16]. The software routines correspond to system calls (*syscalls*).

RT applications are mostly iterative applications like multimedia ones. In RT applications, all tasks have a predefined time to execute one iteration. Consequently, the expected execution time of the RT application is the sum of the execution time of all non-parallel tasks. Unpredictable events, such as network congestion and non-preemptive interruptions, may delay the execution time of the tasks (and applications). For any RT application is associated a constraint (i.e., the maximum execution time that the application has to finish one iteration), called hyper-period. Therefore, the execution time of the applications has to be monitored to notify the manager the occurrence of hyper-period violations.

Figure 3.8 presents a code snippet for the last task of the application task graph. The SP handles two *syscalls*:

- **RealTime**: send the application hyper-period and the task execution time to the operating system running in the SP, and to the CM (line 2 of Figure 3.8). The task execution time corresponds to the number of clock cycles the scheduler execute one iteration of the task.
- **PeriodMonitoring**: send the current time to the CM. The first task of the application includes this syscall at the beginning of an iteration. The last task includes this syscall

at the end of the application iteration (line 7 of Figure 3.8). Therefore, the two *Period-Monitoring* calls allow the CM to check the constraints because it captures all delays the tasks get in the iteration.

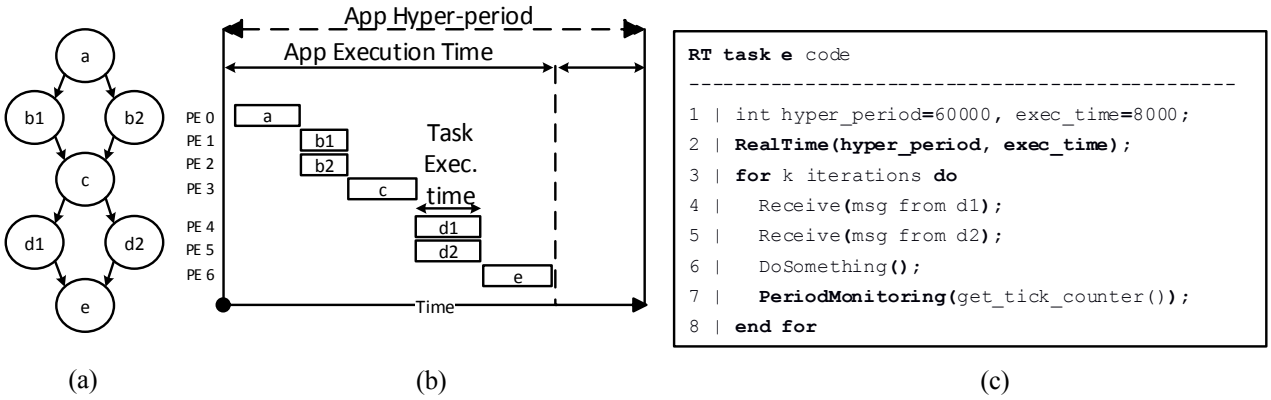


Figure 3.8 – Application layer observing. (a) RT application task graph. (b) Scheduling of the RT application. (c) Code snippet for the last RT task. Syscall *RealTime* sets constraints. Syscall *ReportPeriod* registers the end of each hyper-period

3.3.4 Epoch Considerations - Epoch Delay Observing

It is essential to consider the impact of the observing messages into the NoC traffic. Since the epoch is configurable at design-time, it is necessary to define a trade-off between the epoch period and the traffic from observing messages. The epoch has to be as minimal as possible to improve the accuracy and speed of the decisions, without congesting the NoC. An evaluation of distinct epochs [Cas17] for the same reference many-core states an epoch around 250 Kticks as the ideal.

Despite the fact that this evaluation shows a good range of epoch periods, the traffic impact due to observing messages relies on how much load is currently on the system. The *Dif Time* measures the time between the injection and the reception of each observing message. All messages include a timestamp flit, so the PE receiver compares the timestamp from the message with the received time to compute the message delay. Therefore, the proposed hierarchical observing can also measure the delay of this own messages.

Figure 3.9 shows the *Dif Time* for the same scenario of Section 3.3.1. The epoch for intra-cluster observing messages is 50 Kticks while inter-cluster messages are sent every time the CM receives all intra-cluster observing messages. The y-axis is the *Dif Time* and the x-axis is the execution time. The labels at each point correspond to SP position in the cluster. Most of the observing messages at PE level require 100 clock cycles (ticks) to be transmitted (94.94%), while some messages may be delayed due to congestion, being the worst-case around 3,600 ticks (7% of the epoch) for this cluster. As the transmission

of the inter-cluster observing messages occurs when all active SPs transmit their observing messages it is desirable a small *Dif Time* to avoid delays in the system management. The observed average transmission time of an inter-cluster observing message is 300 ticks, according to the cluster size.

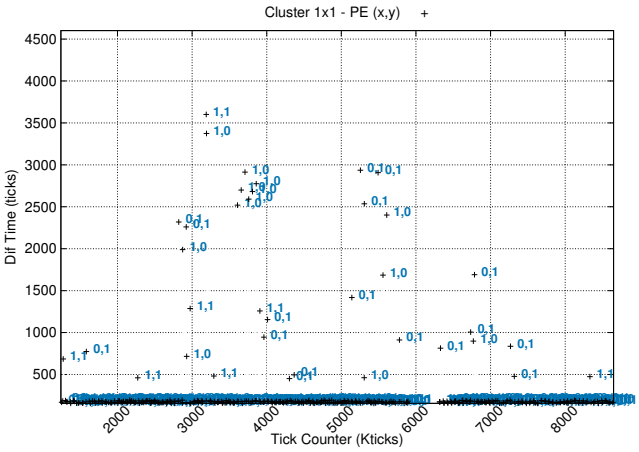


Figure 3.9 – Arrival delay of the observing messages (*Dif Time*) considering the same test case of Figure 3.5.

3.4 Final Remarks

This Chapter proposed a hierarchical observing method for many-core systems. Initially, the Chapter presented an overview of the observing approaches. The sensing of the observing data is obtained from different counters inserted into the hardware. In particular, the sensing of energy and power requires a previous characterization step. Besides the presentation of the power characterization method, the Chapter presented a comparison with a widely used characterization tool.

According to the presented results, the hierarchical observing scheme provides the multi-layer sensing to support the development of RM. The proposed observing method does not assume the existence of any sensors in the hardware. Also, the required hardware is restricted to counters registers so that this method is low intrusive and easily adaptable to different processors. Related works [HV14] that implement RM on real systems report error and noise on the sensors. Thus, the proposed observing can be an alternative or a complement to these methods. Besides, energy or power data at the PE level is usually not available on real systems even when a policy management is applied at the PE level.

4. ACTUATION

The management of many-core employs an actuators set, also called as knobs, to meet the settings that the Decision proposes such as assign tasks for processing, manage the power, among others. Figure 4.1 classifies the actuation methods according to its implementation, software or hardware.

Definition 8. *Hardware Actuation* - an actuation that makes changes on the physical system settings.

Definition 9. *Software Actuation* - an actuation that makes changes on the system configuration, related to the allocation of resources to applications or tasks.

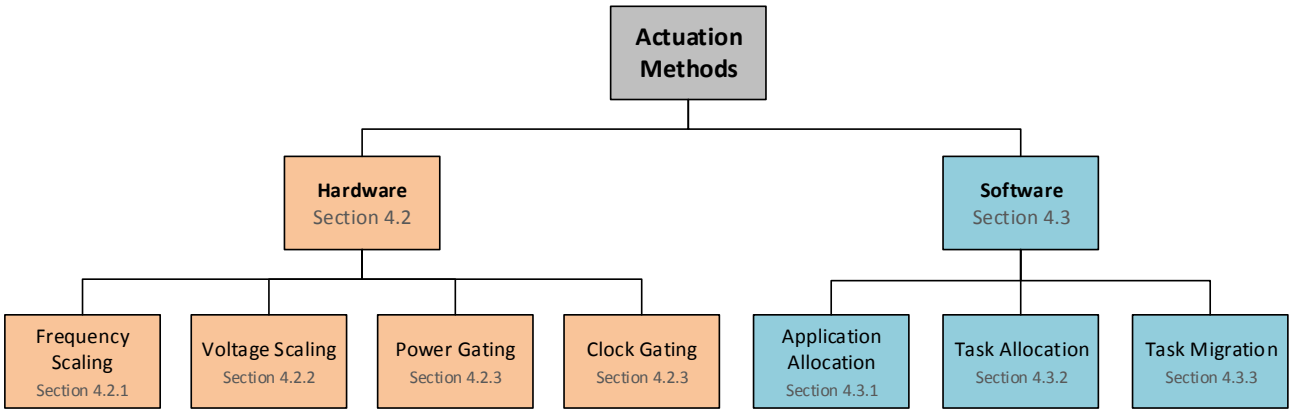


Figure 4.1 – Classification of actuation methods, and the actuators set adopted in this Thesis.

Figure 4.1 also lists the actuation methods adopted in this Thesis, and the corresponding Section describing each one. The hardware actuation methods include:

- *Frequency Scaling* - technique used to increase or decrease the frequency of a hardware component.
- *Voltage Scaling* - technique used to increase or decrease the voltage of a hardware component.
- *Power Gating* - technique used to shutdown the power supply of an idle block of a circuit [JMSN05].
- *Clock Gating* - technique used to disable the clock switching to stop unnecessary gate activity [WPW00].

Communication protocols implement the functionality of the software actuators. Any software actuation includes a set of messages, each one related to a specific service to implement the behavior of the given actuation. The software actuation methods include:

- *Application Allocation* - protocol used to select a cluster to execute an incoming new application, reserving the resources (Definition 6) required to allocate the application.
- *Task Allocation* - protocol used to assign a task of the incoming application to an SP with available resources (Definition 6).
- *Task Migration* - protocol used to transfer a task from one SP to another SP.

The system can set any actuator at runtime so that it enables system adaptability (Definition 4). Also, distinct settings are possible for each actuator in Figure 4.1. The cooperation of software and hardware actuators can bring advantages to the system management [ZH16]. However, dealing with this comprehensive actuators set is challenging. In general, the quantity of possible system settings increases exponentially as far as the number of actuators grows. As a consequence, a comprehensive actuators set increases the complexity of the Resource Management (RM). The Decision phase is in charge of finding the best system settings for a given workload. Meanwhile, the Actuation phase coordinates the system to follows the settings selected at the Decision state.

Two properties are relevant for the actuation methods: (*i*) the latency for enforcing an actuation, and (*ii*) the impact delivered by an actuator [ZH16]. Hardware actuators are usually fast, but they usually have a limited overall impact (resources, power, among others metrics) at the system point of view due to inherent hardware limitations (actuation at the PE level). On the other hand, software actuators are slower than the hardware ones to take effect, but they usually are more flexible and enforce a higher impact on the system. Concerning the resources impact and the latency, the actuators set is sorted to fit in the hierarchical approach.The higher the actuation impact and latency, the higher is its hierarchical level. According to the resources impact and actuation latency, the actuators set is organized as follows:

1. System level: application allocation;
2. Cluster level: task allocation, task migration;
3. PE level: dynamic voltage and frequency scaling (DVFS), power gating and clock gating (all hardware actuators).

Regarding the software actuators, application allocation and task allocation are correlated problems. Once the RM allows an application allocation, the RM also enables the allocation of all tasks of this application as well. Since a task graph composes one application, the task allocation is assumed as an inner problem of the application allocation. This Thesis distinguishes application allocation and task allocation as two instances of the same problem so that the first one is a system level problem and the second one is a cluster level problem.

This Chapter highlights are as follows:

- DVFS design at the PE level to support power management as well as guidelines for modeling others hardware actuators;
- Protocol for all software actuators;
- Evaluation of the power and resource impact of each actuator.

Section 4.1 overviews related works regarding the comprehensiveness of RM, i.e., how many features, including the actuators, the RM attends. Next, Section 4.2 presents the design and model of the hardware actuators. Following, Section 4.3 presents the protocol of the software actuators. Section 4.4 evaluates the power and resources impact of actuators methods. Section 4.5 concludes the Chapter.

4.1 Overview of Actuation Mechanisms in Resource Management

Combine a mix of software and hardware actuators can bring advantages of both of them to the system management [ZH16]. Table 4.1 summarizes related works concerning the actuators commonly found in many-core systems: (*i*) Applications Admission (AA); (*ii*) Task Allocation - TA; (*iii*) Task Migration - TM; (*iv*) Dynamic Voltage-Frequency Scaling - DVFS; (*v*) Power-Gating - PG; (*vi*) Clock-Gating - CG. Table 4.1 excludes some actuators for specific architectures. For example, a memory controller is relevant for shared memory systems. Besides, Table 4.1 also includes the level of the heterogeneity of the system, because the heterogeneity increases the complexity of the actuation and, consequently, the management.

Table 4.1 – Actuators mechanisms found in RM for many-core systems.

Proposal	AA	TA	TM	DVFS	PG	CG	Many-core
Muthukaruppan et al. [MPV+13]	✗	✓	✓	✓	✓	✗	Asymmetric
Bogdan et al. [BMJ13]	✗	✗	✗	✓	✓	✗	Homogeneous
Raghunathan et al. [RTGM13]	✗	✓	✗	✓	✓	✗	Homogeneous
Hanumaiah et al. [HV14]	✗	✗	✓	✓	✗	✗	Homogeneous
Lai et al. [LLW+13]	✗	✗	✗	✓	✗	✗	Homogeneous
Yu et al. [YSH14]	✗	✗	✓	✓	✗	✗	Heterogeneous
Maiti et al. [MKP15]	✗	✓	✗	✓	✗	✓	Homogeneous
Kapadia et al. [KP15]	✓	✓	✗	✓	✗	✗	Homogeneous
Haghbayan et al. [RHK+15]	✗	✓	✗	✓	✓	✗	Homogeneous
Zhang et al. [ZH16]	✗	✓	✓	✓	✗	✗	Homogeneous
Olsen et al. [OA17]	✓	✓	✗	✗	✗	✗	Homogeneous
This	✓	✓	✓	✓	✓	✓	**Homogeneous**

AA: Application Allocation; TA: Task Allocation; TM: Task Migration; PG: Power-Gating; CG: Clock-Gating

Concerning software actuators, AA is the less employed. Regarding the works considering AA as a previous problem of TA, such as our approach, Authors [KP15, OA17]

present frameworks for deciding the best number of tasks for a given application before the TA. In Rahmani et al. [RHK+15] work, the application enters the system if there are available processors, but can also be killed suddenly if the power overcomes the capping. Regarding TA, some works [MKP15, OA17, RJD17, KP15] assume the PE as one resource, while others admit the two or more tasks per PE [ZH16, MPV+13, HV14], like this proposal. Asymmetric and heterogeneous systems require distinct strategies for TA. To conclude the software actuators, TM enables runtime adaptability for task allocation into RM. However, some authors decided to turn the migration off due to the overhead costs and deal with the issues inherent to dynamic workloads using only DVFS [RHK+15], or change the level of parallelism of the application [KP15, OA17] instead of employing TM.

Once the tasks are mapped, DVFS is the mechanism to deploy power management in many-cores. The granularity of DVFS varies between PE level [BMJ13, HV14, RJD17, KP15, MKP15, ZH16, YSH14, RTGM13] (this proposal) and cluster level [LLW+13, MPV+13]. When no tasks are running, some works [RHK+15, MPV+13, RTGM13] assume PG to avoid energy wasting and reduce the average power. CG disables the clock switching of parts of the circuit to save dynamic power. Since CG is a widely used power actuator, works at Table 4.1 can employ it, but just Maiti et al. [MKP15] work mentions CG explicitly. PG is a power actuator suitable for PEs with no tasks due to the latency to wake-up the circuit, while CG is ideal for small periods of idle times throughout the execution of tasks. However, the manager can use DVFS at minimum levels plus CG when no tasks are running in case of absence of PG [LLW+13, HV14, MKP15].

Usually, Authors abstract the actuators because the decision is the focus. However, some related works consider previous design-time steps to capture actuators properties like latency and power to make the control aware of them [LLW+13, MPV+13, HV14] when taking decisions. This proposal also considers the overhead of the hardware actuators (Section 4.2). Ruaro et al. [RM17] work evaluates the costs of task migration for the adopted reference platform.

4.2 Hardware Actuators

This Section describes four hardware actuation methods (Figure 4.1) deployed in this Thesis. The model of the frequency scaling (Subsection 4.2.1) requires modifications on the original PE structure to cope with different frequencies and keep the accuracy regarding clock cycle. To guarantee realistic DVFS support, the model of the voltage scaling (Subsection 4.2.2) considers the hardware overheads (latency, energy), standard cells library characterized for distinct supply voltages provided by the foundry, and the delays inherent of the voltage scaling for establishing a correct DVFS protocol. The PG model is mirrored from other works when making comparisons instead of proposing a specific model for the refer-

ence platform. Meanwhile, CG is a native functionality of the reference many-core. As PG and CG are mechanisms commonly found in RM to deal with power cap issues, Subsection 4.2.3 discusses the adoption these actuators.

4.2.1 Frequency Scaling

Figure 4.2 illustrates the hardware modifications on the PE for frequency scaling support. The frequency scaling actuates only on the processor, memory, and DMNI. The main goal is to enable processors to work at different frequencies while the NoC transmits message by using the nominal frequency. The reason for transmitting message at the nominal frequency is to avoid PEs running at higher frequencies stall due to PEs running at lower frequencies. The new PE includes a Clock Generator, which creates the scaled frequency from the nominal frequency. Frequency domain line separates the blocks of hardware running at nominal frequency (blue color) and scaled frequency (gold color).

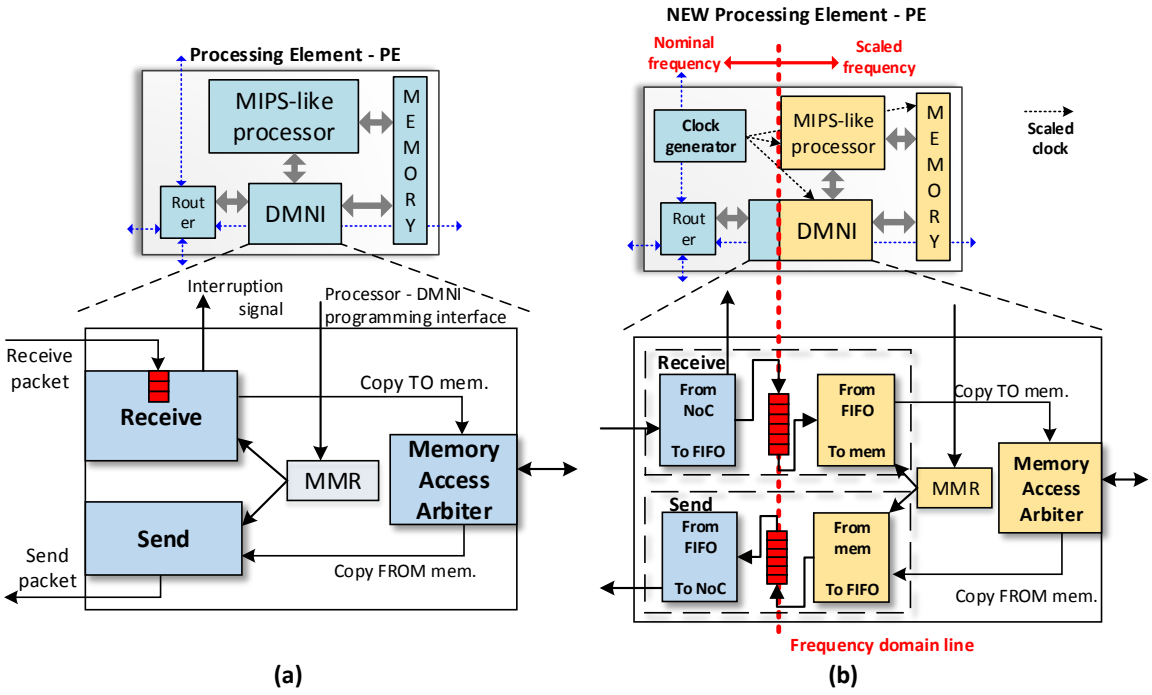


Figure 4.2 – (a) Original PE and (b) new PE with DVFS support. The new PE has Clock Generator hardware as well as changes on the DMNI [MSM16].

The DMNI synchronizes the hardware modules working at different frequencies. The original DMNI (Figure 4.2a) has a Send module responsible for reading the data from memory and converting it to a message for sending to the network. The Receive module reads the message from the NoC and copies them to the memory. The new version of DMNI (Figure 4.2b) has two bisynchronous FIFOs included in the Send/Receive modules to synchronize the DMNI. Both Send and Receive modules are divided into two modules,

one running at the *nominal frequency* and the other one running at the *scaled frequency*. The Receive module reads the message from the NoC at the *nominal frequency* and, next, writes the flits into the bisynchronous FIFO of the receiver. If this FIFO is not empty and the memory is ready for writing, the receiver copies the message from the FIFO to the memory by using the *scaled frequency*. The sending process is similar to receiving, but the data flow is in the opposite direction (read from memory, send to the network).

The main hardware overheads for frequency scaling support are the Clock Generator and the bisynchronous FIFO at the DMNI. Due to the insertion of FIFOs in the new DMNI, the average execution time of the applications is penalized in 6.55%.

4.2.2 Voltage Scaling Model

A voltage regulator is an analog circuit that allows voltage scaling in a system. The method for modeling the voltage scaling on the cycle-accurate reference platform considers a set of low-level characteristics. First, standard cells characterized for 1.1V, 1.0V and 0.9V define the supply voltages supported by the system since the foundry provides liberty files only for these supply voltages (65 nm technology). Next, the processor netlist (1.1V-250MHz) from characterization flow (Section 3.2) is evaluated for 1.0V and 0.9V supply voltages. Following, it is verified the delay for reading the memory at 1.0V and 0.9V. The minimum period to obtain a zero or positive time slack concerning processor and memory is 4.479ns and 5.229ns, for 1.0V and 0.9V, respectively. Finally, the router netlist is also evaluated for 1.0V and 0.9V supply voltages, but the goal is to certify the router can run at 250MHz in any voltage since the frequency scaling does not affect the router.

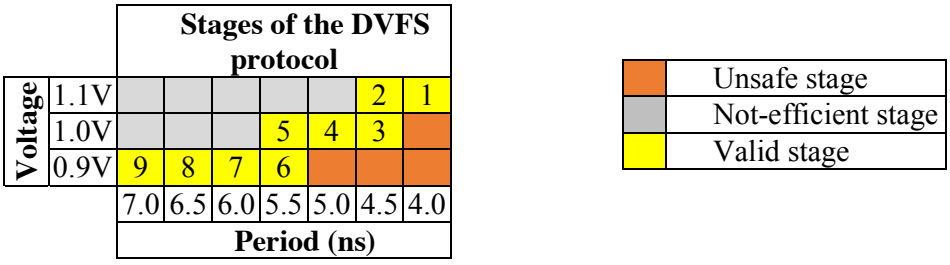


Figure 4.3 – DVFS protocol – Valid voltage and frequency pairs [MSM16].

Figure 4.3 defines the DVFS protocol by linking the minimum period for scaling the voltage safely with the frequency range generated by the Clock Generator. The numbers in the yellow boxes define valid *vf-pairs*. The ascending order establishes the protocol to scale the *vf-pair* down, while the descending order is the protocol to scale the *vf-pair* up. The system always starts at the nominal *vf-pair* (1.1V-4ns).

In general, coarse-grain voltage regulators present latency in the order of milliseconds while fine-grain latency is lower than hundreds of nanoseconds [KBW12, LWP14]. On the other hand, the energy overhead from on-chip voltage regulators to support fine-grain voltage scaling is non-negligible [KGWB08]. Due to the low latency of fine-grain voltage regulators and the feature of frequency scaling at the PE-level, the model a fine-grain (PE-level) voltage scaling assumes that the latency of a voltage scaling (up or down) is 100 ns (25 clock cycles at the nominal frequency), and the energy overhead from on-chip voltage regulators increases the PE energy in 10% [CCK07].

4.2.3 Power-Gating and Clock-Gating

Many-cores widely employ Power Gating (PG) and Clock Gating (CG) are power actuators widely employed in many-core systems. PG provides larger power impact than DVFS and CG because it is the only mechanism to eliminate leakage [AMP+15, JMSN05]. PG is usually deployed when no tasks are running in the PE. However, the PE needs to stay off long enough to compensate the time, power and energy overheads to wake it up back. As this proposal assumes applications can entry anytime, integrating PG model would require predictions concerning how many time the PE will be on or off. For this Thesis, PG is not integrated, but for comparisons with related works at Chapter 6, all RMs are normalized under the same rule: if the PE is running no tasks, it is considered power-gated. In fact, all RMs based on bottom-up characterization flows make this assumption [HRW+14, RHK+15, RTGM13, KRS+14].

CG is more suitable to deal with idle times when the PE is executing tasks due to the short duration of idle times. The power impact is smaller than PG, but it is also significant. The proposed CG model affects only the processor and the memory. The processor supports clock hold, i.e., when the processor is idle, the clock signal is disabled, saving dynamic power. Since the characterization computes dynamic power from the instructions counters, no changes are required to model CG for the processor. Similarly, concerning memory, dynamic power comes from load and store operations only. Therefore, when memory operations are not happening, the memory is considered in idle and no dynamic power is accumulated during idle periods. The router is continuously spending dynamic power, in the active or in the idle state, but the idle state of the router considers the dynamic power from buffers (Equation 3.5). The timing overhead from CG is considered negligible, and it is not inserted into the model.

4.3 Software Actuation

The software actuators (Definition 9) are abstractions to act in the system by using message passing communication services (inherent to the adopted many-core) to invoke services . Accordingly, this Section describes the communication protocol to synchronize the software actuators. The communication protocols satisfy the hierarchy sorting for the actuators set, i.e., the application allocation (Subsection 4.3.1) is a system level actuator, and task allocation (Subsection 4.3.2) and migration (Subsection 4.3.3) are cluster level actuators.

4.3.1 Application Allocation

The Application Allocation assigns an application task graph to a cluster. The services to synchronize the Application Allocation are as follows:

- `NEW_APP`: message sent from the GM to a CM for notifying that a new application was assigned to the cluster. This message carries the application task graph and design-time data (such as estimated power and application type - BE or RT). The CM maps all applications' tasks after receiving this message.

- `APP_ALLOCATION_REQUEST`: message sent from a CM to the GM for notifying that the cluster is ready to receive the tasks from the incoming application and requires the task allocation protocol. This message carries the position of each task of the incoming application.

Figure 4.4 presents a sequence diagram for allocating one application. An application from the Application Repository can enter at any time into the system by setting a hardware interruption to signalize to the GM the allocation request (step 1). Next, the GM runs an algorithm to *decide* if the system can admit the application and, in case of success, selects the cluster to allocate the application. At step 2 the GM sends a `NEW_APP` message to the chosen CM and blocks the interruption for new applications. After receiving the `NEW_APP` message, the CM triggers the task mapping algorithm for the incoming application. Once the CM *decides* the mapping of all tasks of the application, the CM sends `APP_ALLOCATION_REQUEST` message to the GM (step 3). After processing the `APP_ALLOCATION_REQUEST` service, the GM unblocks the interruption for new applications and starts the task allocation protocol (steps from 4 to 8 presented in the next Section). The Application Admission and Task Mapping are *Decision* algorithms presented in Chapters 5 and 6.

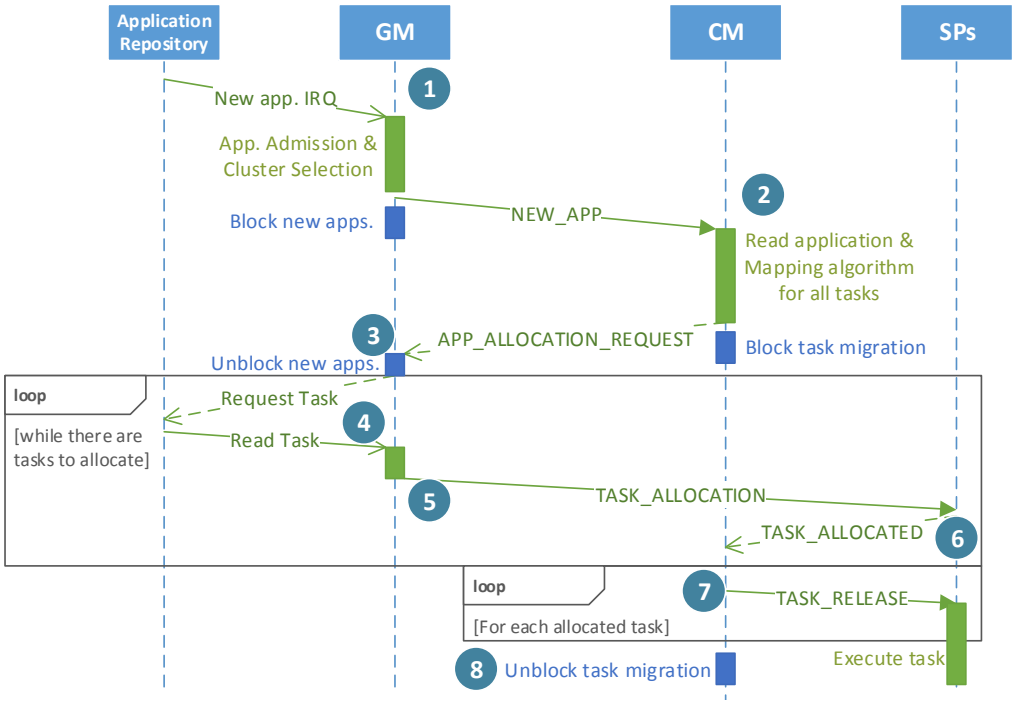


Figure 4.4 – Diagram of the Application Allocation and Task Allocation protocols.

4.3.2 Task Allocation

The Task Allocation protocol follows the Application Allocation protocol. This protocol coordinates the assignment of all tasks to their SPs, once the mapping decision for all tasks was already defined. Because the Task Allocation transmits the object code of all application tasks to the SPs of a cluster (i.e., a considerable communication load is about to start), any task migration is temporarily blocked. The services to synchronize the Task Allocation are as follows:

- `TASK_ALLOCATION`: it loads the object code of a task into the memory of the SP (direction of the message: GM to SP);
- `TASK_ALLOCATED`: it notifies the CM that a task was successfully loaded into an SP (direction of the message: SP to CM)
- `TASK_RELEASE`: it releases the allocated task (direction of the message: CM to SP).

Once the Application Allocation finishes, a loop to allocate the tasks begins (Figure 4.4). The GM reads a task from the Application Repository (step 4) and sends a `TASK_ALLOCATION` message to an SP (step 5). The GM is aware of the task mapping because this information was embodied in the `APP_ALLOCATION_REQUEST` message. Next, when the SP receives the task to execute, it sends a `TASK_ALLOCATED` message to the CM to notify that the task was successfully allocated (step 6). When the CM receives all `TASK_ALLOCATED`

messages, it sends TASK_RELEASE messages to release the allocated tasks (step 7), i.e., the new application starts. Finally, after the last received TASK_RELEASE message, the CM unblocks the migrations to finish the Task Allocation protocol (step 8).

Although Task Allocation is classified as a cluster level actuation, the GM plays a relevant role in the protocol due to the exclusive access to the Application Repository. Furthermore, the GM sends TASK_ALLOCATION messages directly to the SPs of any cluster. This design choice comes from two reasons: (*i*) to avoid the transmission of the object code first to the CM and then to the SPs; (*ii*) due to the adoption of XY routing, if all object codes were transmitted to the CM, a network congestion (hotspots) would occur.

4.3.3 Task Migration

Task Migration is an essential feature to support adaptability because it allows remapping of the application at runtime. The task migration goal is to move a task from a source SP (SP_{src}) to a target SP (SP_{tgt}). The *decision* state is in charge of defining the criteria to choose the task to migrate according to the RM goals. The task migration employs a low latency protocol for many-cores with distributed memory [RM17]. The task migration protocol requires neither checkpoints nor task code replication as well as allows task migrations in parallel.

The typical layout of a memory page loaded with a task contains read-only and read-write sections. The read-only section is the task object code (*text* at Figure 4.5). The read-write sections are global variables (*data* and *bss*) and the *stack* area (Figure 4.5). The Task Migration protocol migrates all sections of the memory page. The services to synchronize the Task Migration are as follows:

- TASK_MIGRATION: it notifies the SP_{src} which task should migrate to SP_{tgt} to initialize task migration. (direction: CM to SP);
- MIGRATE_TEXT: it migrates the *text* section (direction: SP to SP);
- MIGRATE_STACK: it migrates the *stack* section (direction: SP to SP);
- MIGRATE_BSS_DATA: it migrates the *bss* and *data* sections (direction: SP to SP);
- TASK_MIGRATED: it notifies the CM that a task was successfully migrated from SP_{src} to SP_{tgt} (direction: SP to CM).

A decision algorithm starts the Task Migration protocol at a CM. The CM sends the TASK_MIGRATION message to SP_{src} and blocks both application and task allocations while there are ongoing migrations (step 1). Once the SP_{src} is aware of the migration request,

SP_{src} can send a `MIGRATE_TEXT` message to migrate the read-only part of the task code, without blocking its execution (step 2). Next, SP_{src} chooses an appropriate moment to stop the task and save the context (step 3) to enable the migration of the read-write memory segments (step 4). After sending the `MIGRATE_STACK` and `MIGRATE_BSS_DATA` messages, SP_{src} updates the new task location for all communicating tasks if they exist (step 5). In parallel, SP_{tgt} restores the context and can proceed the task execution after reporting to the CM the successful finish of task migration (step 6). After receiving the `TASK_MIGRATED` message, the CM unblocks all task operations (step 7).

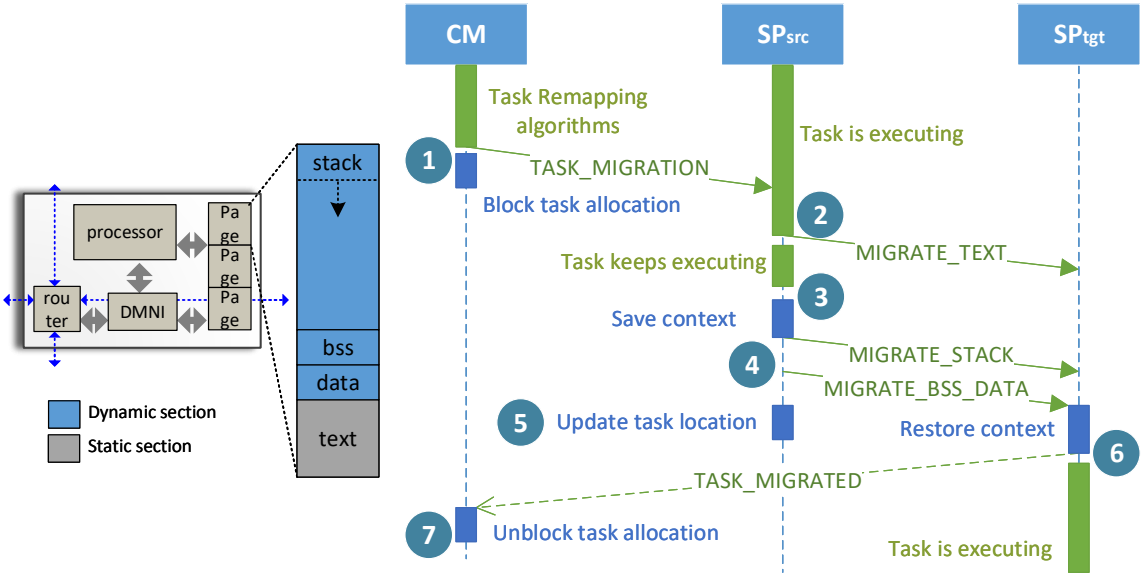


Figure 4.5 – Diagram of Task Migration protocol.

The reason to block task allocations while migrating a task is the same: avoid unnecessary network congestion and ensure the synchronization. Although a concurrency between task migration and task allocation requests is unlikely to happen, the CM guarantees either only task migrations or only task allocations running on the cluster.

4.4 Evaluation of Actuation Methods Concerning power, and resources

This Section evaluates the impact on the power, energy, and resources when setting the actuators individually by using synthetic test cases. These evaluations aim to provide insight into the development of the Decision heuristics in the next Chapters. As mentioned before (Section 4.1), observing the actuation properties and considering these data into Decision phase can lead to better management [LLW+13, MPV+13, HV14] because the system is aware of the costs of its self-configuration.

4.4.1 Analyzing the Power Consumption at the PE Level

First experiment illustrates the maximum and minimum power consumption at the PE-level using the DVFS actuator. Once the DVFS protocol is defined (Figure 4.3), the PE is characterized by each supply voltage, considering the smallest periods (*vf-pairs* 1, 3, 6). The router always works at the nominal frequency (i.e., 4ns). The processor and the memory consider the power per instruction and per operation (read/write), respectively. Therefore, it is not necessary to characterize the modules for each frequency. As a result, each PE component has three look-up tables (1.1V, 1.0V, 0.9V), obtained from the characterization flow.

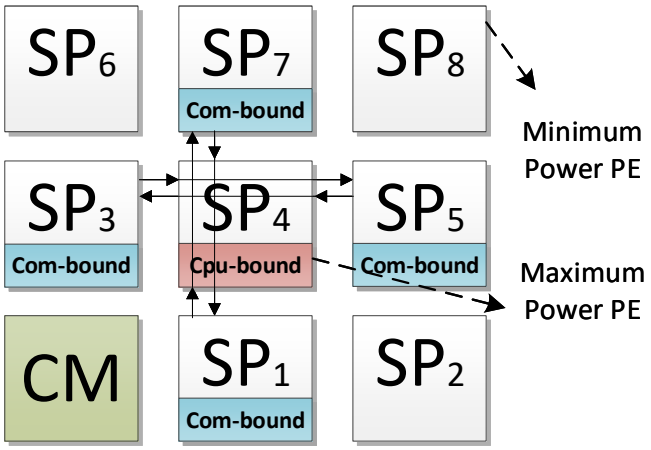


Figure 4.6 – A 3x3 many-core system executing synthetic tasks mapped to maximize the average power dissipation of the central SP, and minimize the power dissipation of the SP at the corners [MRSM17b].

Figure 4.6 presents the experimental setup to determine upper and lower bounds values related to the power consumption in a SP. The central SP (SP_4) executes a CPU-bound task, with an uniform distribution of instructions classes. The SP_4 neighbors execute communication-bound tasks, generating traffic traversing SP_4 router. With this scenario, the consumption of SP_4 defines the *maximum power* (p_{max}) value that an SP can consume in an epoch (Definition 7). On the contrary, SP at the corners (2, 6, 8) spend mostly leakage power because these PE have no tasks to execute (processors in hold state), and there is no traffic traversing the routers of these SP (routers in idle mode). Note that these routers have only three ports so that the router consumption reduces. Therefore, the power consumption at these SP defines the *minimum power* (p_{min}) value that an SP can consume in one epoch.

Figure 4.7 details the power consumption for each SP component considering the scenario presented in Figure 4.6. The histograms assume three supply voltages for an epoch equal to 1 ms. The histograms show the contribution of the three modules in the power consumption, considering *vf-pairs* as 1, 3 and 6. The comparison of p_{max} and p_{min} highlights the effect of the leakage power.

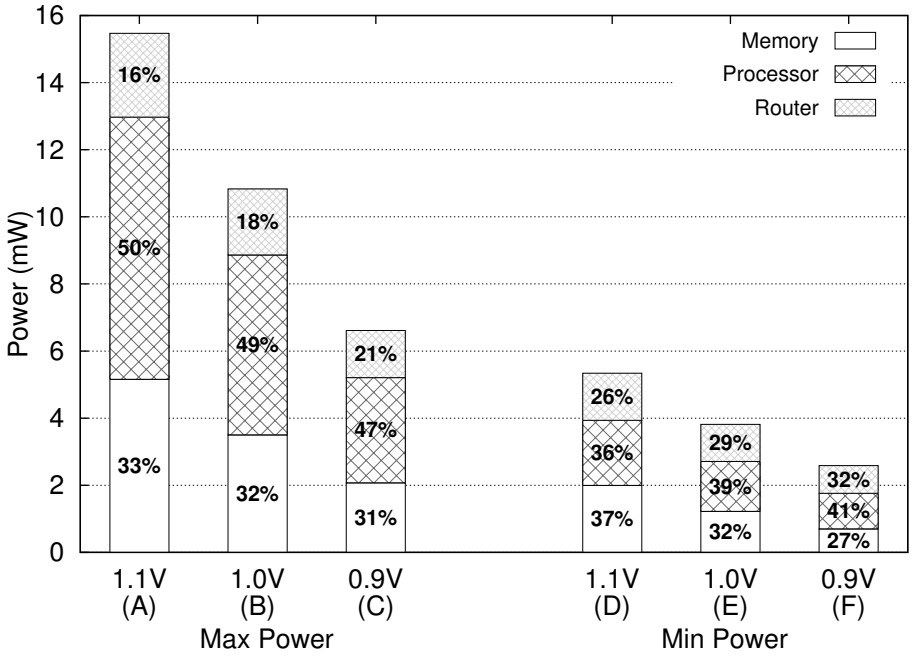


Figure 4.7 – Power profiling of the SP for all supply voltages. The total power (y-axis) corresponds to the power consumption in an epoch of 1 ms [MRSM17b].

The p_{max} histograms show the contribution of the three modules in the power consumption: 50% processor, 30% memory, and 20% router. As the voltage reduces, the portion due to router power increases because only the processor and the memory works on the scaled frequency. Therefore, in the epoch, the number of executed instructions and memory accesses reduces when the frequency is scaled down. The p_{min} histograms present a distinct behavior, with an increased consumption by the routers due to the lack of clock hold. p_{min} is approximately one-third of p_{max} when comparing the bars of the same voltage. The comparison of p_{min} at 0.9V (processor executing CPU-bound tasks) and p_{max} at 1.1V (processor in hold state) evidences the effect of the leakage power. The p_{max} at 0.9V is only 18% higher than the p_{min} at 1.1V. These results show a tendency that reduces p_{min} with new technologies, such as FD-SOI.

4.4.2 Power Consumption During Task Phases

Figure 4.8 depicts the scheduling of two tasks running in two distinct SPs. The blue bars mean when the processor is busy, and the green bars are idle periods. Figure 4.9 shows the power profile of tasks A and B including the CG model. At the beginning of the execution, from 0 to 1,000 Kticks, Task A is in a busy phase spending around 14mW while Task B stays mostly in idle state consuming around 5.5mW. After 1,000 Kticks, both tasks alternated busy and idle states. Note that the idle phase of both tasks generates lower power due to the CG, while the busy phase creates peaks of power. This relation between

power and utilization can also be observed at Figure 1.1 but at the system level instead of PE level. Finally, the third curve (blue line points) at Figure 4.9 illustrates the power of an SP when no tasks are running, i.e., the minimum average power of an SP. As PG is not modeled for this example, the SPs running no tasks shows the potential power savings of PG as well as establishes the limit of CG savings.

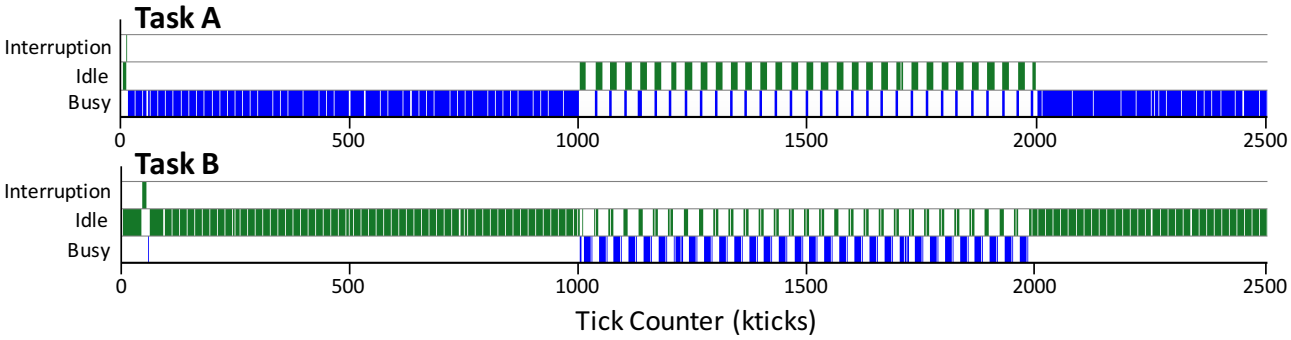


Figure 4.8 – Processor scheduling zooms in task phases of two tasks running in two different PEs to highlight idle times of the tasks.

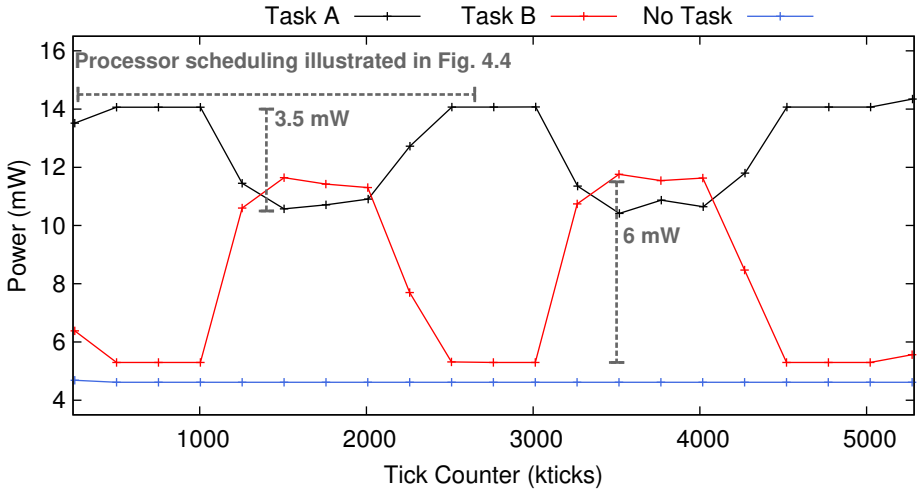


Figure 4.9 – The power curves show the impact of CG relies on task phases. No task curve shows an SP spending only leakage power.

4.4.3 Resource and Power Impact of the Software Actuation

As stated at the beginning of this Chapter, software actuators have a larger impact than the hardware ones. Since the power constrains the resources utilization in RM, Figure 4.10 presents the system occupation for this experiment, using a 4x4 cluster, where each PE may execute up two tasks. Figure 4.11 illustrates the variation of power and resources for each software actuator. The bottom graph presents the individual power of each PE, and the top graph shows the cluster power. The simulation aims to highlight task events disturbing the power and resources. For the sake of readability, the PE running no tasks are considered power-gated in this example.

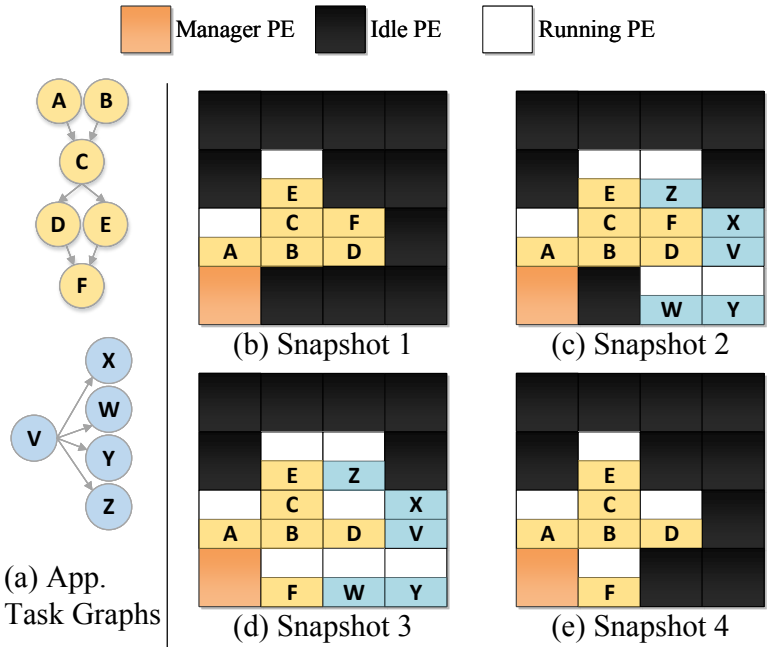


Figure 4.10 – Snapshots taken to show the application mappings at important moments explained by Figure 4.11.

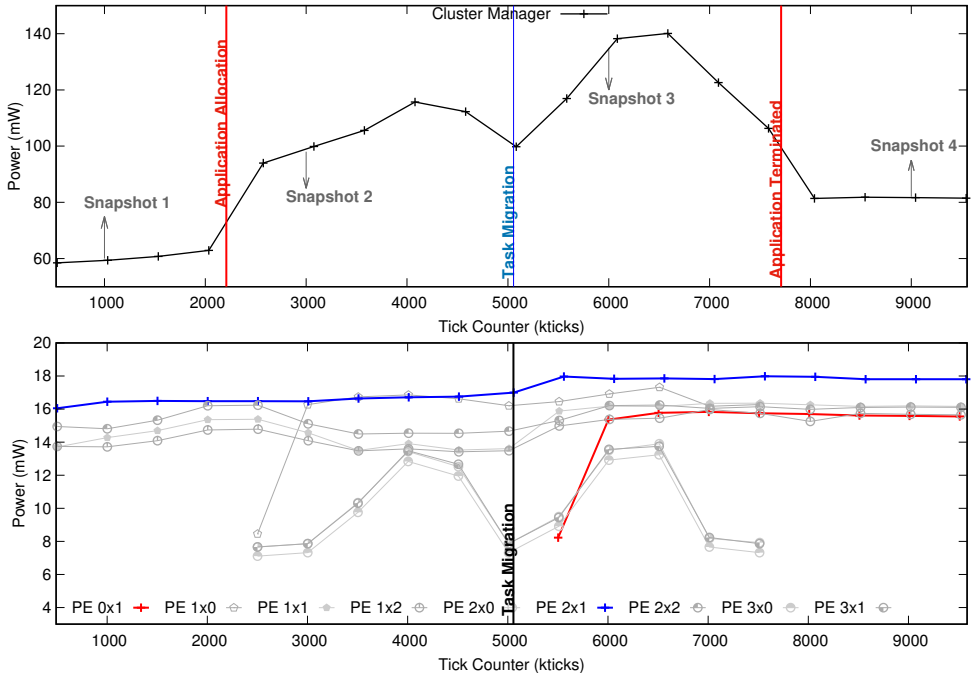


Figure 4.11 – Example of power and resource impact of software actuators in a 4x4 cluster. The top graph shows the power at the cluster level, and the bottom one illustrates the power in each PE.

The simulation starts with six tasks running in four PE (snapshot 1). Due to an Application Allocation, five new tasks occupy additional four PE and make the power increases around 80% (snapshot 2). In this sense, the bottom graph in Figure 4.11 illustrates the power variation felt by each PE due to the AA actuator. A task migration is the second software actuator triggered by the RM. The bottom graph in Figure 4.11 also show the power impact

caused by a task that migrates from a PE running two tasks (PE 2x1 - blue curve) to an idle PE (PE 0x1 - red curve) – snapshot 3. Although the number of tasks and resources is still the same after the task migration, the power in PE 2x1 increases because its utilization increases. Finally, the end of an application execution decreases the PE utilization and consequently the power as well (snapshot 4).

In general, Application Allocation and Task Allocation increase the power. However, the amount of this growth relies on the number of tasks, the number of PE used to allocate the tasks and the task characteristics e.g., if the task is communication-bound or CPU-bound. Although both applications use the same number of PE (four), Figure 4.11 shows that the first allocated application requires more power than the second one because the former has more tasks (six versus five) and the average power of its tasks is higher, according to the power curves shown in the bottom graph.

Concerning task migration, the power increases when occurring task migration from a PE running multiple tasks to an idle PE, similar to the presented in Figure 4.11. As the opposite, the power decreases when a task migration releases a PE and migrates to another busy PE. Similarly to task allocation, the amount of the power variation due to a task migration is variable and relies on the task characteristics. If the number of idle and busy PE does not change, the impact on the power consumption of a task migration is small. Thus, task migration can be best used as a power knob in the following situations: (*i*) when tasks are divided to run in distinct PE to make the application achieve better performance; or (*ii*) tasks are joined to share PE while releasing other PE to reduce power.

Besides the power variation due to different software actuators, Figure 4.11 also illustrates how the task phases affect the power consumption. In the bottom graph, the power of some tasks exhibits a constant behavior while other tasks have a period behavior with peaks and valleys of power. As a consequence, the overall power (top graph) is not constant even though no actuation occurs.

4.5 Final Remarks

This Chapter described the actuators set of the proposed RM. A review of related works shows that this proposal is the most comprehensive, i.e., it covers the largest number of actuators simultaneously. As bigger the amount of actuators is, more complex is the management. The distribution of the actuators management in a 3-level hierarchy scheme ensures scalability to the proposal. The criteria for the distribution is to sort the actuators concerning resource and power impact, and latency, i.e., the larger the actuator impact and latency, the higher is the hierarchy level.

Regarding hardware actuators, the reference many-core does not support the required actuators to deal with power issues. Therefore, the design and the modeling for each

power actuator is detailed to support their inherent overheads. The hardware actuators presented are DVFS, PG, and CG. Meanwhile, communication protocols illustrate how the hierarchical RM can coordinate the software actuators in such a way to avoid resources conflicts and network congestion. Furthermore, the individual switching of each actuator allows the observing of power and resources impact of each actuator.

5. DECIDING - RUNTIME ENERGY MANAGEMENT APPROACH

The workload of many-core systems includes soft real-time (RT) applications. This Chapter assumes the application set executing soft real-time (RT) and best-effort (BE) applications. RT applications, like multimedia, admit small variations in the latency [FSWV07] as well as can deal with some timing violations if these are not frequent [MEP08]. The management of hard real-time applications is out of this Thesis scope. Techniques used to decrease power consumption, such as dynamic voltage and frequency scaling (DVFS), may induce constraint misses if the management is not aware of the RT constraints. Therefore, the execution of RT applications while respecting the constraints of power and timing constitutes a challenging trade-off for Resource Management (RM) systems [SKF+14].

Furthermore, energy efficiency is an essential issue for any device, mainly the ones powered by a battery. The RM can adopt distinct strategies for BE and RT applications to achieve energy efficiency. Besides these strategies regarding the application type, the power and timing constraints constitute a multi-objective problem (Definition 3). Therefore, the execution of BE and RT applications to achieve energy efficiency under power constraints requires a multi-objective RM.

The evaluation of RM proposals reveals the following limitations:

1. Design-time based heuristics. Several works propose heuristics for power management (PM) according to a design-time analysis of the application set [LJJ13, SDK13, DKV14, DAHM16, JLK+14]. However, many-core systems are designed to support dynamic workloads, i.e., new applications may start their execution at any moment, making unfeasible to evaluate all execution scenarios at design-time [SSKH13];

2. Dynamic behavior of a many-core system even when assuming runtime based heuristics. The tasks' execution time is not constant for all iterations due to the inherent dynamic behavior of many-core systems (e.g., traffic congestion, interruptions, operating system scheduling) so that adaptability (Definition 4 on page 27) is required.

To tackle these limitations, this Chapter proposes an adaptive Runtime Energy Management (REM) designed to control both RT and BE applications while exploring the slack time of RT applications and the most efficient energy threshold to save energy for BE applications. This Chapter corresponds to an extension of the publication at the JOLPE journal [MRSM17b]. REM accumulates the benefits of scalability, comprehensiveness, and multi-layer observing scheme for energy, power, and RT constraints from the Observe and Act states previously presented (Chapters 3 and 4, respectively).

The Chapter contributions are as follows:

- apply PM in the scheduling of RT applications without design-time analysis of the applications set;

- employ distinct strategies of management according to the application type (BE or RT);
- support dynamic workload;
- achieve average energy savings in RM running RT application of 18%, with negligible timing violations.

Section 5.1 presents related works in RM for RT applications. Section 5.2 introduces RT and BE application models. Section 5.3 corresponds to the main contribution of this Chapter, the REM heuristic, which relies on the observing data to take decisions. Section 5.4 presents the REM results and a comparison with related works. Section 5.5 concludes the Chapter and points out directions for future works.

5.1 Related Works

A concern in RM is to meet RT constraints for the applications. The trade-off of meeting both RT constraints and the system power-related constraints is the related works challenge. Table 5.1 summarizes the related work. The 2^{nd} column presents the architecture and the core count given by the number of PEs or the NoC size. The 3^{rd} column presents the RM goals. The 4^{th} column distinguishes if the RM is a multi-objective one. The 5^{th} column lists the actuation techniques used to control the system according to the authors' definitions. The 6^{th} column presents the power modeling approach adopted by the Authors: top-down or bottom-up. The last column illustrates if the RM requires design-time evaluations of the application set.

According to Table 5.1, DVFS is the main technique used to control the system to achieve the design goals. However, it is not clear if Authors [MKP15, LJJ13, SDK13, JLK+14, HRW+14] consider the overhead due to the DVFS support. To fulfill this gap, the DVFS model employed in REM includes latency, delay, and energy overheads due to the DVFS support (Section 4.2).

Most works [MKP15, LJJ13, SDK13, DKV14, DAHM16, YSH14, JLK+14] use a small number of cores, compromising scalability. In particular, scalability for heterogeneous many-cores is an open research topic [SCW05]. Haghbayan et al. [HRW+14] work is closer to this Chapter proposal regarding system size but it applies a centralized congestion-aware task mapping [FRD+12] instead of hierarchical management. This Chapter proposal adopts a hierarchical management architecture to act in systems from few up to dozens of PEs, making the proposal scalable.

Although bottom-up modeling [MKP15, LJJ13, SDK13, JLK+14, SDI16] allows fast design space exploration, the accuracy is a function of the models available in the toolset, which may compromise the quality of the results [XJB+15]. Some Authors validate the RM in

embedded platforms [DAHM16, DKV14] and FPGAs [YSH14], but the system size is limited. Among the top-down approaches, Wei et al. [WLW+17] employ real CPU+FPGA hardware while Yu et al. [YSH14] simulate the RM targeting an FPGA device. The cycle-accurate model from Jung et al. [JLK+14] makes assumptions related to the hardware features, which may lead to a mismatch between the model and the actual hardware. REincludeM adopts a modeling step by characterization (Section 3.3) before the RM design to the low-level results (e.g., energy per instruction, energy per flit) in the cycle-accurate model.

Table 5.1 – State-of-art in RM for Many-core Systems running RT applications.

Proposal	Arch., # PEs	RM goals	Multi-objective?	Actuation set	Modelling	Design-time Info?
Maiti et al. [MKP15]	Homo., 6x4	Min. energy or max. performance	✓	TA, DVFS, CG	Bottom-up (McPAT)	✓
Li et al. [LJJ13]	Homo., 9	Energy Efficiency and Temperature	✓	TA, DVFS	Bottom-up (HotSpot)	✓
Singh et al. [SDK13]	Homo., 4	Energy efficiency	✗	TA, DVFS	Bottom-up	✓
Das et al. [DKV14]	Homo., 3x3	Energy efficiency and Reliability	✓	DVFS	Bottom-up (validation in ARM-based embedded system)	✓
Das et al. [DAHM16]	Homo. Up to 3x3	Energy efficiency and Reliability	✓	DVFS	Bottom-up (validation in ARM-based embedded system)	✓
Yu et al. [YSH14]	Hetero., 9	Latency	✗	DVFS, TM	Top-down (FPGA simulation)	✗
Jung et al. [JLK+14]	Homo., 3x3	Energy efficiency	✓	TA, TM, DVFS	Top-down (ARM simulator)	✓
Haghbayan et al. [HRW+14]	Homo., 12x12, 11x11, 8x8	Performance under power cap	✓	TA, DVFS, PG	Bottom-up (in-house tool, McPAT, Lumos)	✗
Singh et al. [SDI15]	Homo., up to 6x8	Maximize performance and minimize energy	✓	TA, DVFS, PG	Bottom-up (in-house functional simulator)	✗
Wei et al. [WLW+17]	Hetero., 4xCPU + FPGA	Throughput optimization under power cap	✓	TA, TM, DVFS	Top-down (Intel i5 CPU + Xilinx VC707 FPGA)	✓
This	Homo., 3x3 up to 12x12	Scalability and energy efficiency	✓	AA, TA, TM, DVFS, CG	Top-down (Section 3.2)	✗

AA: Application Allocation; TA: Task Allocation; TM: Task Migration; PG: Power-Gating; CG: Clock-Gating

Hetero.: Heterogeneous; Homo.: Homogeneous

Some works [LJJ13, SDK13, DKV14, DAHM16, JLK+14, WLW+17] assume a previously known application set to execute design-time optimizations. Some proposals [LJJ13, SDK13] use design-time phases to define energy efficient task mapping and employ a runtime power adaptation (DVFS) to improve energy saving. The design-time exploration uses a profiling of known applications to reduce the complexity of runtime heuristics, but it is not flexible to be applied in general purpose application systems, where the application set is unknown at design-time. One work [SDI15] proposes both design-time and runtime management approaches for high-performance computing (HPC) many-core systems. The current proposal makes no assumptions related to the workload and executes both BE and RT tasks. REM takes decisions at runtime by observing the current system status. Similar to this proposal, Li et al. [LJJ13] and Sing et al. [SDK13] works explore the slack time of applications tasks to save energy. This method enables the deployment of heuristics that

combines power techniques (such as DVFS) with the slack time of RT tasks to improve the energy reduction at runtime.

Finally, from the works labeled as multi-objective RM (Table 5.1) most of them [MKP15, LJJ13, JLK+14, HRW+14] execute RT applications under power constraints to achieve a certain goal, such as this proposal. Similarly, Wei et al. [WLW+17] execute different frames in parallel of streaming applications using pipeline under latency and power constraints. Singh et al. [SDI15] work focuses on simultaneously maximizing performance and minimizing the energy of tasks while keeping latency, instead of saving more energy like most of the works. Both works of Das et al. [DKV14, DAHM16] optimize the RM for energy efficiency and reliability. Unfortunately, other works [MKP15, HRW+14] abstract the model of RT tasks, how the RT constraints are observed, and the impact of the power management techniques in the task's execution time. Haghbayan et al. [HRW+14] work and this proposal distinguish BE tasks and RT tasks to take decisions regarding the application type. Also, Haghbayan et al. [HRW+14] do not apply power techniques in the processor mapped with RT tasks.

5.2 Application Model

RT applications contain a set of tasks, where the correct execution includes not only the expected results but also when these results are produced. The RT application model is an extension of the generic model described in Chapter 1. Thus, a directed acyclic task graph $G = (T, E)$ models an m-task application $A = t_1, t_2, ..., t_m$, where each vertex $t_i \in T$ is a task and the directed edge (e_i, e_j), denoted as $e_{ij} \in E$, is the communication between tasks t_i and t_j. An application may be either BE or RT, denoted as A_{BE} or A_{RT}. The following Definitions are applied to A_{RT}s.

Definition 10. $A_{RT}(p)$ - application hyper-period. The hyper-period is the smallest time interval after which the periodic patterns of all the tasks are repeated.

Definition 11. $A_{RT}(x)$ - application execution time. It corresponds to the time to execute all tasks during one $A_{RT}(p)$.

Definition 12. $A_{RT}(s)$ - application slack time. It corresponds to the difference between $A_{RT}(p)$ and $A_{RT}(x)$ as follows:

$$A_{RT}(s) = A_{RT}(p) - A_{RT}(x) \tag{5.1}$$

Definition 13. t_{RT} - RT task. It is a tuple $t_{RT} = (x, u)$ where x is the task execution time including operating system events and communication time, and u is the task utilization. The task utilization, $t_{RT}(u)$ corresponds to:

$$t_{RT}(u) = \frac{t_{RT}(x) * 100}{A_{RT}(p)} \tag{5.2}$$

Definition 14. t_{BE} - BE task. It is a task without timing constraint.

Figure 5.1(a) shows an example of a task graph for an A_{RT}, and its scheduling without (Figure 5.1(b)) and with (Figure 5.1(c)) task level DVFS. The DVFS applied at tasks 'b2', 'c' and 'd1' increases their $t_{RT}(x)$ and the $A_{RT}(x)$. Note that, even though $A_{RT}(x)$ increases, $A_{RT}(p)$ is not violated because $A_{RT}(s)$ remains positive. Figure 5.1 also shows examples of CPU sharing between t_{BE} with t_{RT} in PE0 and in PE4.

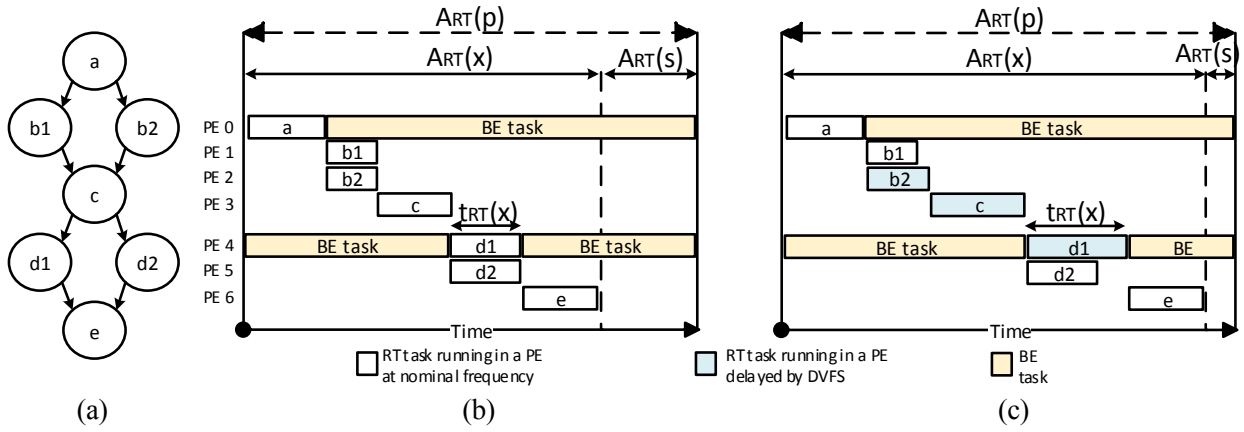


Figure 5.1 – (a)Task graph of an A_{RT} and an example of scheduling to present A_{RT} properties at both (b) nominal *vf-pairs* and (c) scaled *vf-pairs* [MRSM17b].

The $A_{RT}(s)$ is the *key* opportunity for energy savings because the system manager may set the *vf-pairs* of each t_{RT} without violating the $A_{RT}(p)$. Applying DVFS imposes delay to tasks. In Figure 5.1, for instance, some of the t_{RT}s have their execution time increased due to the adoption of the DVFS. Delayed tasks do not violate the $A_{RT}(p)$ *iff* $A_{RT}(s) \geq 0$, i.e., $A_{RT}(x) \leq A_{RT}(p)$.

To support REM, A_{RT} adopts syscalls *RealTime* and *PeriodMonitoring* (Section 3.3.3) to observe the A_{RT} properties and verify the timing constraints. Due to all t_{RT}s call *RealTime*, the CM uses *RealTime* data to compute $t_{RT}(x)$ and $t_{RT}(u)$. Because *PeriodMonitoring* syscall is available only in the firsts and lasts t_{RT}, *PeriodMonitoring* data allows the CM to compute the current $A_{RT}(x)$ and $A_{RT}(s)$. Note that, the *PeriodMonitoring* captures delays induced by the application tasks, traffic in the NoC, and OS events (as interruptions). Therefore, the computation of $A_{RT}(x)$ and $A_{RT}(s)$ enables the REM heuristic to act in the application since they are accurate measures. The SPs schedule RT tasks using the Least Slack Time (LST) scheduler [RM16], and a round-robin scheduler for BE tasks.

5.3 Runtime Energy Management - REM

Figure 5.2 presents a general view of the REM, according to the hierarchy adopted to manage the system (Figure 1.3). When an incoming application request execution in the system, the GM selects the cluster to allocate the application, executing the application admission *decision* procedure (Section 5.3.1). The GM sends to the CM of the selected cluster the application task graph and executes the task mapping and task remapping heuristics (Section 5.3.2). According to the mapping result, the selected SPs receive the object code of the tasks. At the PE level, SP estimates the current energy to enable its CM observing the energy. Particularly, in case of t_{RT}, the running t_{RT} executes *syscalls* for observing the RT constraints. A given SP may execute one t_{RT}, a set of t_{BE}s, or one t_{RT} and a set of t_{BE}s. According to the task set executing in the PEs, the CM decides the *vf-pairs* settings that SPs have to act. Sections 5.3.3, 5.3.4, and 5.3.5 detail the methods.

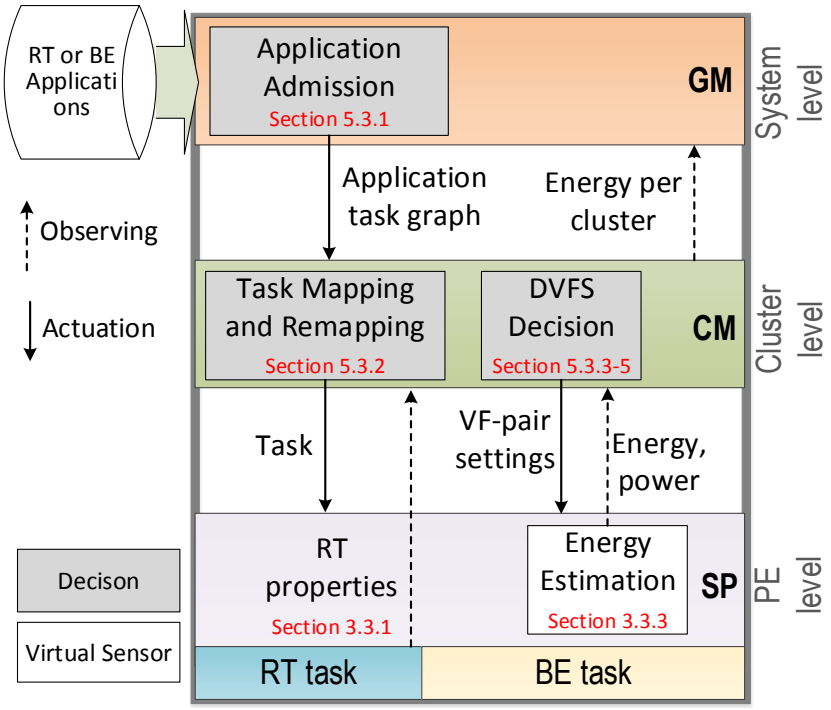


Figure 5.2 – General REM overview [MRSM17b].

5.3.1 Application Admission

The GM receives requests to execute new applications from the application repository by executing Algorithm 5.1. The application type, RT or BE, is the only design-time information available to the REM. The GM verifies if there are enough resources (see Definition 6) in the system to run all tasks of the application. In the absence of resources to

execute the application, the GM delays the admission of the incoming application up to the release of enough resources (lines 3-5).

Lines 6-11 creates the set cl_{set}, with the clusters that may execute the application. If the cl_{set} is not empty (lines 12-13), the selected cluster is the one with the lowest consumed energy based on the observing data. The goal is to distribute the load evenly in the system to avoid hotspots. Otherwise (lines 14-15), the selected cluster is the one with the maximum number of available resources. In such a case, a reclustering protocol [CMMM13] is applied, and the select cluster borrows resources from neighbor clusters.

Algorithm 5.1 Application Admission

```
1: Inputs: app, sys
2: Outputs: cl_output
3: if sys.resources < app.resources then
4:     return ∅
5: end if
6: cl_set ← ∅
7: for each cl_k ∈ sys.cl_set do
8:     if cl_k.resources > app.resources then
9:         cl_set ← cl_set ∪ cl_k
10:    end if
11: end for
12: if cl_set ≠ ∅ then
13:    cl_output ← minEnergy(cl_set)
14: else
15:    cl_output ← maxvAilableResources(cl_set)
16: end if
```

5.3.2 Task Mapping and Remapping

The manager PEs employ a set of cost functions to guide the task mapping and remapping:

- *availableSPs*: it returns the set of SPs with available resources to execute tasks;
- *SPs_run_RT_task*: it returns the set of SPs running t_{RT}s;
- *minEnergy*: it returns the SP with the minimum current energy nearest to the SPs executing tasks of the application.

The manager PE maps a task t_i according to the Algorithm 5.2. The first action is the creation of a set with the SPs able to execute t_i (line 2) - SP_{set}. For a t_{RT} (line 4), the

mapping algorithm excludes from SP_{set} the SPs running t_{RT}s (line 5), and then chooses the SP with the minimum current energy (line 6). For t_{BE}s only the minimum current energy is considered (line 8). If an SP may receive t_i, the algorithm (line 11) executes the mapping by calling the task allocation protocol, i.e., request to the GM to transfer the object code of t_i to the selected SP. The mapping of a t_{RT} may not occur, even if the cluster has resources to execute the task. A task mapping failure happens when SPs that are not executing t_{RT}s have all resources used by t_{BE}s. In this exceptional case, it is necessary to migrate a t_{BE}, releasing a memory page to receive a t_{RT}.

Algorithm 5.2 REM Task Mapping

1: **Inputs**: $cl.SP_{set}$, $app.t_i$
2: $SP_{set} \leftarrow$ **availableSPs**($cl.SP_{set}$)
3: $SP_{selected} \leftarrow \emptyset$
4: **if** $app.t_i.type$ = RT **then**
5: $SP_{set} \leftarrow SP_{set} -$ **SPs_run_RT_task**(SP_{set})
6: $SP_{selected} \leftarrow$ **minEnergy**(SP_{set})
7: **else**
8: $SP_{selected} \leftarrow$ **minEnergy**(SP_{set})
9: **end if**
10: **if** $SP_{selected} \neq \emptyset$ **then**
11: Map task $app.t_i$ in $SP_{selected}$
12: **else**
13: Call *Reclustering protocol*
14: **end if**

A manager PE migrates a t_{BE} according to Algorithm 5.3. Initially, it is created a set of SPs running t_{RT}s with available resources to receive a new t_{BE} (line 2) - RT_SP_{set}. Then, lines 4-6 creates the set $migSP_{set}$ containing SPs executing only t_{BE}s without resources to execute new tasks. Next, the algorithm selects an SP address (SP_{tgt}) to receive the t_{BE} and the SP address with the t_{BE} to be migrated (SP_{src}). Finally, REM Task Remapping function selects a t_{BE} mapped in the SP_{src}, remapping it to the SP_{tgt}.

Algorithm 5.3 REM Task Remapping

1: **Inputs**: $cl.SP_{set}$
2: $RT\_SP_{set} \leftarrow$ **availableSPs**($cl.SP_{set}$) $\cap$ **SPs_run_RT_task**($cl.SP_{set}$)
3: $usedSP_{set} \leftarrow cl.SP_{set} -$ **availableSPs**($cl.SP_{set}$)
4: $migSP_{set} \leftarrow usedSP_{set} -$ **SPs_run_RT_task**($cl.SP_{set}$)
5: $SP_{src} \leftarrow$ **minEnergy**($migSP_{set}$)
6: $SP_{tgt} \leftarrow$ **minEnergy**(RT_SP_{set})
7: Migrate a task from SP_{src} to SP_{tgt}

After the remapping process, the SP_{src} can receive a t_{RT}. Therefore, when the REM Task Mapping (Algorithm 5.2) for a t_{RT} fails, the remapping is fired, and the t_{RT} is mapped in SP_{src} afterwards. Since the GM admits an application only if there is room in the system, the remapping algorithm always finds an SP available to remap the t_{BE}.

A restriction adopted in the mapping procedure is: only one t_{RT} may be assigned to an SP at the same time. This restriction avoids the overlapping of t_{RT}s, preventing deadline misses. Indrusiak et al. [Ind14] presents a schedulability test, executed at design-time to enable the mapping of more than one t_{RT} in the same processor.

5.3.3 DVFS for an SP running only an RT task

The algorithm to apply DVFS to SPs executing one t_{RT} considers the timing properties of the A_{RT}s. To manage RT applications, only *vf-pairs* 1, 3, and 6 are used, which are the most energy efficient *vf-pairs* (Figure 4.3). The REM adopts the following definitions to set the *vf-pairs* of each task of an RT application:

Definition 15. *HIGH_UTILIZATION* - $t_{RT}(u)$ above a predefined threshold related to the $A_{RT}(p)$.

Definition 16. *LOW_UTILIZATION* - $t_{RT}(u)$ below a predefined threshold related to the $A_{RT}(p)$.

The current proposal adopts fixed thresholds because the adaptation at runtime of these thresholds without evaluating the applications set at design-time would require a larger warming up period and learning-based algorithms [YSM+15]. In this Chapter proposal HIGH_UTILIZATION and LOW_UTILIZATION correspond to 70% and 30%, respectively.

Algorithm 5.4 runs at manager PEs, and it defines the *vf-pair* of an SP executing a t_{RT}. This algorithm is triggered when the manager processor receives a packet created by the execution of a *syscall RealTime* in an SP. According to Figure 4.3, *vf-pairs* 3 and 6 delay a task execution by 12.5% and 37.5%, respectively, due to the frequency reduction. With a high CPU utilization, the SP operate at the nominal voltage (lines 3-4). With a low CPU utilization, the SP operates at the *vf-pair* 6 and delays the task by 37.5% (lines 5-6). Between thresholds, the task execution is delayed by 12.5% (lines 7-8).

If the manager PE receives a predefined number of hyper-period slack time violations (from the *syscall PeriodMonitoring*), the REM reset the *vf-pair* of all application tasks to the nominal voltage (*vf-pair(1)*). Further, to save energy, the manager PEs set the lowest *vf-pair* to the SP when the SP has no tasks to execute.

Algorithm 5.4 REM DVFS for RT tasks

1: **Inputs**: A_{RT}, t_{RT}, SP
2: $t_{RT}(u) \leftarrow (t_{RT}(x) * 100)/A_{RT}(p)$
3: **if** $t_{RT}(u) > HIGH\_UTILIZATION$ **then**
4: **sendVF**(SP, vf-pair(1))
5: **else if** $t_{RT}(u) < LOW\_UTILIZATION$ **then**
6: **sendVF**(SP, vf-pair(6))
7: **else**
8: **sendVF**(SP, vf-pair(3))
9: **end if**

5.3.4 DVFS for an SP running only BE tasks

The utilization and the behavior of t_{BE}s are unpredictable due to the absence of timing constraints. Thus, the strategy to achieve energy savings considers the energy profiling data (Figure 4.7). The most energy efficient voltage is the 1.0V because the E_{max} in an epoch reduces approximately 30% with an execution time overhead of 12.5%, compared to 1.1V. The E_{max} for 0.9V reduces more than 50% the energy compared to 1.1V, but the execution time overhead of 37.5% makes the leakage mitigate part of energy savings. Hence, the REM defines three energy zones: (i) *hot zone*: the current energy is above E_{max} of 1.0V; (ii) *cold zone*: the energy is below E_{max} of 0.9V; (iii) *warm zone*: energy between hot and cold zones.

The thresholds can be redefined to meet other goals instead of energy efficiency. For example, a restricted power cap can enforce the designer to adjust the thresholds to lower values than the proposed ones, i.e., a power-cap at PE level. In this case, the per-PE power cap penalizes the energy efficiency but guarantees no thermal issues [PKM+14].

When an observing message arrives from an SP running only t_{BE}s, the manager PE triggers Algorithm 5.5. The *vf-pair* is decremented when the observing sampling is in the hot zone (lines 2-3), or it is incremented when the observing sampling is in the cold zone (lines 4-5). The goal of the energy zones is to keep the voltage of SPs executing t_{BE}s at 1.0V most of the time, regardless the SP utilization.

Algorithm 5.5 REM DVFS for BE tasks

1: **Inputs**: $energy$, SP
2: **if** $energy > HOT\_ZONE$ **then**
3: **sendVF**(SP, DVFS_DOWN)
4: **else if** $energy \leq COLD\_ZONE$ **then**
5: **sendVF**(SP, DVFS_UP)
6: **end if**

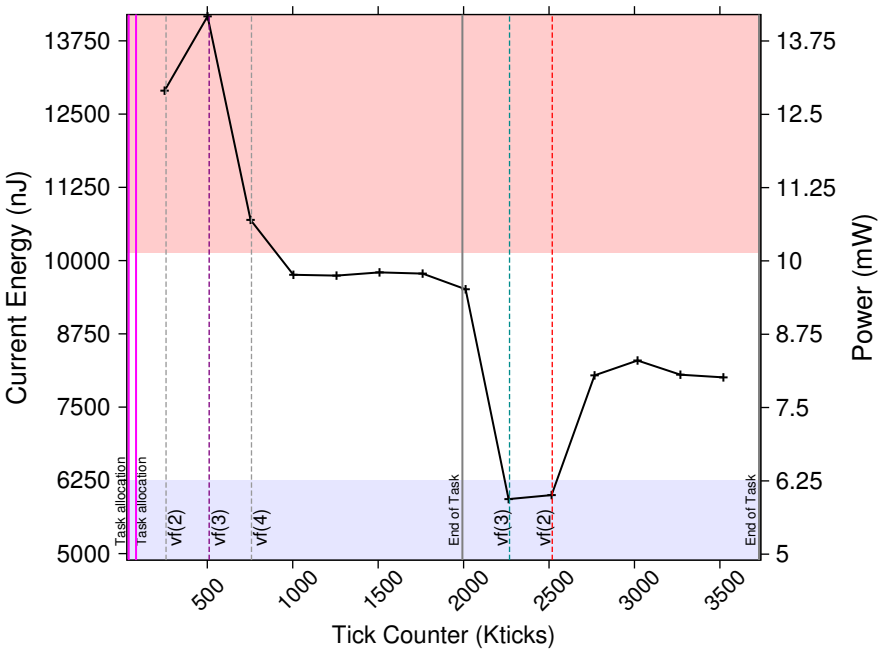


Figure 5.3 – REM uses DVFS for keeping the energy profile of the SP in the warm zone (white part). The warm zone of energy is from E_{max} of 0.9V to E_{max} of 1.0V [MRSM17b] (Section 4.4).

Figure 5.3 shows an example of the current energy in the left y-axis of an SP executing t_{BE}s (the right y-axis presents the average power in the epoch). When the simulation starts, the REM identifies a peak of energy (hot zone, the red part of the graph) in the first three observing samplings and sends three orders to scale the *vf-pair* down until the current energy decreases to warm zone (white part of the graph) in the fourth sampling. Next, the energy stays on the warm zone until the end of one t_{BE} (2,000 Kticks). The controller identifies that the remaining task are working in the cold zone (blue part of the graph) and scales the *vf-pair* up to the warm zone.

5.3.5 DVFS for an SP running an RT task and BE tasks

The REM allows t_{BE}s and one t_{RT} sharing the same SP. As the t_{RT} has higher priority in the scheduler, the rules to set the *vf-pairs* are the same adopted in the Algorithm 5.4. The t_{RT}s with lower utilization offer more CPU time to schedule t_{BE}s. Thus, when the REM maps or migrates t_{BE}s together with a t_{RT}, the SP chosen is the one with the least utilization in the cluster. In fact, the cost function *minEnergy* returns the SP with lower utilization for these cases.

5.4 Results

The experiments use the clock cycle-accurate RTL SystemC model of the reference many-core system. Applications and OS are described in C language, compiled from C code and executed over cycle-accurate models of the processing cores. The RT benchmarks are *DTW* (6 tasks), *Dijkstra* (7 tasks), and *MPEG* (5 tasks) while the BE applications are *synthetic* (6 tasks) and *prod-cons* (2 tasks).

The simulation of the baseline many-core architecture generates the results used for comparing the proposed methods. The baseline many-core has no DVFS while keeping the RT scheduler for processing t_{RT}s, the mapping, and the energy observing. The baseline platform does not have the 10% of energy overhead neither spends 6.55% more time due to the DVFS implementation (Section 4.2).

Section 5.4.1 evaluates the effect of the traffic congestion and the RT application mapping, with the goal of justifying the cluster-based architecture and the runtime admission and mapping heuristics. Next, Sections 5.4.2, 5.4.3, and 5.4.4, evaluate the execution time of the RT applications, the total system energy, and the number of hyper-period violations. Finally, Section 5.4.5 makes a comparison of the proposal with the *state-of-the-art*.

5.4.1 Evaluation of the traffic congestion and the RT application mapping

The goal of this first experiment is to present the effect of the traffic congestion and the number of t_{RT}s mapped in the same PE. Figure 5.4 presents $A_{RT}(x)$ (black curve) and $A_{RT}(p)$ (red line) for different mapping scenarios considering the application illustrated in Figure 5.1.

Figure 5.4(a) presents the best-case scenario, with one task per PE, without disturbing traffic. After a warm-up period, the REM actuates, and the $A_{RT}(x)$ stabilizes. Figure 5.4(b) considers the same mapping of Figure 5.4(a), but with congestion due to the traffic from other applications executing in the system. This experiment maps the RT application in the middle of a 6x6 many-core, surrounded by other tasks generating traffic crossing the RT application. As observed, it is not possible to guarantee the constraints in the presence of congestion even for a best-case mapping if the links used by the RT application suffer congestion. Design-time techniques, such as the ones presented in [LJJ13, SDK13, DKV14, DAHM16, JLK+14], are unable to deal with the effect of congestion in the NoC, thus they are not suitable for dynamic workload scenarios.

Next, Figure 5.4(c) and Figure 5.4(d) evaluate the impact of a multi-task mapping. Figure 5.4(c) shows $A_{RT}(x)$ when tasks 'b1' and 'b2' share the same PE. $A_{RT}(p)$ is not respected for this mapping because, according to the application task graph, these two tasks

should execute in parallel, i.e., at different PEs. Figure 5.4(d) maps non-parallel tasks at the same PE. As observed, $A_{RT}(s)$ reduces, and a deadline miss occurs (at 1100 Kticks). These happen because the resources sharing between tasks of the same application may generate unpredictable events, like interruptions due to messages addressed to a given task mapped in the PE.

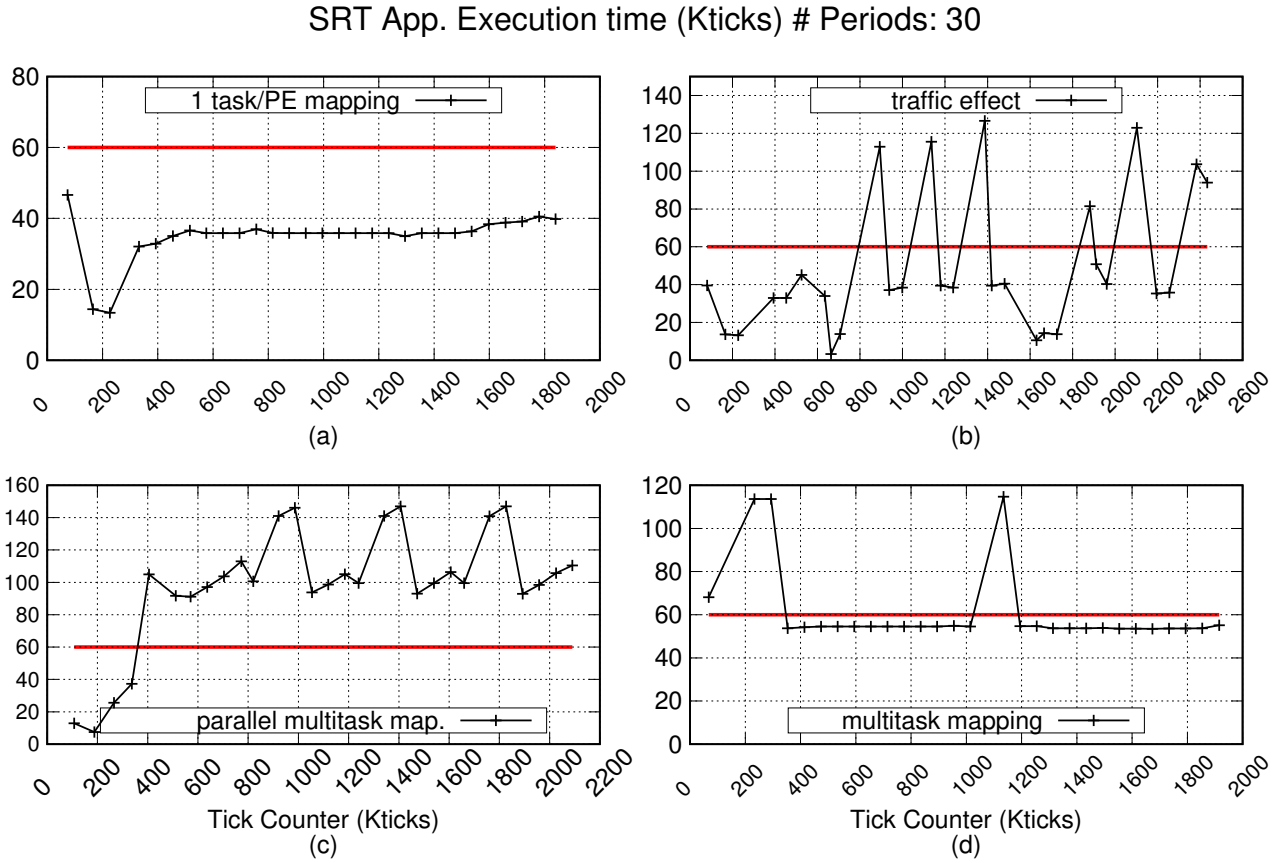


Figure 5.4 – Evaluation of $A_{RT}(x)$ for different mapping scenarios, considering traffic congestion and multi-task mapping.

REM tackles the issues in Figure 5.4(b-d) with the application admission and task mapping heuristics. The hierarchical organization helps to minimize the effect of traffic congestion (Figure 5.4(b)) by mapping applications inside the clusters and minimizing the application fragmentation (i.e., tasks belonging to same application spread at distant PEs). Besides, the Application Admission algorithm prioritizes the cluster with the largest number of available SPs to increase the mapping search space. The mapping algorithm avoids the problems depicted in Figure 5.4(c-d) because only one t_{RT} may be assigned to an SP at the same time, as discussed in the mapping heuristic Section (Section 5.3.2).

5.4.2 Evaluation of REM for one RT application

This experiment aims to show how REM can save energy while meeting the RT constraints. Figure 5.5(a) presents the hyper-period of an RT application (green horizontal line), the baseline execution time (black curve), and the execution time using REM (red curve). The distance from the execution time lines to the hyper-period corresponds to the slack-time.

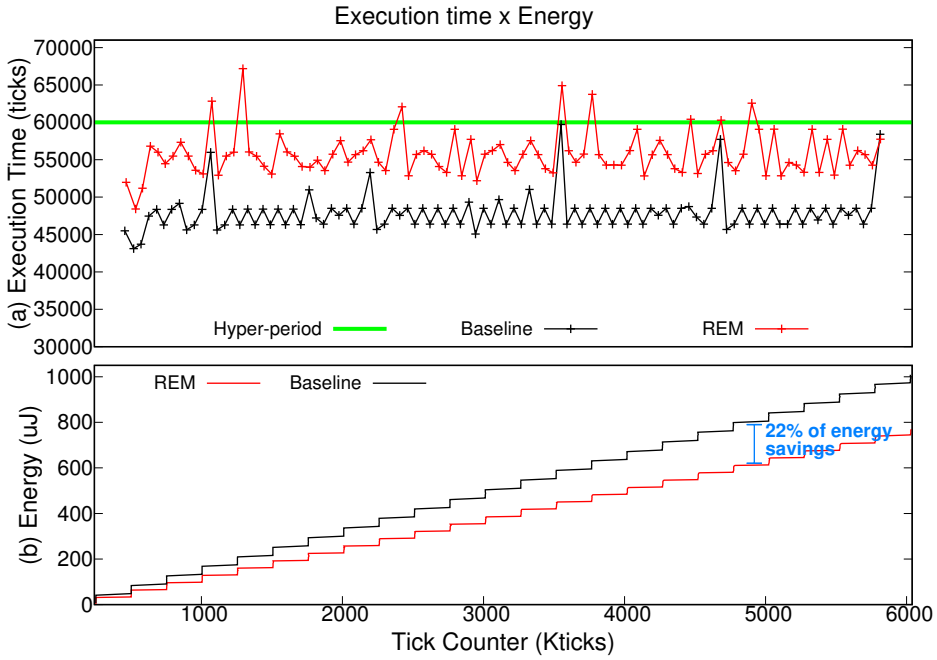


Figure 5.5 – Execution time and energy of an RT application with and without REM for 100 iterations [MRSM17a].

The hyper-period defined by syscall *RealTime* for each iteration is 60 Kticks (green horizontal line). The execution of the baseline system does not present hyper-period violations, but a slack time is observed. Using REM, after the definition of the *vf-pairs*, the slack time becomes narrower, with few violations in the hyper-period (smaller than 10% of the hyper-period value). Figure 5.5(b) plots the energy consumed by the system for the baseline and the system with REM. As the simulation advances the energy savings increases, reaching 22% after execution 100 iterations of the application. Figure 5.5(b) shows REM achieves energy savings for one RT application while keeping the execution time of the application within the expected hyper-period, with few violations. Considering periodic applications, like video decoding or some other application with RT constraints, executing for longer periods than the ones presented in the simulation, those applications have the potential to obtain significant energy savings by extrapolating both curves.

5.4.3 REM for Controlled Scenarios

The goal of this experiment is to evaluate REM in controlled scenarios. The evaluation considers the baseline system and four scenarios:

1. *BE-only* – a 4x4 many-core running t_{BE}s.
2. *RT-only* – a 4x4 many-core running t_{RT}s.
3. *RT+BE* – a 4x4 many-core running t_{RT}s and t_{BE}s, with the same number of tasks for both applications.
4. *RT+BE+mig* – a 4x4 many-core running t_{RT}s and t_{RT}s, with a configuration that REM Task Remapping fires a task migration.

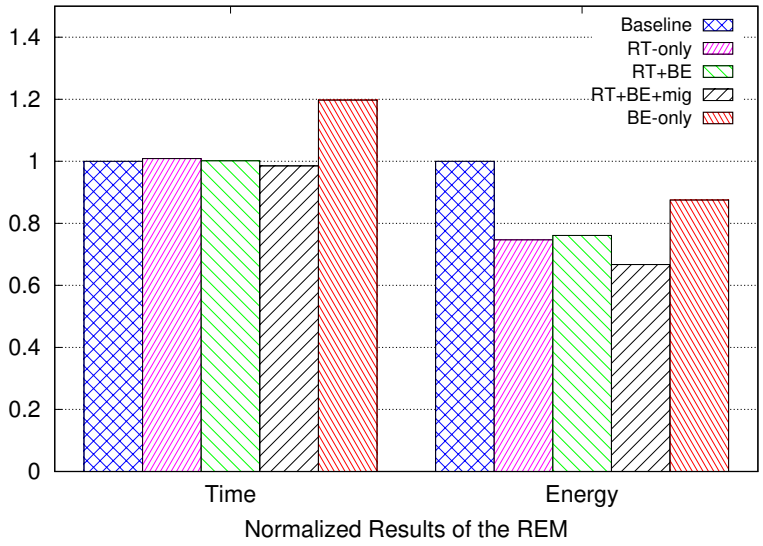


Figure 5.6 – Execution time and energy results of REM for controlled scenarios normalized regarding the baseline system.

Figure 5.6 shows the normalized results w.r.t the baseline system. The graph presents the execution time of RT applications and the consumed energy for all scenarios. The execution time of the RT applications is not disturbed by the BE applications, even when task migration occurs. Scenarios RT-only and RT+BE reduced the energy in average by 25%. It is worth noting that the effect of the task migration. Because of Task Remapping algorithm, PEs are exclusively reserved for RT, enabling the reduction of the *vf-pair* of the PE. It is possible to note a slight reduction in the execution time for this scenario. The performance of the BE applications is in effect a trade-off execution time versus leakage energy. As shown in Figure 5.3, the goal is to keep the energy of the PE running t_{BE}s in the

warm zone. In this experiment, the execution time increases 20%, and the energy reduces by 13%.

Only scenario RT+BE presents violations in the hyper-period. The number of hyper-period violations is small (less than 2%) with an amplitude lower than 10% of the defined constraint. This behavior is similar to the one presented previously.

5.4.4 Evaluation of the REM proposal with a mix of BE and RT applications in large systems

Table 5.2 presents experiments for systems having up to 144 PEs. Five many-core systems (first column), divided into different cluster sizes (second column), run the applications set. The number of t_{RT}s and t_{BE}s to process is the same of the number of SPs of the many-core (third column). For instance, the 6x6 many-core has 1 GM, 3 CMs, and 32 SPs such as the example of Figure 1.2. A burst of three or four applications starts each 1 ms. The fourth column presents the number of task migrations executed during the simulation. The fifth column shows the percentage of hyper-period violations (the execution of the applications in the baseline system does no present violations). The REM produces a small number of hyper-period violations (< 2.1 %), with an amplitude inferior to 15% of the defined constraint. This result shows that the proposed REM is suitable for RT applications. For example, few hyper-period violations decoding a video frame do not affect the latency of the applications [FSWV07]. The last column presents the energy savings, for a simulation time of 50 ms. The average energy reduction observed is 18%. This result is coherent with the previous section since the experiments mix BE and RT applications, with larger energy saving in PEs executing t_{RT}s and smaller gains in PEs executing t_{BE}s.

Table 5.2 – Violations of hyper-periods and energy savings of REM compared to the baseline system.

Many-core size	Cluster size	# of tasks for each type	# of task migrations	Violations of hyper-periods	Energy savings
6x6	3x3	32	3	1,47%	15%
8x8	4x4	60	6	2,09%	18%
9x9	3x3	72	8	1,84%	20%
12x12	3x3	128	8	1,87%	18%
12x12	4x4	135	14	1.97%	18%

To put in perspective these results, Figure 5.7 shows the energy consumption of the system for the baseline and the system with REM. At the beginning of the simulation, as the zoom-in shows, the dominant events at the beginning of simulation are application and task mapping (at 2,000 Kticks all applications are already running). Next, the manager

PEs can set the *vf-pairs* for each task because SPs are executing tasks and sending observing messages containing energy and RT data. After this period, the gap between the curves increases because most of the management decisions are taken at the beginning of the tasks' execution. As the simulation advances, e.g., at 360 ms (90.000 Kticks) the energy savings reaches 28%. This result is achieved because all SPs are running a t_{RT} with utilization values that allow the REM to save energy, optimizing the results.

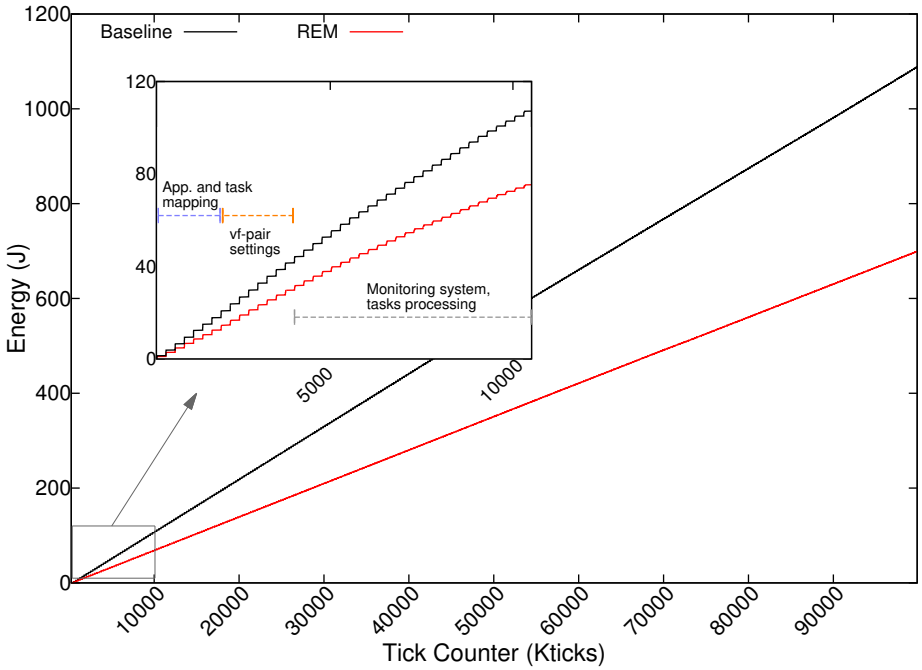


Figure 5.7 – Energy consumption of a 6x6 many-core.

5.4.5 Comparison of the REM with Related Works

Table 5.3 compares the results of related works with REM qualitatively. The fifth column of Table 5.3 is the Authors' baseline reference to present their results. The principal advantage of the REM is scalability. The distributed management enables to apply the proposal to large systems (144 PEs). Design-time analysis of the applications set leads the system management to achieve better results than the ones without awareness of the application set. However, such systems do not support the admission of new applications or adaptive workloads.

The works without design-time analysis of the application set include Maiti et al. [MKP15], Haghbayan et al. [HRW+14], Singh et al. [SDI15], and Yu et al. [YSH14]. Maiti et al. [MKP15] focus on process variability and near-threshold voltage, using thermal constraints. The goal of Haghbayan et al. [HRW+14] is to optimize performance under power constraints. Yu et al. [YSH14] address adaptive workloads (e.g. H.264/SVC).

Considering works with design-time analysis of the application set, Li et al. [LJJ13] make mapping and DVFS for heterogeneous many-core. Jung et al. [JLK+14] adopt as a baseline a worst-case mapping, explaining the larger energy reduction. Singh et al. [SDI15] propose both design-time based and a runtime management an HPC many-core to optimize energy and performance.

Wei et al. [WLW+17] aim to maximize throughput using a CPU+FPGA platform. The technique of merging pipeline and DVFS is used to allow the execution of more frames simultaneously under the same power cap for a required latency, but the same technique could be applied for energy savings purposes.

Table 5.3 – Qualitative comparison between related works.

Proposal	BE or RT tasks?	Scalability	Design-time analysis of applications	Authors' baseline for comparison	Design goals
Li et al. [LJJ13]	RT	No	Yes	Worst and Best case of mapping	Best: +6.3% energy; Worst: -5% energy
Singh et al. [SDK13]	RT	No	Yes	Other works	Up to -52% energy
Das et al. [DKV14]	RT	No	Yes	Other works	Up to -40% energy
Das et al. [DAHM16]	RT	No	Yes	Linux on-demand	-15% energy
Jung et al. [JLK+14]	RT	No	Yes	Worst-case mapping, no management	Up to -36% energy
Maiti et al. [MKP15]	RT	No	No	No process variability, simple DVFS	-15% energy
Yu et al. [YSH14]	RT	No	No	No DVFS, with thermal constraints	Up to -31.5% temperature
Haghbayan et al. [HRW+14]	Both	Yes	No	No management, no constraints	+46% performance
Singh et al. [SDI15]	RT	Yes	Both (design-time and runtime)	No DVFS	-15.8% and -5.8% energy for runtime and design-time approaches, respectively. +5.8% performance for design-time approach*
Wei et al. [WLW+17]	RT	N.A.	Yes	CPU-only platform	+37% throughput in average
This	Both	Yes	No	No DVFS	-18% energy in average

*the average result of performance for runtime approach is not presented

Authors develop their proposals using distinct frameworks (Table 5.1), so a direct comparison is difficult. Most of the related works are unable to fulfill the assumptions of this proposal because REM targets dynamic workloads for many-core systems. Nonetheless, the proposals from Haghbayan et al. [HRW+14] and Singh et al. [SDI15] have similar features to this Chapter proposal: scalability and the applications set is previously unknown.

Table 5.4 compares REM to Haghbayan et al. [HRW+14] and Singh et al. [SDI15] works. The results from Singh et al. [SDI15] were obtained from the paper because the methodology for defining the baseline system for comparison in both works is similar, despite the different experimental setup. Table 5.4 reports results of the runtime approach for a 3x8 system. To achieve a fair comparison with Haghbayan et al. [HRW+14], it is considered: (*i*) the same static mapping in a 6x6 many-core; (*ii*) the restriction from Haghbayan et al. [HRW+14] work of one t_{BE} per PE; (*iii*) four applications (24 tasks) composes the test case; (*iv*) the baseline for both works is the same system without DVFS (6x6 many-core with

the same static mapping); (*v*) both works employ the same energy and power observing and estimation. As REM spends 160 mW for this controlled scenario, the power cap in Haghbayan et al. [HRW+14] work is set to 160 mW.

Table 5.4 – Quantitative comparison between related works regarding energy savings.

BE or RT tasks?	Singh et al. [SDI15]	Haghbayan et al. [HRW+14]	REM
RT	15.8%	0%	25%
BE	N.A.	-5.78%	10.55%

Results show larger energy savings compared to [HRW+14] and [SDI15] because REM employs different strategies according to the application type. Note that Singh et al. [SDI15] also include the goal of maximizing utility in a context of HPC, so if the goal of both approaches were similar, the energy saving of both works would be closer. Haghbayan et al. [HRW+14] execute t_{RT}s but do not actuate on them, so there are no energy savings for these applications. Considering t_{BE}s, due to the power cap constraints, Haghbayan et al. [HRW+14] increase the execution time and thus the consumed energy. Compared to the baseline system, the execution time grows 20.81% and 5.78% for Haghbayan et al. [HRW+14] and REM, respectively. The main difference between REM and [HRW+14] is the DVFS granularity. Haghbayan et al. [HRW+14] work applies DVFS at the application level, while REM applies DVFS at the task level, i.e., tasks of the same application can execute at different *vf-pairs*. On the other hand, Haghbayan et al. [HRW+14] approach is more flexible concerning power cap. Haghbayan et al. [HRW+14] work can follow a power cap below and above 160 mW while REM is not able to adapt the heuristics according to a power cap.

5.5 Final Remarks

This Chapter proposed a hierarchical Decision module by using the Observing and Actuation structure previously presented (Chapters 3 and 4, respectively) to deploy the Observe-Decide-Act paradigm in a Resource Management targeting RT and BE applications, named as Runtime Energy Management - REM.

REM assumes different strategies for saving energy of RT and BE tasks. REM explores the slack time of RT applications to reduce the consumed energy in many-core systems while executing RT applications to meet the applications' constraints. Regarding BE tasks, REM assumes energy zones calibrated in such a way to both stimulate the PE to run in the energy efficiency zone and respect a power cap at the PE level. As shown in the results section, the proposal is scalable, with similar energy savings for different system sizes (from 36 up to 144 PEs), and produces a small number of hyper-period violations.

Summarizing, the main original contribution of this Chapter is the integration of a comprehensive set of techniques for RM and the execution of RT and BE applications respecting the RT constraints without design-time analisys.

Future works are as follows: (*i*) include other actuation techniques, such as power gating; (*ii*) evaluate the approach for SOI technologies; (*iii*) include additional cost functions to the REM heuristic to enable more than one RT task running at the same SP.

6. DECIDING - MULTI-OBJECTIVE RESOURCE MANAGEMENT

The typical workload of many-core systems, such as servers for cloud computation, produces peaks and valleys of resources utilization throughout the time (Section 1.2). The power capping limits the full system utilization in a workload peak, but also creates power slack to allow another policy for resource management (RM) in a valley phase (Figure 1.1). Related works do not consider this workload issue, proposing RM with hardened goals.

This Chapter proposes a hierarchical adaptive Multi-Objective Resource Management (MORM) for many-core systems under a power cap. MORM aims dynamic workloads with peaks and valleys of utilization. The hierarchical approach allows clusters of processing elements (PEs) to execute applications according to different objectives simultaneously. A cluster can drive the PEs to optimize either performance or energy. *MORM* system manager can dynamically shift the goals of a cluster according to the workload behavior.

Note that, unlike the previous Chapter, real-time (RT) applications are not in the *MORM* scope. Although *REM* and *MORM* share most of the features of Observing and Actuation states as well as the hierarchical organization, they are distinct *Decisions* states under unrelated assumptions and their decision algorithms are not connected.

MORM is a *multi-objective* and *adaptive* management. *Multi-objective* means that *MORM* addresses power, energy, and performance concomitantelly (Definition 3 on page 27). *MORM* is also *adaptive* because it can optimize at runtime either performance or energy of a cluster (Definition 4 on page 27). *MORM* is aligned with important trends for resource allocation [SDMI17]: (*i*) multi-objective resource allocation; (*ii*) consideration of communication and computation loads; (*iii*) large-scale architectures.

The *contributions* of this Chapter are as follows:

- An approach that can dynamically adapt the system to the frequent changes of system goals due to the workload variation;
- A comprehensive approach performs the trade-off between conflicting goals: performance or energy;
- A hierarchical organization which distributes the workload in clusters allowing applications to run according to different goals simultaneously;
- The reference many-core (Section 1.6) employs an RM from related works into the reference many-core for comparison purposes;
- The results show that *MORM* achieves better results in performance and energy compared to related works.

Section 6.1 reviews related works. Section 6.2 introduces the application model and presents how to generate the values associated to the application task graph. Section 6.3 overviews MORM. Sections 6.4 and 6.5 detail the *MORM* decisions at the system and cluster levels, respectively. Section 6.6 presents the experimental results, and Section 6.7 concludes this Chapter.

6.1 Related Works

Table 6.1 summarizes related works concerning the features required by RMs: (*i*) management of the applications admission (AA); (*ii*) task mapping/remapping (TMap and TR); (*iii*) DVFS control; (*iv*) power cap; (*v*) hierarchical management in clusters to ensure scalability; and (*vi*) adaptive goals, i.e., the RM can manage the applications to meet distinct goals dynamically.

Regarding application admission, some works [KP15, OA17] present frameworks for deciding the best number of tasks by adapting the application parallelism to the available resources on the system or the power capping. In [RHK+15], the application enters the system if there are available processors, but can also be killed suddenly if the power overcomes the capping. *MORM* can remap running tasks to share PEs and map incoming applications in a reduced number of PEs to open power and resources room.

Table 6.1 – Features of comprehensive RM for many-core systems.

Proposal	AA	TMap	TR	DVFS	Pwr cap	Cluster	Adaptive Goal
Olsen et al. [OA17]	✓	✓	✗	✗	✓	✗	✗
Zhang et al. [ZH16]	✗	✓	✓	✓	✓	✗	✗
Kapadia et al. [KP15]	✓	✓	✗	✓	✓	✗	✗
Rahmani et al. [RHK+15]	✗	✓	✗	✓	✓	✗	✗
This	✓	✓	✓	✓	✓	✓	✓

AA: Application Admission; TMap: Task Mapping; TR: Task Remapping

The application needs to be mapped once it enters into the system. Mapping heuristics have inherent challenges such as disturbances on other applications, traffic, and scalability so that mapping is an NP-hard problem [SSKH13]. In general, for homogeneous many-cores, the best runtime mappings assuming one task per PE assign tasks in a continuous shape to avoid network congestion and optimize performance [FRD+12]. On the other hand, the assignment of more than one task per PE minimizes the hop number and the number of active PEs and leads to energy savings [MOSM15]. Some proposals [KP15, RHK+15] deploy distinct algorithms to map one task per processor in square shapes. *MORM* takes advantage of the cluster organization to propose two lightweight mapping algorithms that enable a fast adaptation between energy and performance goals.

Task remapping employs task migration to deal with the availability of resources dynamically. When a task arrives to execute, a mapping review of running tasks delivers a better result than just mapping the task on the available resources. As works [RHK+15, KP15] do not support multi-tasking mapping, task remapping brings no significant advantages concerning power and resources, and then it is not employed. *MORM* uses two task remapping approaches, *join* and *split*, to perform adaptability for optimizing the cluster or the system to a new goal according to the workload. The *join* remapping stimulates the PE sharing to save energy by mitigating the communication between tasks and creating more idle PEs for power gating. The *split* remapping spreads the tasks in more PEs to optimize performance.

As soon as tasks are running, DVFS is the power actuator to trade-off energy and performance at the task level. Besides that, some works [RHK+15] employ DVFS at the application level to optimize the power with PID controllers. Alternative approaches for power capping propose DVFS assignment through reinforcement learning [CM15] and probabilistic [PKS+17] techniques. However, a recent work [ZH16] shows that a joint actuation between DVFS and resources allocation boost the performance under a power cap. The DVFS approach employed in this proposal works with task mapping/remapping heuristics to maximize the adaptive goal as well as can also identify opportunities to save energy according to the task phase.

Finally, the main feature and contribution of *MORM* is the adaptability according to the workload. No related works assume a dynamic change in the workload, i.e., regardless the system utilization the control strategy follows the same goal. While the workload is low, the power slack allows the control to decide an optimization for saving energy, or a boosting on performance. At peaks of workload, the cluster hierarchy allows the control to set some clusters to boost some applications while still keeping the power capping by slowing others clusters down.

6.2 Application Model

The application model adopted is derived from the generic model described in Chapter 1. Applications are modeled as directed acyclic task graphs, *A=(T, E, Q, P)*, where:

- the vertex $t_i \in T$ is a task.
- the directed edge $e_{ij} \in E$ is the communication between tasks t_i and t_j.
- the value q_i associated to each task t_i corresponds to the power consumption of t_i when executing in a PE without CPU sharing at the nominal voltage and frequency.

- the value p_{ij} corresponds to the power consumption of a communicating task pair t_i-t_j when executing in the same PE at the most energy efficient voltage and frequency (Section 6.5.1).
- the set $Q = \{\sigma\}$ is the application power from q_i values given by $\sigma = \sum_{i=1}^{n} q_i$, considering that each SP executes one task and n is the number of tasks of the application.
- the set $P = \{\rho_1, \rho_2, ..., \rho_n\}$ is the application power obtained from q_i and p_{ij} considering a multi-task mapping.

6.2.1 Application Power Profiling

Obtaining q_i and p_{ij} (and consequently Q and P sets) requires a design-time evaluation of the application set. The applications are simulated individually with the minimum size for allocating all application tasks to avoid disturbances from other sources. All SPs executes in the nominal voltage and frequency to get q_i and in the most energy efficient voltage and frequency (Section 6.5.1) to derive p_{ij}

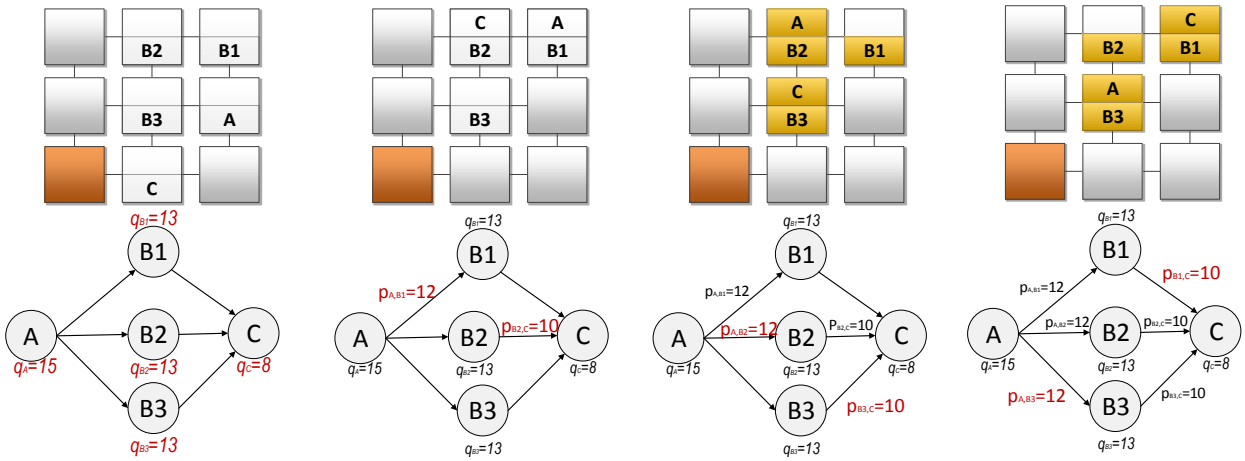


Figure 6.1 – Steps to obtain q_i and p_{ij} for a given application.

Figure 6.1 shows all steps to produce q_i and p_{ij} for a given application. To obtain q_i, the application tasks are manually mapped so that an SP executes one task in a contiguous shape. After the end of the simulation, q_i corresponds to the worst-case value among all power sampling reported by the SP where t_i is allocated.

Next to derive a p_{ij}, the application tasks are mapped so that an SP can allocate up to two tasks. The communication tasks t_i and t_j share an SP while the remaining tasks run individually in another SPs. Similarly to q_i, p_{ij} is the worst-case value among all power samplings reported by the SP where t_i and t_j are allocated. Note that, generating all p_{ij} values

require multiple simulations to test all e_{ij} values assigned exclusively to an SP. For instance, the application from Figure 6.1 has six edges and needs three simulations to extract all p_{ij} values. For this example, the first task mapping could allocate t_A and t_{B1}, and t_{B2} and t_C to share SPs for deriving p_{A-B1} and p_{B2-C}. Second simulation reports p_{A-B2} and p_{B3-C} and the last one generates p_{A-B3} and p_{B1-C}.

For three or more tasks per PE, the decision assumes E_{max} values (Section 4.4) as reference.

6.3 Multi-Objective Resource Management - MORM

Figure 6.2 overviews the *MORM* concerning the hierarchical organization, which adopts the observe-decide-act paradigm [DJS15] to manage the system. Observing data follows a bottom-up direction. SPs send data (e.g., energy, temperature, CPU utilization, NoC congestion) to their CMs. Each CM transmits to the GM the current power consumption of its cluster. Manager PEs (CMs and GM) take *decisions* at cluster and system levels. At the cluster level, a given CM may decide to modify, for example, the voltage-frequency pair of a set of SPs. At the system level, the GM may change the operation mode of a given cluster. Thus, *actuation* follows a top-down direction, with actions send from the GM to CMs and from CMs to SPs.

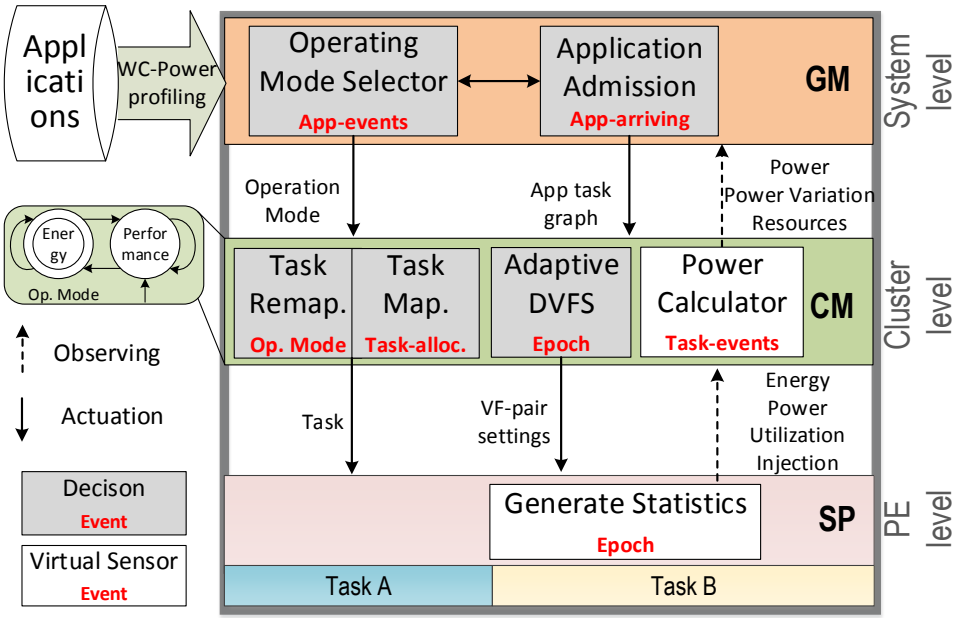


Figure 6.2 – General MORM overview.

Clusters may operate in one of two operation modes:

Definition 17. *Performance mode* - the cluster optimizes the resources to minimize the execution time of the running applications.

Definition 18. *Energy mode* - the cluster optimizes the resources to improve the energy efficiency.

In Figure 6.2, gray and white boxes correspond to *decision* algorithms and *virtual sensors*, respectively. Labels inside each box correspond to the event responsible for firing both the decisions algorithms and observing data transmission. Events are classified into four classes:

Definition 19. *Application events* - correspond to external notifications to the GM that an application is ready for admission or a CM reports the end of an application to the GM.

Definition 20. *Task events* - correspond to the moment that CM maps or remaps a task to an SP or the moment an SP reports to a CM that a task finished its execution.

Definition 21. *Operation Mode events* - correspond to the moment that GM changes the operation mode of a given cluster.

Definition 22. *Epoch events* - correspond to a periodical hardware interruption, where SPs report the observing data to its CM.

The system level management is in charge to take decisions at the application level, to maintain the power cap, and to choose the operation modes (Section 6.4). The cluster level management controls the DVFS, and task mapping and remapping (Section 6.5).

6.4 MORM System Level Decisions

MORM allows a new arriving application to execute *iff* the application does not exceed the power cap, and the system has available resources. First, the GM verifies the additional power required by the incoming application (Section 6.4.1). After, the GM verifies the resources availability (Section 6.4.2). If none of the conditions are satisfied, *MORM* can modify the operation mode of the clusters to find room for the incoming application.

6.4.1 Operating Mode Selector

The GM decides the operation mode of each cluster based on different power values from the system, cluster and application. The power values used in the Operating Mode Selector are the following:

Definition 23. *Application Power Performance* - prediction of the application power by using the performance mode (Q set).

Definition 24. *Application Power Energy* - prediction of the application power by using the energy mode (*P* set).

Definition 25. *Cluster Power* - the sum of all monitored power samples in the SPs belonging to the CM.

Definition 26. *Cluster Power Variation* - effect on the *Cluster Power* when the cluster operation mode changes (defs. 17 and 18, on page 111). [1]

Definition 27. *System Power* - the sum of all *Cluster Power* values.

Definition 28. *System Power Cap* - the upper bound value of power.

MORM employs proactive actuations to respect the power cap based on the estimation of power disturbances due to the application events (def. 19) and operation mode events (def. 21). Application events modify the total system power and *Operating Mode Selector* takes decisions by evaluating the expected power impact due to these events. If an application finishes its execution, the CM reset the counters of the SPs where the application was mapped and then update the GM with new power values. When an application requests admission, the GM takes decisions based on the application power estimation (def. 23). Similarly, *operation mode events* (def. 21) disturb the *cluster power* (def. 25) due to task remappings and voltage-frequency changes that the CM executes when receiving a new operation mode (Section 6.5). The GM is aware of the power disturbance from operation mode events by observing *cluster power variation* (def. 26) of each cluster.

Algorithm 6.1 is the proactive power control knob for shifting operation modes of the clusters while respecting the system power cap (def. 28) based on the amount of power disturbance induced by application events (def. 19). At the beginning of the system execution, all clusters operate in performance mode. The algorithm may update the *operation mode* of the clusters when an application requests admission into the system (lines 2-29), or it finishes the execution (lines 30-37).

The algorithm receives as inputs the application description (*app*), the set of clusters operating in energy mode ($energyCl_{set}$), and the set of clusters operating in performance mode ($perfCl_{set}$).

If *app* is requesting its admission (line 2), *MORM* estimates the increasing of power if the application is mapped in performance mode (line 3). If the estimation does not exceed the system power cap (def. 28), *app* may be admitted in performance mode (line 5). Otherwise, *MORM* estimates the increasing of power to admit the application in energy mode (line 7):

- If the estimated power is above the cap, the loop between lines 9-14 evaluates if *app* may be admitted by changing a given cluster in $perfCl_{set}$ to energy mode. Line 10

[1] Section 6.5.4 details the cluster power variation computation (page 119).

Algorithm 6.1 MORM Operating Mode Selector

```
1: Inputs: app, energyCl_set, perfCl_set
2: if app is arriving then
3:     newPwr ← sys.pwr + app.pwrPerformance
4:     if newPwr < sys.pwrCap then
5:         Allows the admission of the application in performance mode
6:     else
7:         newPwr ← sys.pwr + app.pwrEnergy
8:         if newPwr > sys.pwrCap then
9:             for each cl_i ∈ perfCl_set do
10:                newPwr ← newPwr + cl_i.pwrVariation
11:                if newPwr > sys.pwrCap then
12:                    shiftOpMode(cl_i, energy)
13:                end if
14:            end for
15:            if newPwr < sys.pwrCap then
16:                Allows the admission of the application in energy mode
17:            else
18:                Application enqueued to be admitted later
19:            end if
20:        else
21:            if energyCl_set = ∅ then
22:                cl_output ← maxAvailSPs(perfCl_set, performance)
23:                shiftOpMode(cl_output, energy)
24:                Allows the admission of the application in energy mode
25:            else
26:                Allows the admission of the application in energy mode
27:            end if
28:        end if
29:    end if
30: else                                          ▷ app finished its execution
31:    for each cl_i ∈ energyCl_set do
32:        newPwr ← sys.pwr + cl_i.pwrVariation
33:        if newPwr < sys.pwrCap then
34:            shiftOpMode(cl_i, performance)
35:        end if
36:    end for
37: end if
```

makes this estimation by using the $cl_i.pwrVariation$ (def. 26), and if it is possible to admit *app*, cl_i shifts from performance to energy mode (line 12). If the power is below the capping, the *app* may be admitted in energy mode (line 16). Otherwise, *app* is enqueued to be executed later (line 18).

- If the estimated power is below the capping, one cluster must be in energy mode to receive *app*. If there is no cluster in energy mode (line 21), the function *maxAvailSPs* finds the cluster running in performance mode with the maximum number of available processors, cl_{out}. Algorithm 6.1 shifts cl_{out} to energy mode (line 23), then *app* may be admitted in energy mode (line 24). Otherwise, a cluster is already running in energy mode and *app* may be admitted (lines 26-27).

When a given application finishes its execution (lines 31-36), the algorithm verifies if it is possible to shift a cluster from energy mode to performance mode.

6.4.2 Application Admission

After verifying the application admissibility regarding power, *Application Admission* (Algorithm 6.2) verifies the application admissibility regarding available resources and selects the cluster to map the application. The algorithm receives as inputs the application description (*app*), the app_{mode} defined in Algorithm 6.1, and the set of clusters ($sys.cl_{set}$).

Algorithm 6.2 MORM Application Admission

```
1: Inputs: app, app_mode, sys.cl_set
2: Outputs: cl_output
3: cl_set ← ∅
4: SP_min ← getMinSPsAdmitApp(app, app_mode)
5: for each cl_i ∈ sys.cl_set do
6:     if app_mode = cl_i.mode and cl_i.freeSP ≥ SP_min then
7:         cl_set ← cl_set ∪ cl_i
8:     end if
9: end for
10: if cl_set ≠ ∅ then
11:     cl_output ← maxAvailSPs(cl_set, app_mode)
12:     return cl_output
13: end if
14: cl_output ← maxAvailSPs(sys.cl_set, performance)
15: app_mode ← energy
16: SP_min ← getMinSPsAdmitApp(app, app_mode)          ▷ update SP_min
17: if cl_output.freeSP ≥ SP_min then
18:     shiftOpMode(cl_output, energy)
19:     return cl_output
20: else
21:     return ∅
22: end if
```

The algorithm starts by creating an empty set, cl_{set}, which contains the clusters candidate to receive *app* (line 3). Next, it computes the number of SPs required for executing an application according to the application mode, function *getMinSPsAdmitApp* (line 4). In performance mode, the number of SPs is equal to the number of the application tasks, and in energy mode, this value is smaller due to the CPU sharing among communicating tasks.

The loop between lines 5-9 fills cl_{set} with the clusters' identifiers that may receive *app*. At the end of the loop, if cl_{set} is not empty, the selected cluster is the one with the maximum SPs running no tasks (lines 10-13). The function *maxAvailSPs* returns a cluster identifier with an operation mode equal to the *app* mode (second parameter of the function).

If there is no SPs available in any cluster, i.e., cl_{set} is empty, Algorithm 6.2 selects a cluster in performance mode as the candidate to receive the new application (line 14). In this case, the application mode changes to energy mode (line 15) and the number of required SPs is updated (line 16). If cl_{output} can receive *app* in energy mode, *Application Admission* changes its operation mode to energy mode (line 18), and cl_{output} receives *app* (line 19). Otherwise, the function returns null, and the application waits in a queue for later admission.

6.5 MORM Cluster Level Decisions

The adaptability at the cluster level enables clusters to work at different operation modes simultaneously. This section describes the three mechanisms adopted at the cluster level: adaptive DVFS, task mapping and task remapping.

6.5.1 Adaptive DVFS

Adaptive DVFS is an adaptive threshold-based algorithm that follows the cluster operation mode. From the available voltage-frequency pairs (*vf-pairs*), the algorithm adopts three values:

Performance (VF_{perf}): nominal *vf-pair*, i.e., the highest voltage and frequency values (used when the cluster is in performance mode).

Low power (VF_{min}): lowest voltage and frequency values.

EDP (VF_{EDP}): most energy efficient *vf-pair* between VF_{perf} and VF_{min} (used when the cluster is in energy mode).

The *Adaptive DVFS* applies VF_{min} in two cases related to communication issues, regardless the operation mode:

1. HI (high injection): an SP is injecting messages on the network in a higher speed than the messages are consumed;

2. LU (low processor utilization): an SP is in idle state most of time waiting for messages.

MORM adopts a similar method to [RHK+15] to evaluate the message injection rate. The injection is the average utilization of the input buffer in the local port. The HI threshold is activated when the injection is higher than, for example, 75%. The LU threshold is activated when the utilization is below than, for instance, 25%.

Figure 6.3 illustrates an example of how *Adaptive DVFS* associates VF settings according to the operation mode by using a fine-grain DVFS to save power. The CM sets most of SPs at VF_{perf} in performance mode and at VF_{EDP} in energy mode. In this example, SPs executing tasks t_A and t_C are most of the time in the idle state, because these tasks send data to the processing tasks (t_{B1} to t_{B4}) and receive the processed data, respectively. Thus, in both performance and energy modes, the SPs executing these tasks use VF_{min}.

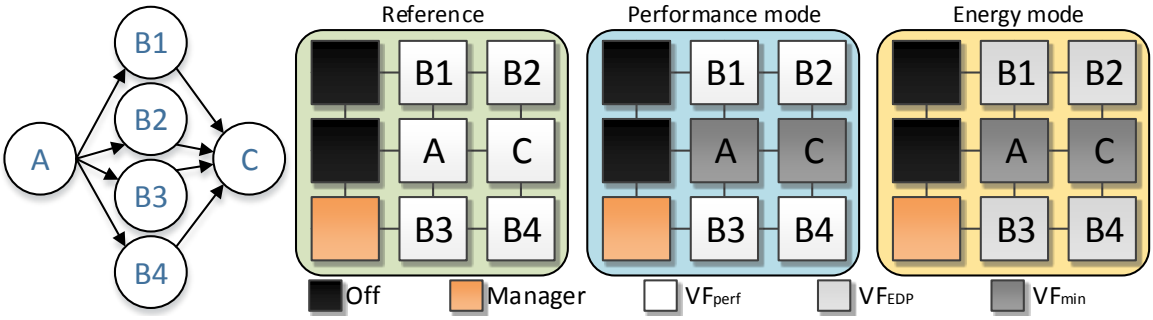


Figure 6.3 – *MORM Adaptative DVFS* selects the VF settings of a 3x3 cluster in both *operation modes* for a given application. Reference: no DVFS.

6.5.2 Task Mapping

MORM employs two mapping algorithms to meet the different goals of the two operation modes. Besides that, the mapping algorithms also provide adaptability for remapping when the cluster operation mode changes. The mapping algorithms receive the name of the operation modes. *Performance Mapping* is a single-task mapping, which maximizes the parallelism of applications and optimizes the execution time. The *Energy Mapping* employs multi-task mapping to enable the SP sharing between tasks. The *Energy Mapping* follows three constraints: (*i*) communicating tasks may be mapped in the same SP; (*ii*) parallel tasks never share the same SP (as t_{B1} to t_{B4} in Figure 6.3); (*iii*) tasks belonging to different applications never share the same SP.

The mapping algorithms benefit from the hierarchical organization to reduce the hop distance between communication task. Even if the system is large (e.g., 12x12), the

cluster size is typically 4x4 [CMMM13] to reduce the search space of mapping. The mapping algorithms select the SPs in a spiral order, which is the recommended way to transverse the SPs in 2D-mesh NoCs [BGD+04]. Also, the algorithms traverse the application graph using the Breadth-first search (BFS) algorithm. Finally, the algorithms try to find a contiguous group of free SPs in the spiral path in such a way to avoid interleaving between tasks of distinct applications when possible.

Figure 6.4(a-c) shows the task mappings for two applications in a 4x4 cluster. The numbers labeling the graph vertices sort the tasks for mapping. The system admits the blue application first (Figure 6.4(a)). For the applications in Figure 6.4(a), for example, *Performance Mapping* produces the single task mapping (Figure 6.4(b)) while *Energy Mapping* results in the multi-task mapping (Figure 6.4(c)). Note that, Energy Mapping example follows the constraints defined for energy savings.

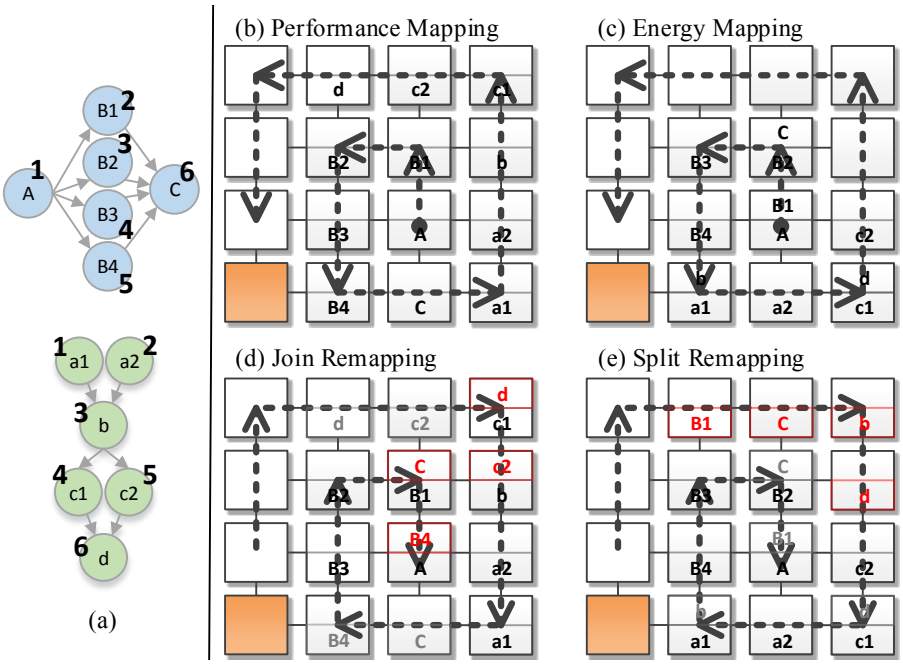


Figure 6.4 – How MORM maps tasks of (a) two applications into a 4x4 cluster in (b) performance mode and in (c) energy mode. MORM calls (d) *Join Remapping* when a cluster running tasks in (b) performance mode shifts to energy mode. Similarly, MORM executes (e) *Split Remapping* when a cluster running tasks in (c) energy mode shifts to performance mode.

6.5.3 Task Remapping

When the GM changes the operation mode of a cluster, the CM executes task remapping algorithms. When the change is from performance to energy mode, *Join Remapping* maps communicating tasks in an SP following the assumptions of *Energy Mapping*. On

the other hand, *Split Remapping* algorithm looks for tasks sharing SPs to split their execution using more SPs when changing from energy mode to performance mode.

Figure 6.4(d-e) presents the remapping results from mapping examples of Figure 6.4(b-c). Both algorithms search for tasks to migrate on the opposite spiral order. The new SPs to receive the migrating tasks are chosen in the spiral order. This search order is required to avoid fragmentation between idle SPs and running SPs concerning one application. The fragmentation between idle SPs and running SPs can happen between applications, like the one resulting mapping of Figure 6.4(d).

The main benefit of using the *Split* and *Join* remapping algorithms is to reduce the number of task migrations. For instance, *Join* remapping (Figure 6.4 (d)) needs to migrate four tasks (in red). If a new mapping targeting energy would be executed, the number of migrations would be higher (eleven in this example).

Note that, task (re)mapping and DVFS deploy a joint effort to drive the cluster according to the operation mode. *MORM* is aligned with Zhang et al. [ZH16] work, which shows the cooperation of both hardware and software power knobs to optimize performance.

6.5.4 Cluster Power Variation Computation

The CM provides the estimated power variation value to the GM if the cluster operation mode changes. The GM employs the power variation for selecting the operating mode while respecting power cap (Section 6.4.1). Algorithm 6.3 describes how CM estimates the power variation.

Algorithm 6.3 Power Calculator

```
1: Input: cl
2: Outputs: pwrVariation
3: predictedPwr ← 0
4: for each app_i ∈ cl.app_set do
5:     if cl.opMode = ENERGY then
6:         predictedPwr+ = app_i.profilePwrPerf
7:     else
8:         predictedPwr+ = app_i.profilePwrEnergy
9:     end if
10: end for
11: pwrVariation ← cl.pwr − predictedPwr
```

When the cluster is running in *energy mode*, the CM estimates the power in performance mode from profiling data of all running applications (lines 5-6) - Definition 23 (on page 112). Otherwise, the CM predicts the power of all running applications in *energy mode*

(lines 7-8). The difference between the current power value from the observing data and the predicted power variation is the estimated power variation value (line 11).

Estimate the power before task or application events with accuracy is challenging due to events like NoC traffic, CPU utilization, and memory accesses. A more accurate algorithm should consider all these features but most of them are unpredictable. Note that, Algorithm 6.3 also does not consider remapping algorithms may generate different task placements than mapping ones (Figure 6.4). Despite that, Algorithm 6.3 enables *MORM* to respect the power cap employing a lightweight algorithm since the application power profiling consider worst-case power samplings (Section 6.2.1).

6.6 Results

The experiments are conducted in an in-house clock cycle accurate RTL SystemC model of the reference many-core system. The benchmarks are DTW (6 tasks), AES (5 tasks), MPEG (5 tasks) and Synthetic (communication-bound application with six tasks). *MORM* is compared with state-of-art comprehensive system management targeting dynamic workloads. The *Performance-objective control* (*PF-only*) employs a first node selection [FRD+12] for single task mapping aiming congestion reduction [HKR+15] and, from the same authors, a feedback-based PID control that uses DVFS to follow the power-cap reference [RHK+15]. The first node selection, the mapping algorithm, and the PID control (overviewed in Section 2.7) are implemented in the reference platform.[2] Carrying out a fair comparison between *MORM* and *PF-only* requires that both systems share the following assumptions:

- Observing epoch is 250 Kticks (ideal epoch for the reference platform [Cas17]);
- Both approaches consider the router energy. *PF-only* does not consider the NoC energy in its original version [RHK+15];
- The SPs running no tasks are considered off. Section 4.2.3 discusses the issues related to this assumption, which is not used in the REM approach (Chapter 5).

The first set of experiments compares *MORM* and *PF-only* under a typical workload with peaks and valleys of utilization (Figure 1.1). This evaluation details the mapping and *vf*-settings at different moments of the execution to highlight the *MORM* contributions concerning adaptability and to distinguish the actuation approach of both managements.

The second set of experiments isolates the peak workload and the valley workload for analysis. A test case with low workload and another one with high workload are created

[2]The *PF-only* system was validated with the Authors during the internship at University of California, Irvine.

to illustrate the advantages and disadvantages of *MORM* and *PF-only* in distinct phases of the workload.

The third set of experiments evaluates cluster level results. The goal is to compare energy and performance modes considering only *MORM* Cluster Level Decisions and demonstrate that *MORM* trades energy and performance according to the operation mode.

6.6.1 Typical Workload Results

To highlight the *MORM* contributions, the same test case runs applications arriving in bursts in such a way to create a dynamic workload. At the beginning of the simulation, the many-core is loaded with two *long* applications[3]. Next, the second burst adds two *short* applications to the workload. Finally, the last burst includes more two *short* applications on the workload. Figure 6.5 presents the power consumption of *PF-only* and *MORM* under a power cap of 180mW as well as illustrates when the application bursts occur (red vertical bars) and when snapshots are taken for energy evaluation (blue vertical bars).

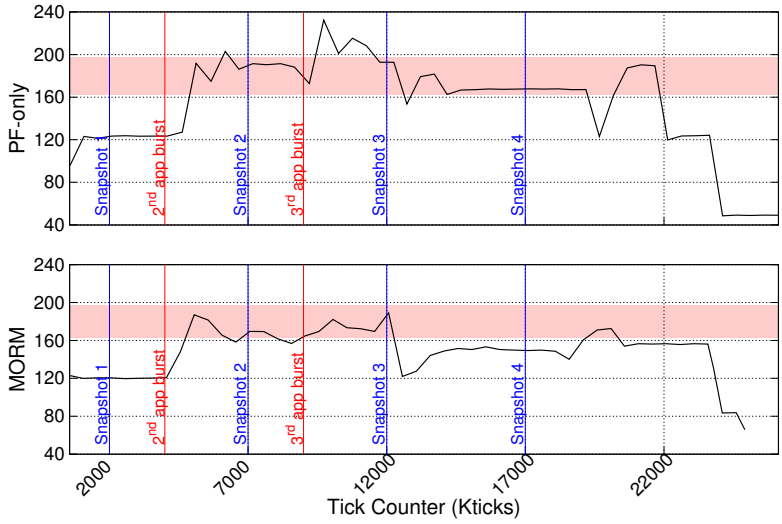


Figure 6.5 – Average Power results for *PF-only* and *MORM* running typical workload. Y-axis: power (mW), X-axis: time (in Kticks).

The proactive actuation adopted in *Operating Mode Selector* allows *MORM* to run with no capping violations, while the reactive PID controller in *PF-only* presents power violations after a burst of incoming applications. On the other hand, *MORM* underutilizes the available power because the power predictions consider the worst-case application power (Section 6.2.1) and the overshoots of power for *PF-only* might not be a problem in practice due to thermal inertia [RLH+16].

[3] *Long* and *short* applications are related to the number of iterations of the applications.

Table 6.2 presents the performance figures evaluated at each snapshot: (*i*) number of executed iterations, which corresponds to the applications performance (higher numbers corresponds to better performance); (*ii*) total consumed energy; (*iii*) occupancy, rate between the active SPs over total number of SPs; (*iv*) CPU utilization, SPs average utilization including the SPs turned off (*dark*). Figure 6.6 presents the task mapping at each snapshot. Note that *PF-only* adopts centralized management (1 CM), single task mapping, and single DVFS mode per application. *MORM* adopts distributed management (4 CMs), multi-task mapping, and fine grain DVFS.

Table 6.2 – Data from simulationsnapshots.

Metric	**Snapshot 1**		**Snapshot 2**		**Snapshot 3**	
	PF-only	*MORM*	*PF-only*	*MORM*	*PF-only*	*MORM*
$app1_{long}$ Iter.	54	52	194	189	301	300
$app2_{long}$ Iter.	36	34	144	141	224	235
$app1_{short}$ Iter.	-	-	366	321	860	844
$app2_{short}$ Iter.	-	-	18	28	95	130
$app3_{short}$ Iter.	-	-	-	-	46	59
$app4_{short}$ Iter.	-	-	-	-	27	40
Energy (mJ)	117.9	94.27	409.7	392.36	715.58	734.13
Occupancy	11/35	11/32	22/35	16/32	33/35	22/32
CPU Utilization (%)	22	23	38	43	54	55

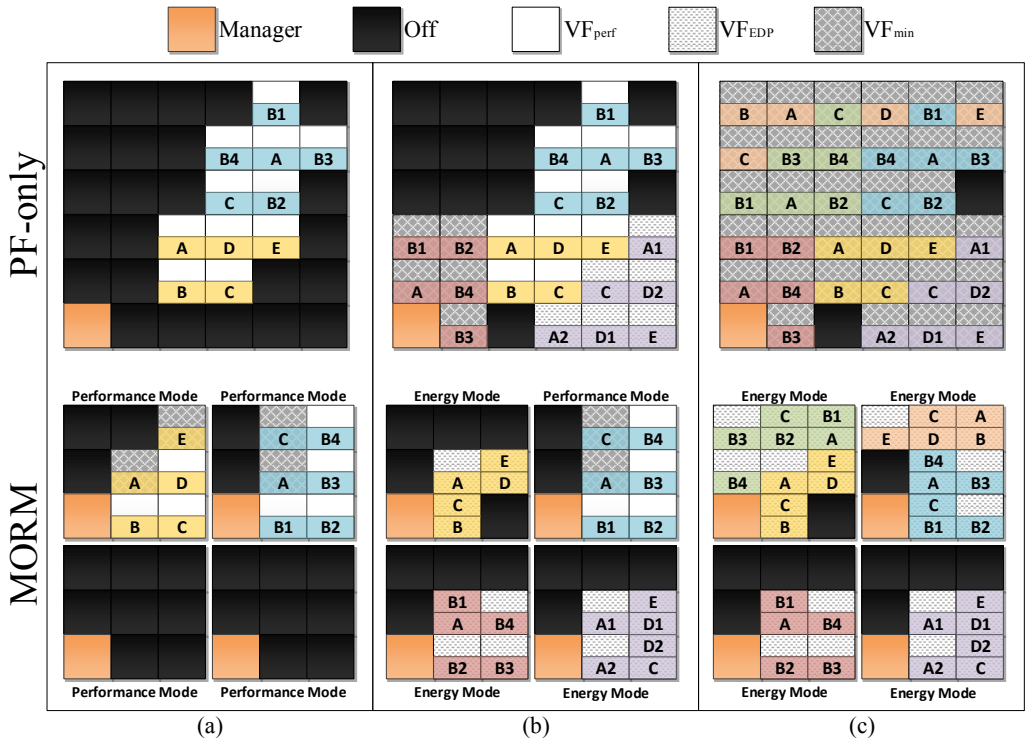


Figure 6.6 – System snapshots (6x6 many-core) taken according the moments highlighted in Figure 6.5 to detail the task allocation and *vf*-settings.

Snapshot 1 (Figure 6.6 (a)) shows the system status after the first application burst. Due to the low workload at this moment, *PF-only* and *MORM* execute the applications at

VF_{perf} and still produce power slack (Figure 6.5). The performance of the applications at Snapshot 1 is slightly better with *PF-only* due to the adopted mapping heuristic. Note that *MORM* applies VF_{min} in the SPs with HI and LU thresholds (e.g., tasks t_A and t_E) to save energy even with the cluster in performance mode (observe this result in Table 6.2).

At Snapshot 2, two new applications are running (Figure 6.6 (b)). *MORM* distributes the workload between clusters, with three ones running in energy mode and one in performance mode (with some tasks in VF_{min}). *PF-only* selects two applications to run in VF_{perf} ($app1_{long}$ and $app2_{long}$), one in VF_{EDP} ($app1_{short}$), and one in VF_{min} ($app2_{short}$). As a result, the performance of the *long* applications remains slightly better in *PF-only*. However, $app2_{short}$ is penalized due to the selection of VF_{min} by *PF-only*. On the other side, *MORM* selects the energy mode for the two *short* applications due to its multi-objective cost-function (trade-off energy-performance). MORM consumed energy is still lower because the lower system occupancy reduces the energy from the leakage power. The smaller occupancy results in higher SPs utilization due to the multi-task mapping. The main highlight from Snapshot 2 is that *MORM* enables all applications to run with a good performance while *PF-only* penalizes the performance of one application to run at VF_{min} to respect the power cap.

Snapshot 3 (Figure 6.6 (c)) illustrates both systems after the third application burst. Due to power cap violation at the third burst using *PF-only* (Figure 6.5), all SPs shift to VF_{min}. Such action penalizes the performance of all applications in *PF-only*. In *MORM*, shifting all clusters to energy mode is enough to cap the power because the system occupancy is smaller than *PF-only*. The energy is higher in *MORM* at Snapshot 3 because 22 SPs are running in VF_{EDP}, while in *PF-only* 33 SPs are running in VF_{min}. Snapshot 3 reveals the benefit of adopting multi-objective management in a peak of workload: *MORM* respects the power capping without penalizing the applications performance.

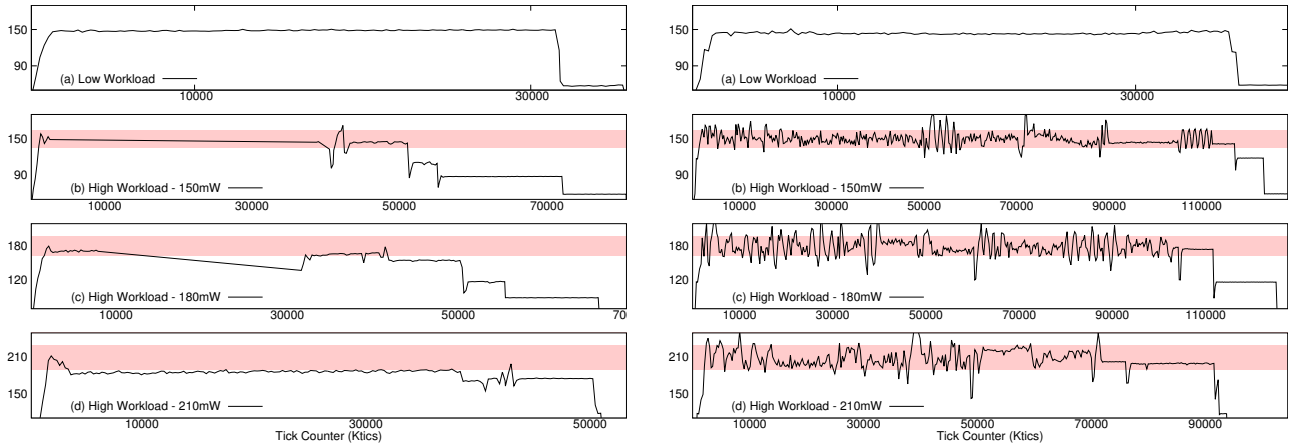
At the end of the execution of this scenario, *MORM* reduced the energy consumption by 2.6% and the execution time by 5.7%. Overall, *MORM* presents the lower variation in the applications performance when the workload varies due to joint actuation between remapping and DVFS. An advantage of *MORM* is the fact that it may execute more tasks than *PF-only* due to the multi-task mapping. Observe in Figure 6.6(c) that, since some processors are off, the system manager could admit new applications if power were available.

6.6.2 Low and High Workload Evaluation

The workload variation presented in the previous Section does not allow an isolated analysis of the peaks and valleys concerning *MORM* and *PF-only*. Thus, test cases are created to isolate the peaks and valleys behavior by eliminating the workload variation. In the *low workload* test case, the many-core is loaded at the beginning of the simulation

with *long* applications so that the number of tasks is around 50% of the number of SPs. In the *high workload* test case, the many-core is loaded at the beginning of the simulation with *long* applications so that the number of tasks is around 125% of the number of SPs. The *low workload* test case corresponds to the valley behavior while the *high workload* corresponds the peak one. Relaxed and restricted power caps are imposed to the *high workload* test cases to evaluate *PF-only* and *MORM* under restrictions of power and resources simultaneously. Accordingly, the experimental setup of this Section contains four scenarios: (*i*) *LW* – low workload; (*ii*) *HW*210 – high workload with a power cap equal to 210 mW; (*iii*) *HW*180 – high workload - 180 mW power cap; (*iv*) *HW*150 – high workload - 150 mW power cap.

Figure 6.7 illustrates the average power for the four scenarios concerning *MORM* and *PF-only*. The first row of power graphs compares the power for the *LW* scenario, while the remaining three rows show the *HW*s ones. The red shaded area represents the power cap.



(a) MORM (b) PF-only

Figure 6.7 – Average Power results for *MORM* and *PF-only* running low and high workloads. Y-axis: power (mW), X-axis: time (in Kticks).

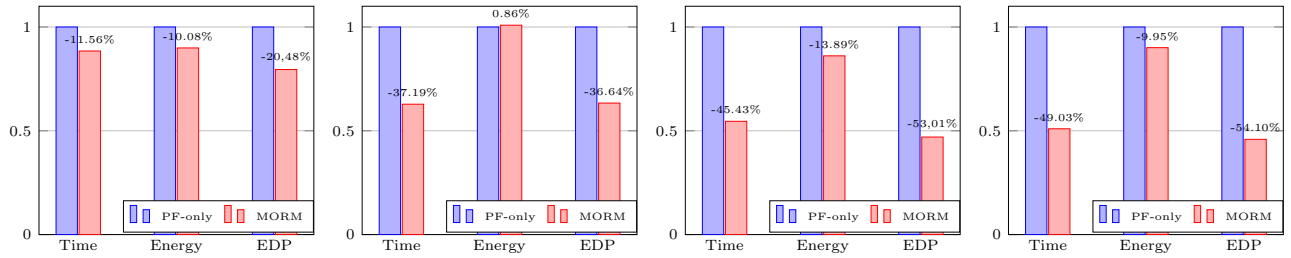
In the *LW* scenario, the power graphs present a similar behavior because all applications execute in VF_{perf} in *PF-only*, and all clusters are in *performance mode* in *MORM*. Therefore, the differences concerning the power are mostly related to the task mapping, as depicted Figure 6.6(a), which is the snapshot of a workload valley. In the *HW* scenarios, both RMs respect the power cap. Note that the execution time is inversely proportional to the power cap, i.e., the execution time reduces when the power cap increases since the power constraint is relaxed.

The *MORM* power curves in *HW* scenarios present power underutilization. This behavior is also identified in an utilization peak in the *typical workload* scenario. As mentioned in the previous Section, the power underutilization increases with the power cap due to the worst-case power predictions and to proactive-only actuations. At 210 mW power cap, all clusters run in energy mode to open room for all applications. As a consequence, *MORM*

cannot take advantage of a relaxed power cap to speed-up some cluster, explaining the larger power underutilization. With smaller power caps (150 and 180 mW), the power curves stay longer and close to the cap because the *Application Admission* proactively blocks the allocation of a new application to avoid power violation. At 150 mW and 180 mW power cap, a period without samples is observed in *MORM* (5000-35000 Kticks in 150mW, and 10000-30000 Kticks in 180mW) but without power cap violations.[4]

Concerning the power curves in *PF-only* for *HW*, the power stays around the cap most of the time despite some overshoots. The unstable behavior around the cap line in *PF-only* occurs because the PID control is constantly adjusting the vf-setting without finding a steady state.[5] This behavior in *HW* has two reasons: (*i*) the actuation at the application level creates a larger impact in power compared to the SP grain assumed in *MORM*; (*ii*) the NoC traffic has an intrinsic correlation with *vf*-settings [RHK+15] and is too high when the system is fully loaded.

The X-axis in Figure 6.7 shows that *MORM* executes all applications faster than *PF-only* for all cases. Figure 6.8 compares *MORM* and *PF-only* regarding execution time, energy and EDP (Energy-Delay Product). Data is normalized according to *PF-only*.



(a) Low Workload (b) High Workload - power cap: 150mW (c) High Workload - power cap: 180mW (d) High Workload - power cap: 210mW

Figure 6.8 – System level results concerning execution time, energy and EDP. The normalized histograms depict the results for the scenarios that represent valleys (a - low workload) and peaks (b-d - high workload) of the workload concerning power capping variation.

In *LW* (Figure 6.8 (a)), *MORM* is 11% faster than *PF-only* because the adoption of a centralized manager increases the NoC traffic traversing the applications. For instance, for the same scenario with an observing epoch eight times larger, *PF-only* is 3.22% faster than *MORM* and 16.72% faster than the *PF-only* with the original epoch settings. Since the number of tasks and the number of active SPs are the same for both RMs, the energy decreases at a similar rate than execution time: 10%. As a result, *MORM* is 20% more efficient in *LW* because it saves time and energy.

In *HW* scenarios, the difference in the execution time is higher due to the cluster organization adopted by MORM, and the reduced traffic due to observing. Besides that, *MORM* can execute all applications simultaneously whether respecting the power cap.

[4] the power graphs per cluster in the Appendix show that the power cap is met.
[5] The Appendix presents detailed graphs of power for *PF-only* where the actuation moments are visible.

Some applications must wait for allocation in *PF-only* because the number of tasks overcomes the number of SPs and the mapping in *PF-only* allows one task per SP. Moreover, *PF-only* selects some applications to run in VF_{min} (as depicted in Figure 6.6(c)) while *MORM* can reduce the power and delays by joining tasks in the same SP running in VF_{EDP}. Although *MORM* total execution time is significantly lower than *PF-only*, the energy consumption savings do not follow the same trend. The reason comes from the fact that *PF-only* executes most of the tasks in VF_{min}, which has the lowest leakage, and demands one task per PE while *MORM* runs most of the time in VF_{EDP}, which has an intermediate leakage but uses fewer PEs due to the multi-tasking mapping.

According to Figure 6.8, in the relaxed power cap (*HW*210) the execution time difference is higher because *MORM* allocates all applications at the beginning of the simulation. This evidences the effect of the single task mapping limitation in *PF-only*. In *HW*150 and *HW*180 power caps, *Application Admission* forces applications to wait for allocation while no power room is available.

Concluding, *MORM* is superior at workload peaks, resulting in smaller execution time and energy. Also, even in peaks of workload *MORM* has space for optimization because the power stays most of the time below the cap. In low workload scenarios, both RMs present similar results.

Network Traffic Analysis

Figure 6.9 illustrates the control messages flow in *PF-only* and *MORM*. The hierarchical organization reduces the traffic around the manager PEs in *MORM* by avoiding the bottleneck in the centralized manager. This Section evaluates the network traffic for *LW* and *HW*210 power caps.[6]

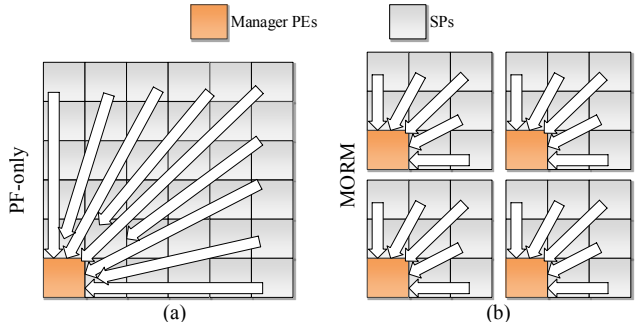
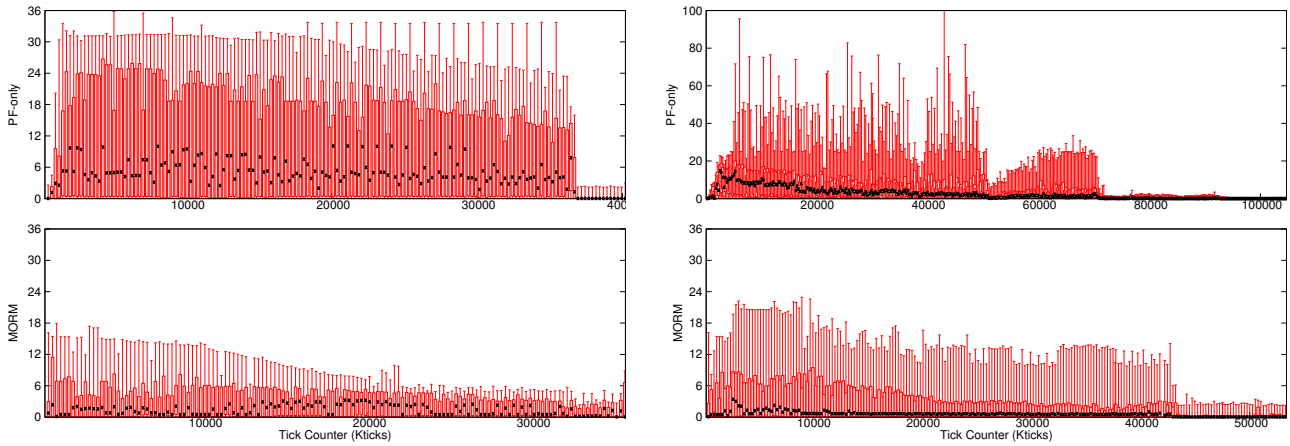


Figure 6.9 – Traffic flow of observing messages in (a) *PF-only* and (b) *MORM*.

As explained in Section 3.3.2, the router injection corresponds to the buffer utilization in the local ports. It is considered as a high injection rate when the buffer utilization reaches 75%. The routers report the percentage of cycles that the input buffer stays with high utilization in the epoch.

[6] The graphs for others caps are presented in the Appendix.

Figure 6.10 presents box plots related to the router injection. As each SP reports its router injection, box plots summarize the router injection of all SPs along the execution time. Overall, the median value (black points in the graphs) is lower than 20% for all cases. However, the variation in the third quartile demonstrates a large dispersion of the injection rate, with worst-cases reaching values 40 times larger than the median value.



(a) Low Workload (b) High Workload - 210 mW

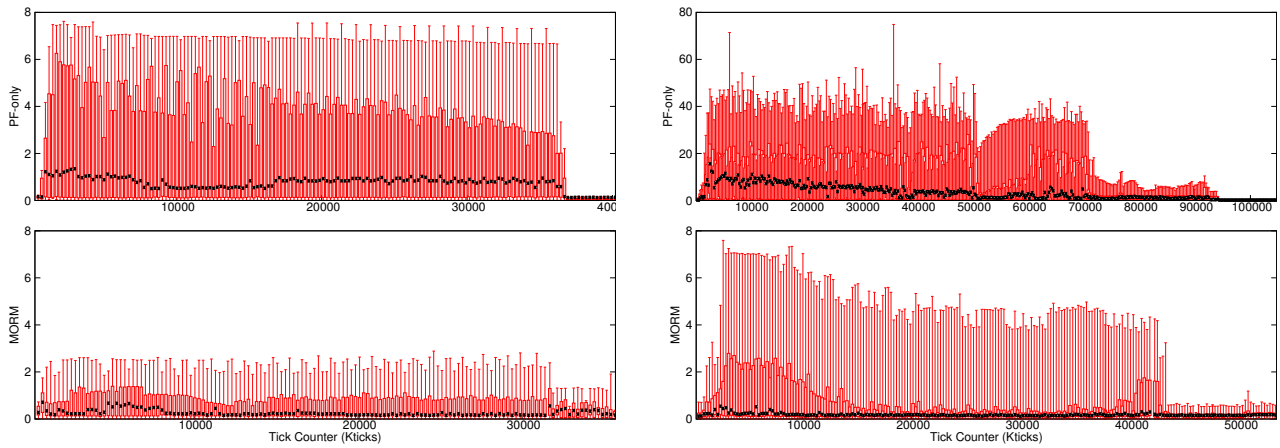
Figure 6.10 – Router Injection (%), presented in the Y-axis (note that the scale of *PF-only* with *HW*210 is not the same of the other graphs).

With *LW*, the *PF-only* router injection rate is around one fourth the *MORM* value (average mean values: 4.13% and 1.04% for *PF-only* and *MORM*, respectively). Although both injection rates may be considered low, the lower router injection rate contributes to the 11% smaller execution time in *LW* (Figure 6.8). Also, *PF-only* running with an epoch eight times larger supports this statement because the execution time reduces 16.72%.

In *HW*210, the router injection increases because more SPs are running tasks and, as a consequence, more control and data messages traverse the network. Besides that, the *PF-only* mapping algorithm may not find a contiguous area to allocate the application (as illustrated in Figure 6.6(c)) so that induces a larger number of hops to transmit the messages. Therefore, the *PF-only* router injection in *HW*210 has a distribution with many samples reaching 100% (buffer full). On the other hand, *MORM* router injection rate increases slightly in *HW*210 compared to *LW*, with similar values (in the third quartile). MORM injection rate in *LW* and *HW*210 demonstrates the scalability of the hierarchical organization.

Similarly to the router injection rate, the router congestion rate is evaluated. The router congestion rate corresponds to the input buffer utilization in the non-local ports (north, south, west, and east) higher than 75%. Figure 6.11 illustrates box plots related to the router congestion rate.

In the *LW* scenario, the router congestion rate is low because the average router injection rate stays below 7,9% for both RMs. Note that, for the worst-case evaluation, *PF-only* reaches 7.62% and *MORM* 2.88%. In the *HW*210 scenario, as expected, the router



(a) Low Workload (b) High Workload - 210 mW

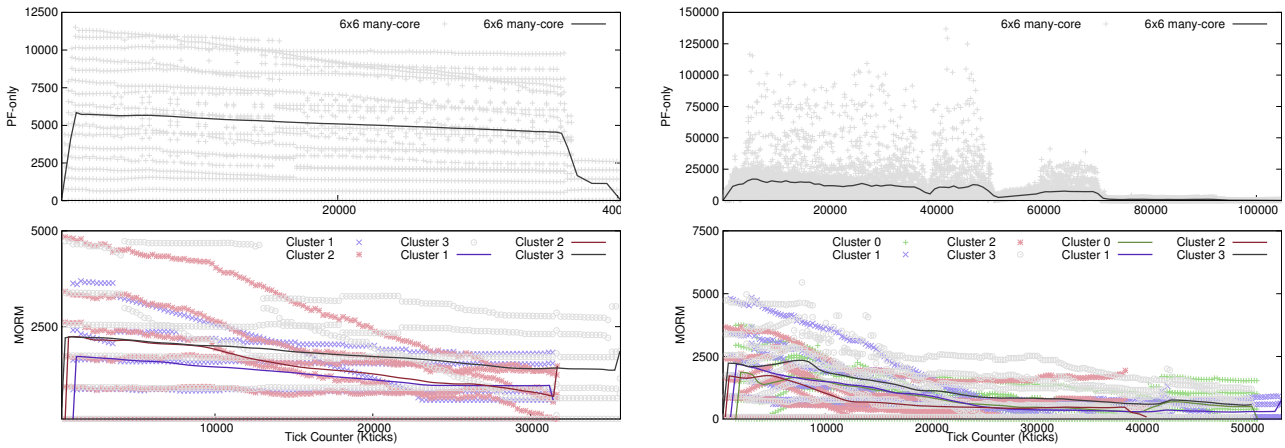
Figure 6.11 – Router Congestion (%), presented in the Y-axis (note that the scale of *PF-only* with *HW*210 is not the same of the other graphs).

congestion rate increase. In *PF-only*, the congestion increases due to traffic bottleneck around the single manager PE. Again, *MORM* management can distribute de traffic in the NoC while keeping the router congestion rate bellow 0.83% and 0.17% (average and mean values respectively).

A consequence of the network congestion is the arrival delay of control messages. This delay, named *Dif Time*, is the time, in clock cycles, for transmitting a message from a source PE to a target PE (Section 3.3.4). Figure 6.12 compares the *Dif Time* of observing messages for *MORM* and *PF-only*. The points in the graphs correspond to *Dif Time* of each observing sample sent along the simulation while the line is an average of the *Dif Time* points concerning the manager PE. Note that all control messages go to the centralized manager in *PF-only* while the control messages target the clusters managers in *MORM*, as illustrated in Figure 6.9.

MORM keeps the *Dif Time* approximately constant with the workload variation (below 2,500 Kticks). The average value for the *PF-only* management is 5,000 and 18,000 Kticks for *LW* and *HW*210 scenarios, respectively. In *HW*210 scenario, the worst-case can reach a delay of 125,000 Kticks (half of the epoch). The reason to explain this delay is the router congestion previously presented.

This Subsection showed why *PF-only* cannot achieve better results than *MORM* although presenting a better power utilization. The centralized management in *PF-only* explains the effects concerning the network traffic and the delay in delivering the control messages. As discussed in Chapter 1, hierarchical and distributed management are mandatory for NoC-based many-core systems [SDMI17], and one of the reasons is to avoid the bottlenecks around the manager PE [CMMM13]. In particular, new NoC topologies have been proposed to create the cluster organization in hardware [YLJ+16] instead software, as in this proposal.

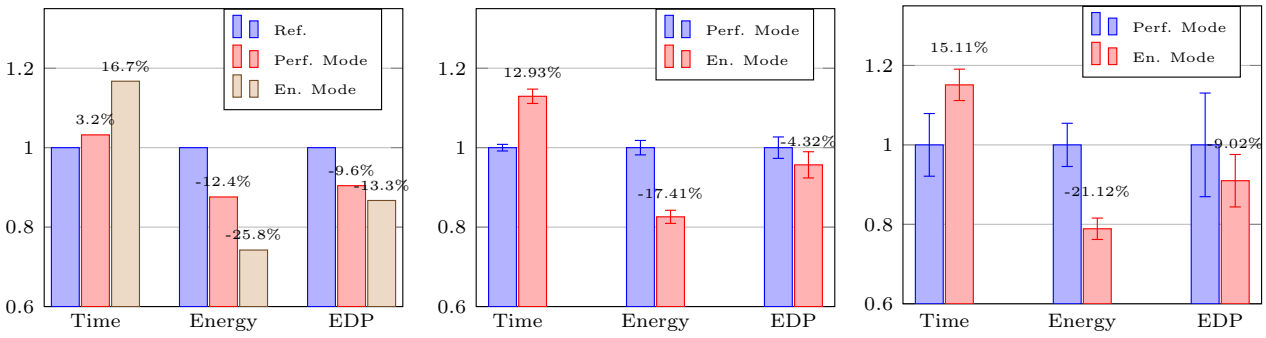


(a) Low Workload (b) High Workload - 210 mW

Figure 6.12 – *Dif Time* shows the arrival delay of the observing messages coming from SPs to manager PEs.

6.6.3 Cluster Level Results

This set of experiments evaluates the effect of the cluster size on the execution time, energy, and EDP for 3x3 and 4x4 clusters. For each cluster size, it is evaluated scenarios with single-task and multi-task mappings (up to 2 tasks per PE). All applications start at the beginning of the simulation. To evaluate the *Adaptive DVFS* we consider three scenarios: (*i*) reference – DVFS turned off; (*ii*) cluster in energy mode; (*iii*) cluster in performance mode.



(a) DVFS (b) 3x3 mapping (c) 4x4 mapping

Figure 6.13 – Cluster level results. (a) results normalized w.r.t. the reference scenario, for performance and energy modes. (b) and (c) average and standard deviation results for 3x3 and 4x4 clusters, normalized regarding the performance mode.

Figure 6.13(a) summarizes the average results normalized to the reference scenario. The EDP is lower in both operation modes because the *Adaptive DVFS* can identify the tasks phase to reduce the energy consumption. In *performance mode*, there are more opportunities to set VF_{min} because the single task mapping leads to a lower CPU utilization. In *energy mode*, the EDP reduction comes from the VF_{EDP}. Comparing energy and per-

formance modes, *Adaptive DVFS* optimizes the goal according to the mode. Note that the performance mode slightly increases the execution time (average value: 3.2%), with gains in energy and EDP.

Figures 6.13(b) and 6.13(c) evaluates different mappings, considering two cluster sizes, normalized to the performance mode. For both modes, five different mappings are considered. The performance mode adopts single task mapping while energy mode adopts multi-task mapping. Results confirm that multi-task mapping is suitable for energy savings while the single task is suitable for performance. The 3x3 cluster results reveal a small standard deviation since the smaller cluster size restricts the mapping search space (Figure 6.13(b)). The 4x4 cluster results produce a larger standard deviation, but the operation mode is still correctly driven for both cases. The standard deviation for *performance mode* is larger in 4x4 clusters because traffic creates some congestion on the NoC because some mappings increase the distance between communicating tasks. The congestion effect in the energy mode is smaller because the multi-task mapping reduces the NoC traffic.

6.7 Final Remarks

This Chapter proposed a hierarchical Decision module for Resource Management by using the Observe-Decide-Act paradigm, called *Multi-Objective Resource Management - MORM*.

MORM can dynamically adapt the running applications according to peaks and valleys of workload inherent to real systems while guaranteeing the power cap. *MORM* is a comprehensive hierarchical approach that conjointly applies DVFS and mapping/remapping to achieve runtime adaptability. The cluster-level adaptability, named as Operation Mode, allows distinct areas of the many-core to execute under different goals: energy or performance. The lightweight mapping algorithms drive the workload to the operation mode while enabling fast adaptation in case of remapping.

Comparison with a *state-of-the-art* RM optimized for single objective showed MORM outperforms the performance in 11.56% in a workload valley while achieves up to 49.43% better performance in a workload peak for any power cap. Besides, MORM energy saving are, in average, 8.27%. The comparison revealed relevant features to be considered in large many-core systems: hierarchical organization, multi-task mapping, and jointly adaptability between software (remapping) and hardware (DVFS) actuations.

Future works include: (*i*) include reactive actuations to avoid power underutilization; (*ii*) propose a method for estimating the power of tasks at execution time; (*iii*) include additional operation modes to the REM heuristic to meet goals as reliability, and security.

7. CONCLUSIONS

This Thesis introduced the following fundamental problem in the Introduction: "*how to control at runtime the set of actuators of a many-core-system in a coordinated way to provide the required adaptability to enable the development of a hierarchical multi-objective resource management running dynamic workloads.*".

To address the fundamental problem, this Thesis stated the following hypothesis: "*to embed multi-objective resource management in large many-cores systems under a dynamic workload by combining, e.g., real-time constraints and energy consumption, or energy consumption and TDP. The Thesis adopts the ODA (observe, decide, act) paradigm in a hierarchical approach to ensure scalability. Observation is essential to provide an adequate measurement infrastructure by monitoring computation and communication resources. Actuation adopts mechanisms, such as DVFS and task allocation, to meet the resource management goals. Decisions adopts comprehensive algorithms to control the system actuators based on the observed data.*"

To demonstrate the Thesis hypothesis, the starting point of this work was the proposition of hierarchical Resource Management (RM). This organization allowed the distribution of the complexity for developing the RM along hierarchy levels, making the approach scalable. Furthermore, the developed RMs follows the ODA paradigm. The ODA paradigm delimits the problem in three states, as a closed control-loop: Observe, Decide, and Act. The manuscript structure followed the ODA states.

Concerning the *Observe* state, a characterization method defined the energy and power parameters for router, memory, and processor. This method uses RTL descriptions of the PE and derives energy and power values from netlist simulations. The observing of energy and power at the PE level (the lowest level in the hierarchy) has low hardware intrusiveness. Results highlighted the method accuracy and demonstrated that widely used power estimators, as McPAT, are not adequate at the RTL level. Next, the energy observing approach is applied in the many-core to provide power data at PE, cluster and system levels. Besides the power data, the *Observe* state also monitors the processor and router utilization to make the *Decide* state aware of communication and computation status, which are essential data for efficient RMs. Results showed that the hierarchical observing has a small impact on the NoC traffic, with the observing messages arriving at the target PEs with a small delay. As a conclusion of the *Observe* state, two original contributions are claimed: (*i*) the characterization method; (*ii*) hierarchical observing approach.

Regarding the *Act* state, the RM uses several actuation methods: DVFS, power gating, clock gating, application allocation, task allocation, and task migration. The actuators are classified according to the resources impact and latency for assigning the actuator to a hierarchy level. The highest level of the hierarchy is in charge of managing the actuator with

the highest impact on the performance and the lowest level is responsible for controlling the actuators with the lowest impact. Due to the power cap inherent in many-core systems, the *Observe* state evidences the power impact of each actuator. In the context of the resources and power management, the challenge is to synchronize the actuators to follow a particular goal while avoiding actuation overlapping. A large number of actuation methods increases the management complexity but also enables the RM to achieve better results than RMs with a restricted set of actuation methods [ZH16]. A *state-of-the-art* comparison shows that no related works provide or detail the same comprehensiveness of actuation methods. In particular, the DVFS actuator considers low-level features such as latency and DC-DC converter overheads in such a way to design a realistic and accurate RM. To conclude the Act state, an original contribution is claimed: the comprehensive actuation set enables RMs to implement heuristics for multi-objective purposes.

The *Decision* state closes the ODA loop. The *Decision* state is in charge of setting the *Act* state for meeting a given goal based on the data from the *Observe* state. The *Decision* state follows the hierarchical structure assumed by other states. *Decisiion* state embeds algorithm and heuristics to manage the system This Thesis presented two proposals of multi-objective RMs, sharing the same Observe and Act states, with distinct Decision states. The first one is the Runtime Energy Management (REM) for real-time and best-effort applications. The second proposal is the Multi-Objective Resource Management (MORM) for trading energy and performance goals according to the workload behavior. Both RMs proposals present the following features to manage dynamic workloads: hierarchical organization, multi-objective decisions, and consideration of communication and computation loads.

REM is an energy-efficient RM for real-time and best-effort applications. REM employs distinct strategies for each type of application, which priority are the real-time ones. The evaluation of the *state-of-the-art* showed that RMs for real-time applications neither consider dynamic workloads nor require a design-time evaluation of the application set. REM stands out from related works when assuming applications that can enter and leave the system at any time (dynamic workload) without previous information to guide the heuristic, i.e., design-time profiling. Results showed that REM saves 18% of energy on average with negligible timing violations compared to an RM without DVFS. Results also showed REM could save 25% of energy for real-time applications compared to 15.8% of related work assuming a similar baseline platform for both proposals.

Regarding MORM, the management of operation modes at the cluster level stands out from related works since clusters can dynamically prioritize a given objective, as long as the power cap is met. MORM jointly coordinates DVFS heuristics, mapping, and remapping to carry-out at runtime cluster adaptability. At the system level, if power and resources are available, the proactive MORM heuristics admit an application, and choose the adequate cluster and operation mode. Considering a typical dynamic workload of many-core sys-

tems, MORM can run applications 11% faster in a workload peak and up to 49% faster in a workload valley compared to a single-objective approach optimized for performance.

The above paragraphs corroborate the assumptions made in the Thesis hypothesis. REM and MORM are multi-objective resource managers to trade-off conflicting objectives in a scalable way, with support to dynamic workloads. Key enablers for reaching these resource managers are the hierarchical organization, the observing infrastructure and the rich set of actuators.

7.1 Future Works

As a guideline for future works, this Thesis has room for improvements as follows:

- Observe: a validation of the characterization method in a real chip, and include DMNI in the characterization flow;
- Act: development of a realistic power gating model;
- Decide, REM: the heuristics assume a per-core capping instead of a system level one. One real-time task per PE limits the resource utilization;
- Decide, MORM: the average power at system level presents exceeding slack (difference between power cap and the power consumption)

Future works to continue the research in resource management and to fullfill the improvements previously mentioned are as follows:

- Apply the characterization method for other technologies where the leakage has a minor impact, such as SOI technologies;
- Perform validation of the characterization method in a real chip;
- Join the scheduling of real-time applications from REM and the operation mode abstraction to propose a more comprehensive and sophisticated Resource Management with at least three objectives such as QoS per application, energy per cluster, performance per cluster, system performance;
- Improve the power utilization by adding reactive actuations;
- Integrate machine learning to perform runtime RM considering unpredictable events such as interruption, network traffic, and task phases;
- Develop RM using the application layer, i.e., insert system calls in the application code to carry out part of the management;

- Take advantage of the operation mode abstraction to include other upcoming desirable features to a many-core such as security, fault tolerance, among others.

REFERENCES

[AMP+15] Arora, M.; Manne, S.; Paul, I.; Jayasena, N.; Tullsen, D. M. “Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on cpu-gpu integrated systems”. In: HPCA, 2015, pp. 366–377.

[ARM13] ARM. “big.LITTLE Technology: The Future of Mobile”, Technical Report, ARM Limited, 2013, 12p.

[BEG+15] Bringmann, O.; Ecker, W.; Gerstlauer, A.; Goyal, A.; Mueller-Gritschneder, D.; Sasidharan, P.; Singh, S. “The next generation of virtual prototyping: Ultra-fast yet accurate simulation of HW/SW systems”. In: DATE, 2015, pp. 1698–1707.

[BGD+04] Bansal, N.; Gupta, S.; Dutt, N.; Nicolau, A.; Gupta, R. “Network Topology Exploration of Mesh-based Coarse-grain Reconfigurable Architectures”. In: DATE, 2004, pp. 474–479.

[BMJ13] Bogdan, P.; Marculescu, R.; Jain, S. “Dynamic power management for multidomain system-on-chip platforms: an optimal control approach”, *ACM Transactions on Design Automation of Electronic Systems*, vol. 18–4, 2013, pp. 46:1–46:20.

[BSP+16] Bohnenstiehl, B.; Stillmaker, A.; Pimentel, J.; Andreas, T.; Liu, B.; Tran, A.; Adeagbo, E.; Baas, B. “A 5.8 pJ/Op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array”. In: VLSIC, 2016, pp. 1–2.

[Cas17] Castilhos, G. “Gerenciamento térmico e energético em MPSoCs”, Ph.D. Thesis, PUCRS, Porto Alegre, Brasil, 2017, 87p.

[CCK07] Choi, Y.; Chang, N.; Kim, T. “DC–DC converter-aware power management for low-power embedded systems”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26–8, 2007, pp. 1367–1381.

[CHE11] Carlson, T. E.; Heirmant, W.; Eeckhout, L. “Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation”. In: SC, 2011, pp. 1–12.

[CM15] Chen, Z.; Marculescu, D. “Distributed reinforcement learning for power limited many-core system performance optimization”. In: DATE, 2015, pp. 1521–1526.

[CMMM13] Castilhos, G.; Mandelli, M.; Madalozzo, G.; Moraes, F. “Distributed resource management in NoC-based MPSoCs with dynamic cluster sizes”. In: ISVLSI, 2013, pp. 153–158.

[CPMC17] Chatterjee, N.; Paul, S.; Mukherjee, P.; Chattopadhyay, S. “Deadline and energy aware dynamic task mapping and scheduling for Network-on-Chip based multi-core platform”, *Journal of Systems Architecture*, vol. 74, 2017, pp. 61–77.

[DAHM16] Das, A.; Al-Hashimi, B. M.; Merrett, G. V. “Adaptive and hierarchical runtime manager for energy-aware thermal management of embedded systems”, *ACM Transactions on Embedded Computing Systems*, vol. 15–2, 2016, pp. 24:1–24:25.

[DDVAPL14] De Dinechin, B. D.; Van Amstel, D.; Poulhiès, M.; Lager, G. “Time-critical computing on a single-chip massively parallel processor”. In: DATE, 2014, pp. 1–6.

[DJS15] Dutt, N.; Jantsch, A.; Sarma, S. “Self-aware cyber-physical systems-on-chip”. In: ICCAD, 2015, pp. 46–50.

[DJS16] Dutt, N.; Jantsch, A.; Sarma, S. “Toward smart embedded systems: A self-aware system-on-chip (SoC) perspective”, *ACM Transactions on Embedded Computing Systems*, vol. 15–2, 2016, pp. 22:1–22:27.

[DKV14] Das, A.; Kumar, A.; Veeravalli, B. “Temperature aware energy-reliability trade-offs for mapping of throughput-constrained applications on multimedia MPSoCs”. In: DATE, 2014, pp. 1–6.

[EBA+12] Esmaeilzadeh, H.; Blem, E.; Amant, R. S.; Sankaralingam, K.; Burger, D. “Dark silicon and the end of multicore scaling”, *IEEE Micro*, vol. 32–3, 2012, pp. 122–134.

[FDLP13] Fattah, M.; Daneshtalab, M.; Liljeberg, P.; Plosila, J. “Smart hill climbing for agile dynamic mapping in many-core systems”. In: DAC, 2013, pp. 1–6.

[FRD+12] Fattah, M.; Ramirez, M.; Daneshtalab, M.; Liljeberg, P.; Plosila, J. “CoNA: Dynamic application mapping for congestion reduction in many-core systems”. In: ICCD, 2012, pp. 364–370.

[FSWV07] Feghali, R.; Speranza, F.; Wang, D.; Vincent, A. “Video quality metric for bit rate control via joint adjustment of quantization and frame rate”, *IEEE Transactions on Broadcasting*, vol. 53–1, 2007, pp. 441–446.

[HGV+06] Huang, W.; Ghosh, S.; Velusamy, S.; Sankaranarayanan, K.; Skadron, K.; Stan, M. R. “HotSpot: A compact thermal modeling methodology for early-stage VLSI design”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14–5, 2006, pp. 501–513.

[HKR+15] Haghbayan, M.-H.; Kanduri, A.; Rahmani, A.-M.; Liljeberg, P.; Jantsch, A.; Tenhunen, H. “Mappro: Proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip”. In: NOCs, 2015, pp. 1–8.

[HRW+14] Haghbayan, M.-H.; Rahmani, A.-M.; Weldezion, A. Y.; Liljeberg, P.; Plosila, J.; Jantsch, A.; Tenhunen, H. “Dark silicon aware power management for manycore systems under dynamic workloads”. In: ICCD, 2014, pp. 509–512.

[HV14] Hanumaiah, V.; Vrudhula, S. “Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling”, *IEEE Transactions on Computers*, vol. 63–2, 2014, pp. 349–360.

[Ind14] Indrusiak, L. S. “End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration”, *Journal of systems architecture*, vol. 60–7, 2014, pp. 553–561.

[Int10] Intel. “The SCC Programmer’s Guide”, Technical Report, Intel Corporation, 2010, 41p.

[Int17] Intel. “Intel Core i9-7980XE Extreme Edition Processor”. Source: https://www.intel.com/content/www/us/en/products/processors/core/x-series/i9-7980xe.html, Nov 2017.

[JLK+14] Jung, H.; Lee, C.; Kang, S.-H.; Kim, S.; Oh, H.; Ha, S. “Dynamic behavior specification and dynamic mapping for real-time embedded systems: HOPES approach”, *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13–4s, 2014, pp. 135:1–135:26.

[JMSN05] Jiang, H.; Marek-Sadowska, M.; Nassif, S. R. “Benefits and costs of power-gating technique”. In: ICCD, 2005, pp. 559–566.

[KBW12] Kim, W.; Brooks, D.; Wei, G.-Y. “A fully-integrated 3-level DC-DC converter for nanosecond-scale DVFS”, *IEEE Journal of Solid-State Circuits*, vol. 47–1, 2012, pp. 206–219.

[KGWB08] Kim, W.; Gupta, M. S.; Wei, G.-Y.; Brooks, D. “System level analysis of fast, per-core DVFS using on-chip switching regulators”. In: HPCA, 2008, pp. 123–134.

[KP15] Kapadia, N.; Pasricha, S. “VARSHA: Variation and Reliability-aware Application Scheduling with Adaptive Parallelism in the Dark-silicon Era”. In: DATE, 2015, pp. 1060–1065.

[KPSH15] Khdr, H.; Pagani, S.; Shafique, M.; Henkel, J. “Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips”. In: DAC, 2015, pp. 1–6.

[KRS+14] Kriebel, F.; Rehman, S.; Sun, D.; Shafique, M.; Henkel, J. “Aser: Adaptive soft error resilience for reliability-heterogeneous processors in the dark silicon era”. In: DAC, 2014, pp. 1–6.

[LAS+09] Li, S.; Ahn, J. H.; Strong, R. D.; Brockman, J. B.; Tullsen, D. M.; Jouppi, N. P. “McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures”. In: MICRO, 2009, pp. 469–480.

[LCA+11] Li, S.; Chen, K.; Ahn, J. H.; Brockman, J. B.; Jouppi, N. P. “CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques”. In: ICCAD, 2011, pp. 694–701.

[Liu11] Liu, H. “A Measurement Study of Server Utilization in Public Clouds”. In: DASC, 2011, pp. 435–442.

[LJJ13] Li, X.; Jia, Z.; Ju, L. “Slack-Time-Aware Energy Efficient Scheduling for Multiprocessor SoCs”. In: HPCC_EUC, 2013, pp. 278–285.

[LLW+13] Lai, Z.; Lam, K. T.; Wang, C.-L.; Su, J.; Yan, Y.; Zhu, W. “Latency-aware dynamic voltage and frequency scaling on many-core architectures for data-intensive applications”. In: CloudCom-Asia, 2013, pp. 78–83.

[LWP14] Lee, W.; Wang, Y.; Pedram, M. “VRCon: Dynamic reconfiguration of voltage regulators in a multicore platform”. In: DATE, 2014, pp. 1–6.

[Mel17] Mellanox. “TILE-Gx72 Processor”. Source: http://www.mellanox.com/page/products_dyn?product_family=238&mtag=tile_gx72, Nov 2017.

[MEP08] Manolache, S.; Eles, P.; Peng, Z. “Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints”, *ACM Transactions on Embedded Computing Systems*, vol. 7–2, 2008, pp. 19:1–19:35.

[MKP15] Maiti, S.; Kapadia, N.; Pasricha, S. “Process variation aware dynamic power management in multicore systems with extended range voltage/frequency scaling”. In: MWSCAS, 2015, pp. 1–4.

[MOSM15] 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.

[MPV+13] Muthukaruppan, T. S.; Pricopi, M.; Venkataramani, V.; Mitra, T.; Vishin, S. “Hierarchical power management for asymmetric multi-core in dark silicon era”. In: DAC, 2013, pp. 1–9.

[MRM15] Martins, A. L.; Ruaro, M.; Moraes, F. G. “Hierarchical energy monitoring for many-core systems”. In: ICECS, 2015, pp. 657–660.

[MRSM17a] Martins, A.; Ruaro, M.; Santana, A.; Moraes, F. G. “Runtime energy management under real-time constraints in MPSoCs”. In: ISCAS, 2017, pp. 2589–2592.

[MRSM17b] Martins, A. L.; Ruaro, M.; Santana, A. C.; Moraes, F. G. “Distributed Runtime Energy Management for Many-Core Systems Running Real-Time Applications”, *Journal of Low Power Electronics*, vol. 13–3, 2017, pp. 402–418.

[MSC+14] Martins, A. L.; Silva, D. R.; Castilhos, G. M.; Monteiro, T. M.; Moraes, F. G. “A method for NoC-based MPSoC energy consumption estimation”. In: ICECS, 2014, pp. 427–430.

[MSM16] Martins, A. L.; Sant’Ana, A. C.; Moraes, F. G. “Runtime energy management for many-core systems”. In: ICECS, 2016, pp. 380–383.

[NVI11a] NVIDIA. “The Benefits of Quad Core CPUs in Mobile Devices”, Technical Report, NVIDIA Corporation, 2011, 19p.

[NVI11b] NVIDIA. “Variable SMP - A Multi-Core CPU Architecture for Low Power and High Performance”, Technical Report, NVIDIA Corporation, 2011, 16p.

[OA17] Olsen, D.; Anagnostopoulos, I. “Performance-Aware Resource Management of Multi-Threaded Applications on Many-Core Systems”. In: GLSVLSI, 2017, pp. 119–124.

[OGI+09] Ost, L.; Guindani, G.; Indrusiak, L. S.; Reinbrecht, C.; Raupp, T.; Moraes, F. “A high abstraction, high accuracy power estimation model for networks-on-chip”. In: SBCCI, 2009, pp. 31:1–31:6.

[PAC+12] Paterna, F.; Acquaviva, A.; Caprara, A.; Papariello, F.; Desoli, G.; Benini, L. “Variability-aware task allocation for energy-efficient quality of service provisioning in embedded streaming multimedia applications”, *IEEE Transactions on Computers*, vol. 61–7, 2012, pp. 939–953.

[PKM+14] Pagani, S.; Khdr, H.; Munawar, W.; Chen, J.-J.; Shafique, M.; Li, M.; Henkel, J. “TSP: thermal safe power: efficient power budgeting for many-core systems in dark silicon”. In: CODES+ISSS, 2014, pp. 1–10.

[PKS+17] Pathania, A.; Khdr, H.; Shafique, M.; Mitra, T.; Henkel, J. “Scalable probabilistic power budgeting for many-cores”. In: DATE, 2017, pp. 864–869.

[QP16] Quan, W.; Pimentel, A. D. “A hierarchical run-time adaptive resource allocation framework for large-scale MPSoC systems”, *Design Automation for Embedded Systems*, vol. 20–4, 2016, pp. 311–339.

[RHK+15] Rahmani, A.-M.; Haghbayan, M.-H.; Kanduri, A.; Weldezion, A. Y.; Liljeberg, P.; Plosila, J.; Jantsch, A.; Tenhunen, H. “Dynamic power management for manycore platforms in the dark silicon era: A multiobjective control approach”. In: ISLPED, 2015, pp. 219–224.

[RJD17] Rahmani, A. M.; Jantsch, A.; Dutt, N. “HDGM: Hierarchical Dynamic Goal Management for Many-Core Resource Allocation”, *IEEE Embedded Systems Letters*, vol. PP–99, 2017, pp. 1–4.

[RLH+16] Rahmani, A. M.; Liljeberg, P.; Hemani, A.; Jantsch, A.; Tenhunen, H. “Dark Side of Silicon”. Springer, 2016, 373p.

[RLMM16] Ruaro, M.; Lazzarotto, F. B.; Marcon, C. A.; Moraes, F. G. “DMNI: A specialized network interface for NoC-based MPSoCs”. In: ISCAS, 2016, pp. 1202–1205.

[RM16] Ruaro, M.; Moraes, F. G. “Dynamic real-time scheduler for large-scale MPSoCs”. In: GLSVLSI, 2016, pp. 341–346.

[RM17] Ruaro, M.; Moraes, F. G. “Demystifying the cost of task migration in distributed memory many-core systems”. In: ISCAS, 2017, pp. 148–151.

[RTGM13] Raghunathan, B.; Turakhia, Y.; Garg, S.; Marculescu, D. “Cherry-picking: exploiting process variations in dark-silicon homogeneous chip multi-processors”. In: DATE, 2013, pp. 39–44.

[SCW05] Sun, X.-H.; Chen, Y.; Wu, M. “Scalability of heterogeneous computing”. In: ICPP, 2005, pp. 557–564.

[SDI15] Singh, A. K.; Dziurzanski, P.; Indrusiak, L. S. “Value and energy optimizing dynamic resource allocation in many-core HPC systems”. In: CloudCom, 2015, pp. 180–185.

[SDI16] Singh, A. K.; Dziurzanski, P.; Indrusiak, L. S. “Value and energy aware adaptive resource allocation of soft real-time jobs on many-core HPC data centers”. In: ISORC, 2016, pp. 190–197.

[SDK13] Singh, A. K.; Das, A.; Kumar, A. “Energy optimization by exploiting execution slacks in streaming applications on multiprocessor systems”. In: DAC, 2013, pp. 1–7.

[SDMI17] Singh, A. K.; Dziurzanski, P.; Mendis, H. R.; Indrusiak, L. S. “A Survey and Comparative Study of Hard and Soft Real-Time Dynamic Resource Allocation Strategies for Multi-/Many-Core Systems”, *ACM Computing Surveys*, vol. 50–2, 2017, pp. 24:1–24:40.

[SGHM14] Shafique, M.; Garg, S.; Henkel, J.; Marculescu, D. “The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives”. In: DAC, 2014, pp. 1–6.

[SKF+14] Siegel, H. J.; Khemka, B.; Friese, R.; Pasricha, S.; Maciejewski, A. A.; Koenig, G. A.; Powers, S.; Hilton, M.; Rambharos, R.; Okonski, G.; et al.. “Energy-aware resource management for computing systems”. In: IC3, 2014, pp. 7–12.

[SLR+17] Singh, A. K.; Leech, C.; Reddy, B. K.; Al-Hashimi, B. M.; Merrett, G. V. “Learning-based Run-time Power and Energy Management of Multi/Many-core Systems: Current and Future Trends”, *Journal of Low Power Electronics*, vol. 13–3, 2017, pp. 310–325.

[SSKH13] Singh, A. K.; Shafique, M.; Kumar, A.; Henkel, J. “Mapping on multi/many-core systems: survey of current and emerging trends”. In: DAC, 2013, pp. 1–10.

[TMW94] Tiwari, V.; Malik, S.; Wolfe, A. “Power analysis of embedded software: a first step towards software power minimization”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2–4, 1994, pp. 437–445.

[WDH+16] Walker, M. J.; Diestelhorst, S.; Hansson, A.; Balsamo, D.; Merrett, G. V.; Al-Hashimi, B. M. “Thermally-aware composite run-time CPU power models”. In: PATMOS, 2016, pp. 17–24.

[WDH+17] Walker, M. J.; Diestelhorst, S.; Hansson, A.; Das, A. K.; Yang, S.; Al-Hashimi, B. M.; Merrett, G. V. “Accurate and stable run-time power modeling for mobile and embedded CPUs”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36–1, 2017.

[WLW+17] Wei, X.; Liang, Y.; Wang, T.; Lu, S.; Cong, J. “Throughput optimization for streaming applications on CPU-FPGA heterogeneous systems”. In: ASP-DAC, 2017, pp. 488–493.

[WPW00] Wu, Q.; Pedram, M.; Wu, X. “Clock-gating and its application to low power design of sequential circuits”, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47–3, 2000, pp. 415–420.

[WS12] Wang, L.; Skadron, K. “Dark vs. dim silicon and near-threshold computing extended results”, Technical Report, University of Virginia Department of Computer Science TR-2013, 2012, 29p.

[XJB+15] Xi, S. L.; Jacobson, H.; Bose, P.; Wei, G.-Y.; Brooks, D. “Quantifying sources of error in McPAT and potential impacts on architectural studies”. In: HPCA, 2015, pp. 577–589.

[YLJ+16] Yang, L.; Liu, W.; Jiang, W.; Li, M.; Yi, J.; Sha, E. H.-M. “Fotonoc: A hierarchical management strategy based on folded lorus-like network-on-chip for dark silicon many-core systems”. In: ASP-DAC, 2016, pp. 725–730.

[YSH14] Yu, H.; Syed, R.; Ha, Y. “Thermal-aware frequency scaling for adaptive workloads on heterogeneous MPSoCs”. In: DATE, 2014, pp. 1–6.

[YSM+15] Yang, S.; Shafik, R. A.; Merrett, G. V.; Stott, E.; Levine, J. M.; Davis, J.; Al-Hashimi, B. M. “Adaptive energy minimization of embedded heterogeneous systems using regression-based learning”. In: PATMOS, 2015, pp. 103–110.

[ZH16] Zhang, H.; Hoffmann, H. “Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniquess”, *ACM SIGPLAN Notices*, vol. 51–4, 2016, pp. 545–559.

[ZJG16] Zheng, X.; John, L. K.; Gerstlauer, A. “Accurate phase-level cross-platform power and performance estimation”. In: DATE, 2016, pp. 1–6.

APPENDIX A – GRAPHS

A.1 MORM Average Power Results at the Cluster Level

Figure 6.7 shows average power results for all scenarios in Subsection 6.6.2. In MORM results a lack of samplings is observed in *HW150* and *HW180* scenarios at the system level because the GM has an application waiting for admission and cannot process the messages from CM to generate a system level sample. To check the power cap is attended for all scenarios, as it was mentioned in Subsection 6.6.2, Figure A.1 exhibits the MORM average power results at the cluster level. Note that, the sum of the power curves for all clusters does not overcome the power cap at any moment in Figure A.1.

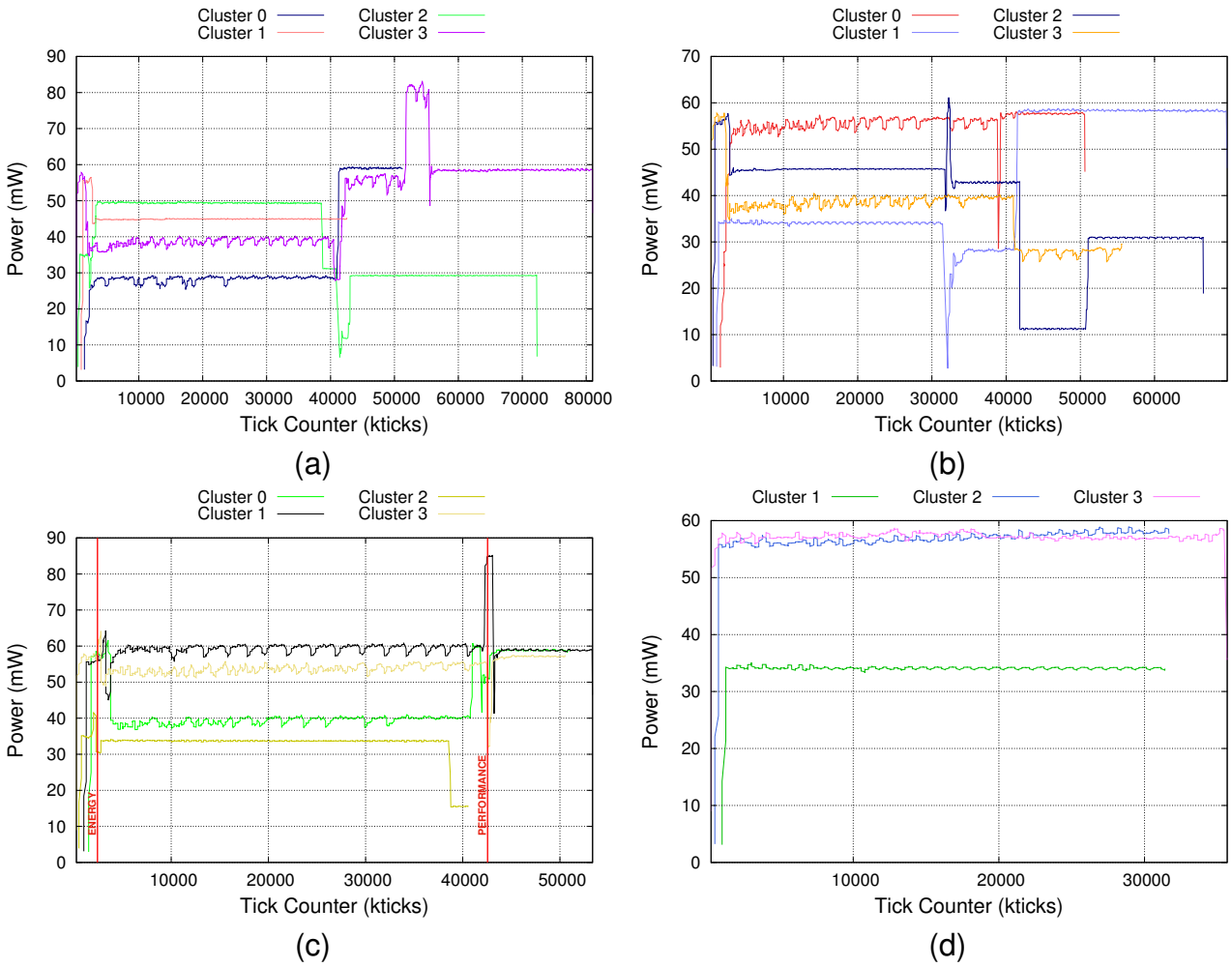


Figure A.1 – MORM power results per cluster running (a) HW150, (b) HW180, (c) HW210, (d) LW scenarios. Note that in LW scenario Cluster 0 has no applications because only three applications are running

A.2 PF-only Average Power Results at the System Level

PF-only exhibits more disturbances in the power profiling compared to MORM as well as some power overshoots (Average power results in Figure 6.7). The noise in the power curve has three main reasons: (i) the centralized manager creates bottlenecks and the observing message gets delay in the receiving; (ii) proactive and reactive actuations in a PID control triggers more actuation compared to MORM and achieve a better utilization of the power cap; and (iii) DVFS at application level employed in PF-only generates more impact in power curve than the DVFS at PE level used in MORM. Figure A.2 shows the power profiling for all scenarios and highlights all actuation moments triggered by PF-only with vertical lines. The blue vertical lines are actuations to reduce the vf-pair for slowing an application down and the red one corresponds to an increase of the vf-pair to speed an application up.

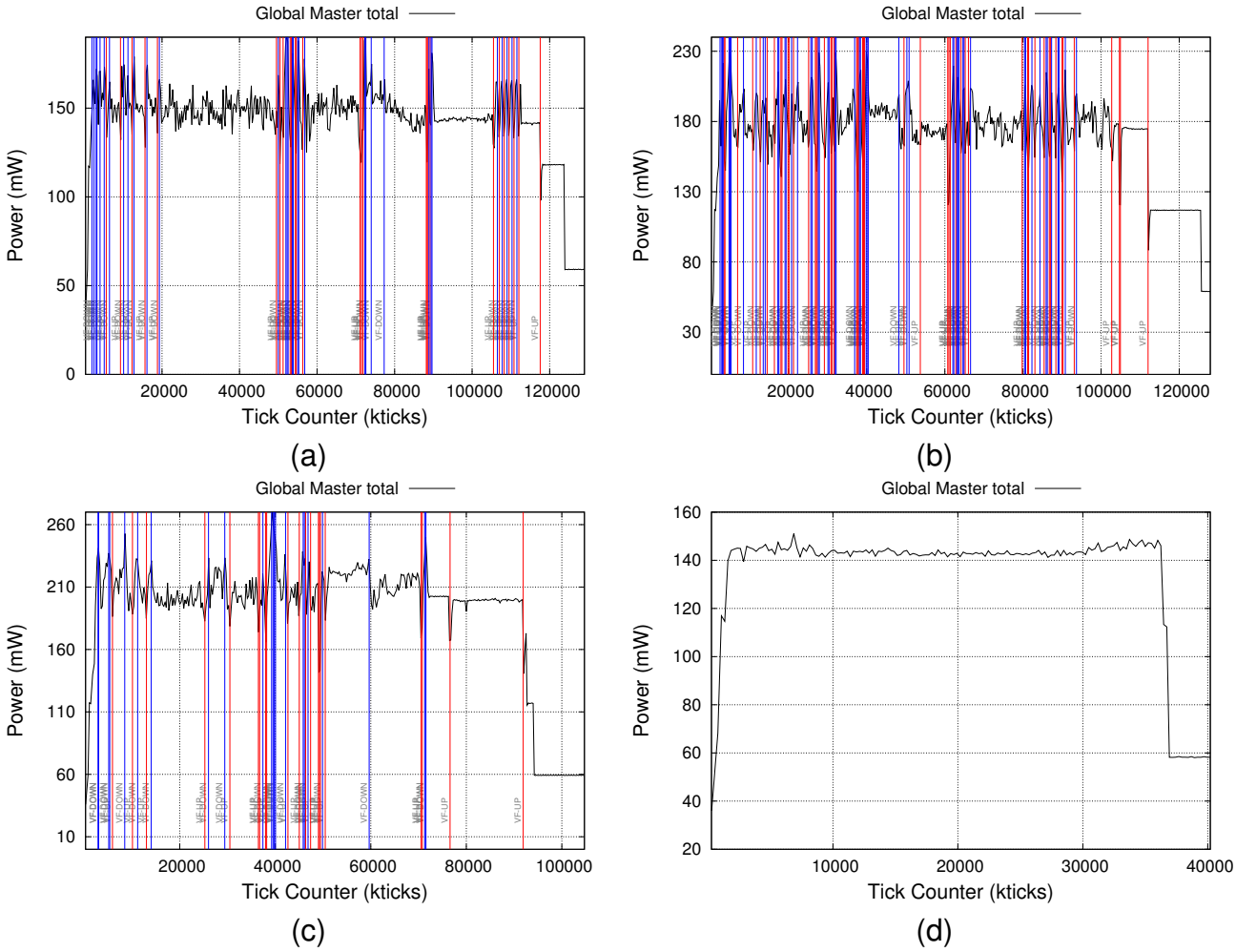


Figure A.2 – PF-only power results running (a) HW150, (b) HW180, (c) HW210, (d) LW scenarios. Vertical lines highlight when the centralized controller triggers actuations.

A.3 Router Congestion

Subsection 6.6.2 shows router congestion results only to *HW210* and *LW* scenarios. Figure A.3 shows router congestion results for all scenarios.

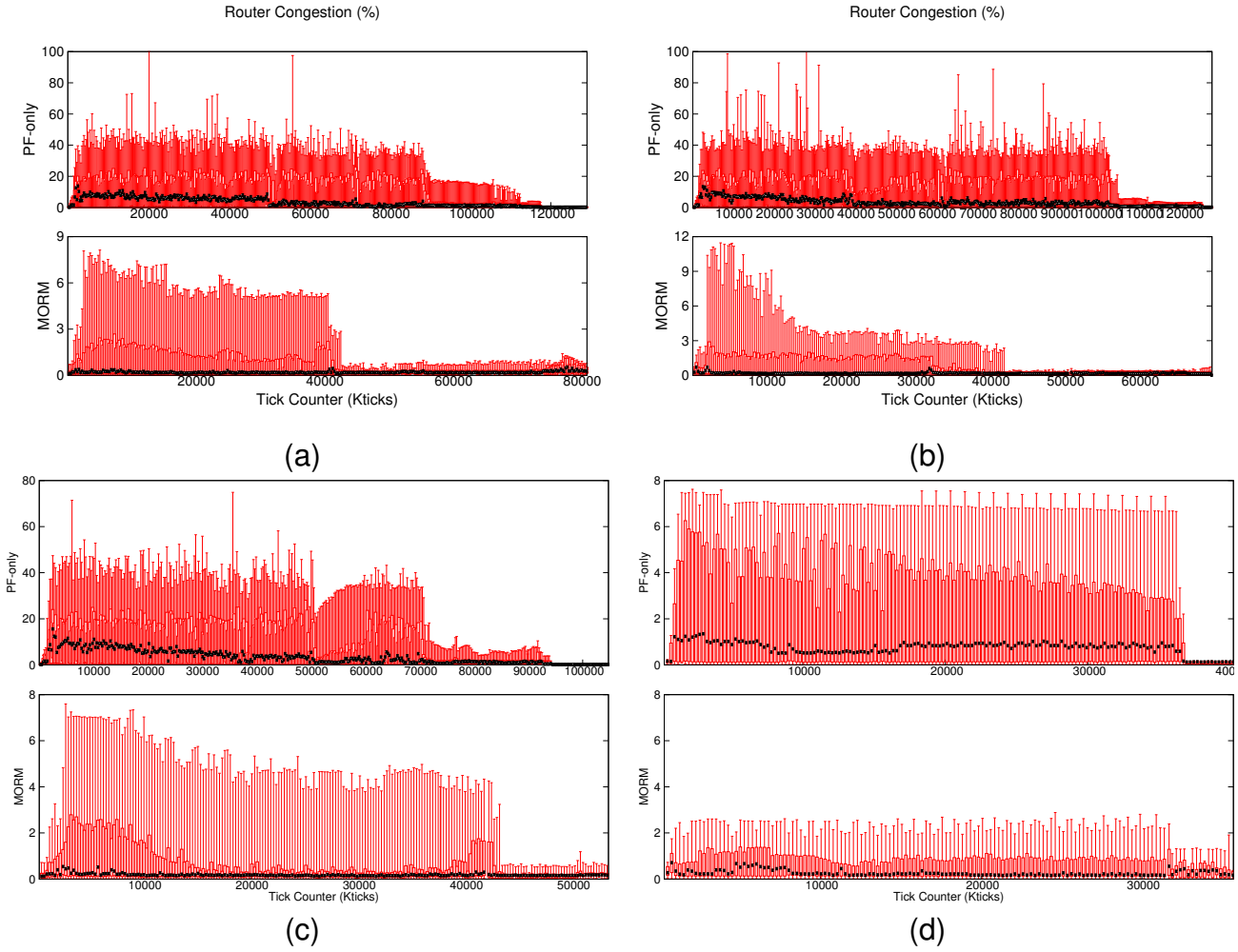


Figure A.3 – Router Congestion (%) results running (a) HW150, (b) HW180, (c) HW210, (d) LW scenarios.

A.4 Router Injection

Subsection 6.6.2 shows router injection results only to *HW210* and *LW* scenarios. Figures A.4 show router injection results for all scenarios.

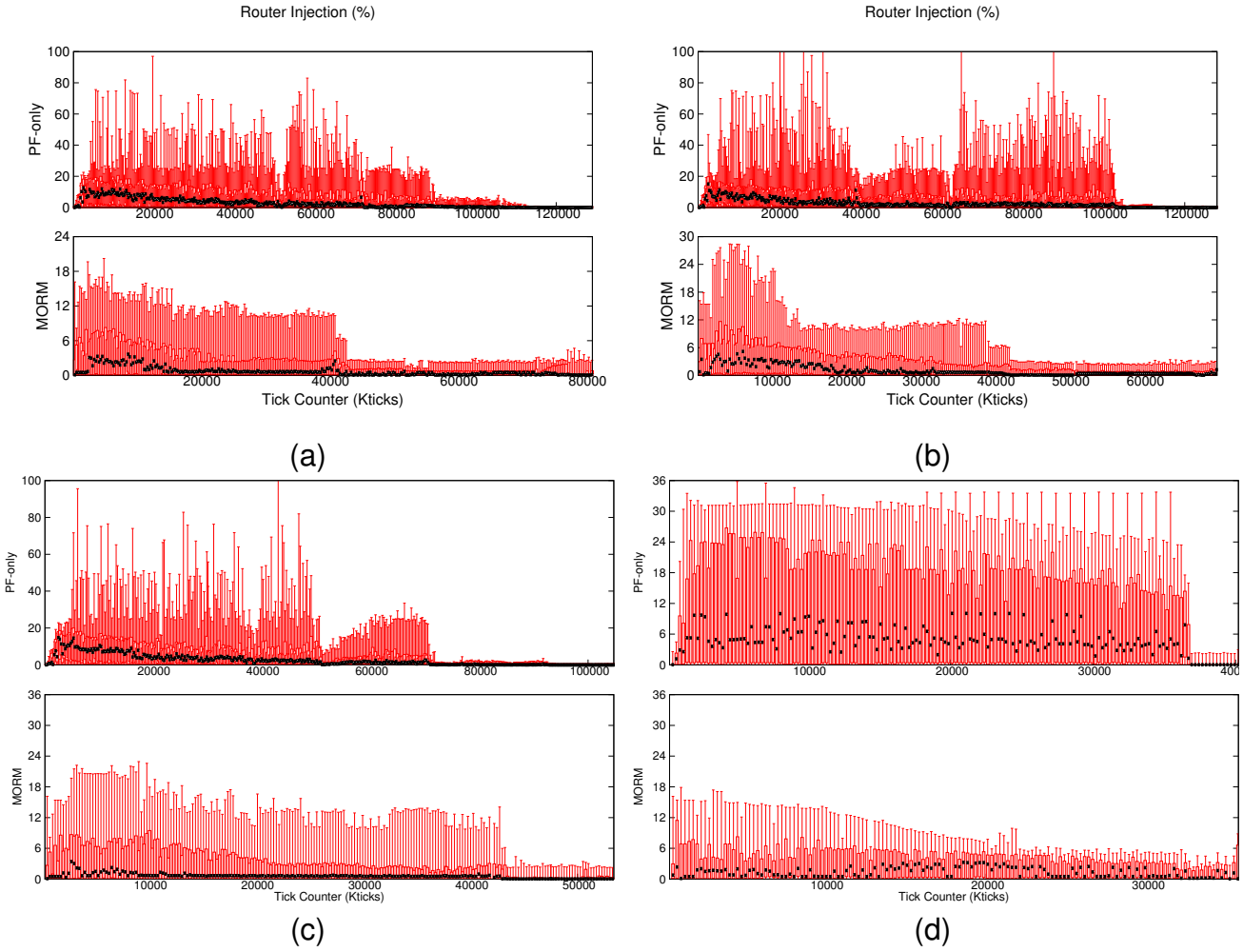


Figure A.4 – Router Injection (%) results running (a) HW150, (b) HW180, (c) HW210, (d) LW scenarios.

A.5 Dif Time

Subsection 6.6.2 shows *Dif Time* results only to *HW210* and *LW* scenarios. Figure A.5 shows *Dif Time* results for all scenarios.

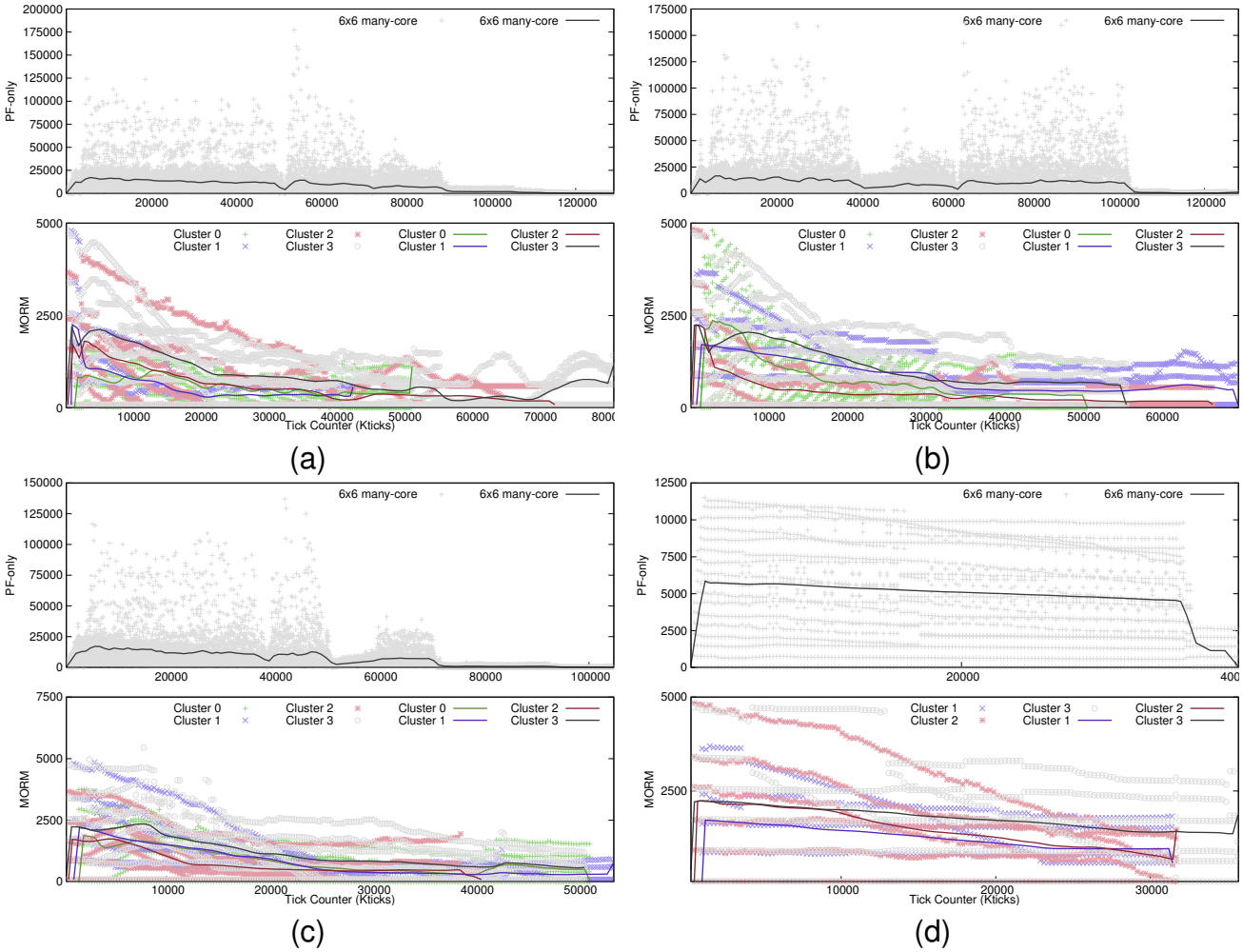


Figure A.5 – *Dif Time* results running (a) HW150, (b) HW180, (c) HW210, (d) LW scenarios.

Pontifícia Universidade Católica do Rio Grande do Sul
Pró-Reitoria de Graduação
Av. Ipiranga, 6681 - Prédio 1 - 3º. andar
Porto Alegre - RS - Brasil
Fone: (51) 3320-3500 - Fax: (51) 3339-1564
E-mail: prograd@pucrs.br
Site: www.pucrs.br