Run-Time Management for Energy Efficiency of Cluster-based Multi/Many-Core Systems

Simei Yang

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

THESE DE DOCTORAT DE

L'UNIVERSITE DE NANTES
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Sciences de l'Information et de la Communication

Par
Simei YANG

Run-Time Management for Energy Efficiency of Cluster-based Multi/Many-Core Systems

Thèse présentée et soutenue à Nantes, le 11 juin 2020
Unité de recherche : IETR UMR CNRS 6164

Rapporteurs avant soutenance :

Andy D. PIMENTEL — Associate Professor, University of Amsterdam
Frédéric ROUSSEAU — Professeur, Université Grenoble-Alpes

Composition du Jury :

Présidente :	Nathalie JULIEN	Professeur, Université Bretagne Sud
Examinateurs :	Andy D. PIMENTEL	Associate Professor, University of Amsterdam
	Frédéric ROUSSEAU	Professeur, Université Grenoble-Alpes
Directeur de thèse :	Sébastien PILLEMENT	Professeur, Université de Nantes
Invités :	Sébastien LE NOURS	Maître de Conférences, Université de Nantes
	Maria MENDEZ REAL	Maître de Conférences, Université de Nantes

Résumé

Les plates-formes multi-cœur organisées en clusters représentent des solutions prometteuses pour offrir un compromis optimisé vis-à-vis des critères de performance et de consommation dans les systèmes embarqués modernes. Sur une telle plate-forme, les cœurs sont divisés en différents clusters et chaque cluster fonctionne à un niveau de tension ou de fréquence (Voltage/Frequency, v/f) donné. Ces plates-formes permettent de mise à l'échelle dynamique de la tension et de la fréquence (Dynamic Voltage/Frequency Scaling (DVFS)) pour chaque cluster, ce qui permet à différents clusters de changer leurs propres niveaux v/f indépendamment. Comme le nombre de cœurs continue d'augmenter, de plus en plus d'applications peuvent être prises en charge sur de telles plates-formes. Impliquant de ce fait une variation des charges de travail supportées en cours de fonctionnement. La complexité croissante des applications supportées et la variabilité des charges de calcul en cours de fonctionnement justifient le besoin d'une gestion en ligne des ressources des plates-formes afin de garantir les exigences de performance du système. **Ce travail de thèse se concentre sur la gestion en cours de fonctionnement des applications exécutées dans des systèmes multi-cœurs organisés en clusters afin d'améliorer l'efficacité énergétique compte tenu des contraintes de performances des applications et des contraintes de ressources de la plate-forme.**

Dans ce contexte, deux principaux problèmes de recherche sont étudiés.

- **Le premier problème de recherche concerne la façon de coordonner l'allocation dynamique des tâches et la mise à l'échelle des niveaux de fréquence/tension.** L'allocation dynamique des tâches et les techniques de DVFS ont été largement appliquées pour optimiser l'efficacité énergétique dans les systèmes multi-cœurs. Cependant, la façon de coordonner ces deux techniques pour gérer plusieurs applications exécutées dans des plates-formes multi-coeurs organisées en clusters reste une question ouverte. L'allocation d'applications multi-tâches implique différentes configurations possibles des tensions et fréquences de fonctionnement au sein d'un seul cluster et à l'échelle de la plate-forme complète. Dans cette thèse, nous proposons différentes stratégies de gestion estimant l'influence mutuelle entre l'allocation des applications et les tensions/fréquences des clusters afin d'optimiser l'énergie consommée localement, à l'échelle d'un cluster, et globalement, à l'échelle de la plate-forme.

- **La seconde problématique de recherche abordée traite de la simulation au niveau système des stratégies de gestion dynamique des plates-formes multi-coeurs.** La simulation au niveau système est utilisée afin d'estimer au plus tôt dans le processus de conception des performances des systèmes étudiés. Cependant, la plupart des environnements de simulation existant ne permettent pas de décrire la gestion dynamique des ressources et notamment l'allocation dynamique des tâches. Dans le cadre de notre travail, nous proposons une nouvelle approche de modélisation et de simulation permettant la prise en compte des stratégies de gestion au sein de tels environnement.

Pour résoudre le premier problème de recherche *i.e.*, afin d'optimiser conjointement l'allocation dynamique des tâches et la sélection des fréquences de fonctionnement, nous adoptons une stratégie de gestion dite hybride. Ces stratégies reposent sur un ensemble d'éléments préparés hors ligne, au cours de la phase de conception. Dans les travaux existants, ces éléments correspondent typiquement à des allocations préparées statiquement pour chaque application supportée. Ces allocations seront ensuite utilisées en cours de fonctionnement afin d'établir une allocation optimisée pour l'ensemble des applications actives. Dans le cadre de notre travail, nous introduisons un nouveau paramètre préparé hors ligne appelé 'Fréquence Minimale Autorisée (MAF, Minimal Allowed Frequency). Ce paramètre définit la fréquence minimale de fonctionnement d'un cluster pour une allocation des tâches d'une application donnée et permettant le respect des contraintes de temps associées. En se basant sur un ensemble d'allocations préparées hors ligne et sur l'estimation du paramètre MAF pour chaque allocation, nous proposons différentes stratégies d'optimisation de l'énergie consommée, à l'échelle d'un seul cluster et à l'échelle d'une plate-forme composée de plusieurs clusters.

Nous étudions tout d'abord l'optimisation de l'énergie consommée à l'échelle d'un seul cluster formé de plusieurs coeurs de processeurs homogènes. L'objectif porte sur la minimisation de la fréquence de fonctionnement requise et ce afin de réduire l'énergie consommée. Les principales contributions sont les suivantes.

- Tout d'abord, une nouvelle stratégie est proposée pour sélectionner pour chaque application active au sein de chaque situation de fonctionnement une allocation appropriée. Cette stratégie repose sur des données préparées au moment de la conception. Ces données correspondent à plusieurs allocations possibles de chaque application ainsi que le critère de MAF déterminé pour chaque allocation possible. La stratégie de sélection proposée considère tout d'abord la minimisation de la fréquence requise du cluster. Elle détermine ensuite les allocations appropriées pour chaque application active selon la situation de fonctionnement considérée. En utilisant les paramètres MAFs estimés, notre stratégie de gestion présente une complexité limitée afin d'explorer les configurations possibles.

- Deuxièmement, une nouvelle stratégie d'allocation des applications est proposée (Grouped Applications Packing under Varied Constraints (GAPVC)). Cette stratégie vise à limiter le nombre de ressources de calcul utilisées en optimisant l'utilisation de chaque ressource. Par rapport à la stratégie simple de combinaison (First-Come-First-Served (FCFS)) qui alloue les tâches d'une même application sur une même ressource de calcul, notre stratégie GAPVC proposée peut réduire le nombre de ressources utilisées sans dégrader les performances des applications. La stratégie de sélection et la stratégie de combinaison des allocations sont appliquées itérativement pour parvenir à une solution quasi optimale.

- Troisièmement, plusieurs cas d'utilisation, comprenant jusqu'à neuf applications actives simultanément, ont été examinés pour évaluer les avantages de l'approche de gestion proposée en termes de puissance dynamique moyenne, de ressources d'utilisation et de

complexité. Nos expériences ont démontré que notre stratégie de gestion peut réduire la consommation moyenne d'énergie d'environ 36% et de 206% par rapport aux méthodes existantes dans la littérature.

Dans un second temps, nous étudions l'optimisation de l'efficacité énergétique au niveau global d'une plate-forme formée par plusieurs clusters de calcul. A ce niveau, l'efficacité énergétique d'une plate-forme dépend de l'allocation des applications d'un cluster à un autre. Lorsque plusieurs applications sont exécutées sur un même cluster, la fréquence de fonctionnement d'un cluster est dépendante de l'application avec la contrainte de temps la plus sévère. Dès lors, dans le cas de plates-formes hétérogènes (avec des ressources de calcul de natures différentes d'un cluster à un autre), l'allocation des applications d'un cluster à un autre peut influer significativement sur la fréquence possible de fonctionnement des clusters. Dans la littérature, l'allocation des applications entre clusters et optimisation des fréquences de fonctionnement sont généralement considérées successivement. Les solutions proposées sont généralement pour des plates-formes de complexité réduite (en nombre de clusterd et en nombre de ressources de calcul au sein des clusters). Les solutions existantes présentent dès lors des limitations lorsque la complexité des plates-formes augmente.

Dans ce travail , nous proposons une stratégie d'allocation des applications et de sélection des fréquences de fonctionnement à l'échelle d'une plate-forme formée par plusieurs clusters. La stratégie proposée peut être utilisée pour gérer plusieurs applications exécutées de manière dynamique sur des plates-formes de différentes tailles. Une structure de gestion hiérarchique à deux niveaux est adoptée, dans laquelle un premier niveau de gestion global détermine les allocations des applications et fixe les niveaux de fréquence des clusters. Le second niveau de gestion local optimise l'allocation et l'ordonnancement des tâches dans chaque cluster. Ce travail apporte les contributions suivantes.

- Tout d'abord, pour la gestion globale, nous présentons un modèle de 0-1 Integer Programming (IP) qui considère une formulation de la puissance dynamique moyenne du système compte tenu des allocations retenues des applications actives et des fréquences possibles de fonctionnement. Le paramètre MAF est utilisé afin d'estimer les fréquences possibles des fonctionnement des clusters. L'objectif du modèle 0-1 IP proposé est de trouver des allocations des applications sur les clusters de la plate-forme cible qui minimisent la consommation dynamique moyenne de l'ensemble du système. Cette minimisation est recherchée tout en tenant compte des contraintes de temps des applications, du nombre de ressources au sein de chaque cluster et des fréquences possibles de fonctionnement.

- Deuxièmement, pour parvenir à la solution du problème d'optimisation formulé, il n'est pas possible de rechercher de manière exhaustive la solution optimale dans un délai raisonnable. Afin de réduire la complexité de la recherche, nous proposons une première stratégie de gestion globale (Neighboring Search Application-to-Cluster Assignment

(NSACA)) qui vise à fournir des solutions quasi optimales. La méthode NSACA considère, pour une situation de fonctionnement donné, l'ensemble des allocations possibles pour toutes les applications actives. Cette stratégie alloue tout d'abord les applications au sein d'un cluster compte tenu du paramètre MAF. Elle améliore ensuite de manière itérative les allocations possibles en considérant des clusters voisins.

- Troisièmement, nous proposons une deuxième stratégie de gestion globale (Greedy Search Application-to-Cluster Assignment (GSACA)) qui optimise l'allocation de chaque application prise individuellement. Cette stratégie vise à établir pour chaque application l'allocation la plus économe en énergie et réduit le nombre de migration d'une situation de fonctionnement à une autre. Le nombre de migrations peut être contrôlé par les utilisateurs. Par rapport à la première stratégie de gestion, la deuxième stratégie de gestion globale peut réduire le nombre de migrations d'applications entre clusters avec un impact limité sur l'énergie consommée.

- Quatrièmement, nos deux stratégies de gestion globales proposées utilisent la stratégie FCFS de combinaison des allocations pour estimer le nombre de ressources de calcul utilisées dans chaque cluster. L'objectif de la stratégie de gestion locale est d'établir l'utilisation des ressources de calcul au sein d'un cluster. Comme la stratégie de combinaison précédemment proposée GAPVC peut réduire le nombre de ressources utilisées dans chaque cluster sans dégrader les performances de l'application, nous considérons également les avantages de l'utilisation de GAPVC au niveau de la gestion locale. L'utilisation de GAPVC permet d'optimiser l'occupation des ressources au sein des différents clusters. En conséquence, une consommation d'énergie moyenne plus faible peut être réalisée dans l'ensemble du système.

- Dans nos expériences, nous avons évalué nos stratégies de gestion proposées pour différents ensembles d'applications actives (jusqu'à 10 applications) exécutées sur différentes tailles de plates-formes (par exemple : jusqu'à 24 noyaux dans un cluster, jusqu'à 8 clusters dans le système). Les résultats expérimentaux ont indiqué que : (1) la consommation moyenne d'énergie réalisée par NSACA n'est que de 1.93% inférieure à la solution optimale (c'est-à-dire par recherche Exhaustive), mais la vitesse de NSACA est 2674 fois plus rapide. (2) sans tenir compte du coût de la migration, un plus grand nombre de migrations dans la GSACA peut conduire à une réduction de la consommation moyenne d'énergie de l'ensemble du système. (3) sur la base des mêmes stratégies de gestion globale, GAPVC peut réduire la consommation moyenne d'énergie du système jusqu'à 57.65% par rapport à FCFS.

Nous nous intéressons également à la problématique de l'évaluation des méthodes de gestion dynamique de plates-formes. Dans le contexte de la modélisation et de la simulation au niveau système, nous apportons les contributions suivantes:

- Nous avons proposé une nouvelle approche de modélisation et de simulation de niveau du système qui permet l'évaluation des stratégies de gestion dynamique de plates-formes multi-core. Afin de favoriser la simulation de ces stratégies, l'approche proposée calcule dynamiquement les instants où les ressources de la plate-forme sont utilisées par les applications en cours d'exécution. Basé sur les instants de simulation calculés, un modèle de gestionnaire d'exécution est introduit pour contrôler à la fois l'ordre d'exécution des tâches et l'avancement du temps de simulation. Cette approche de simulation peut être utilisée afin de simuler différents nombres d'applications exécutées sur des plates-formes hétérogènes dans des configurations de v/f variées. Contrairement à l'approche de simulation basée sur la trace, l'approche de simulation proposée réduit le nombre d'événements nécessaires et le nombre d'appels au moteur de simulation.

- En outre, nous mettons en œuvre et validons l'approche proposée à l'aide du cadre de modélisation d'Intel Cofluent Studio. Grâce à une étude de cas qui tient compte de sept applications (85 tâches au total) fonctionnant sur une plate-forme hétérogène basée sur des clusters, l'approche proposée permet d'évaluer différentes stratégies de gestion en fonction de la latence et du critère de consommation d'énergie. On a observé que l'influence de l'approche proposée sur l'effort de simulation est raisonnable. Par rapport au cadre de Cofluent par défaut (pour 85 tâches en cours d'exécution), la charge de travail de simulation a augmenté de moins de 10.8%.

En résumé, ce travail de thèse étudie deux problèmes de recherche du point de vue des stratégies de gestion en ligne et de l'évaluation des performances. À l'avenir, nous pourrions apporter certaines améliorations possibles à ce travail. Tout d'abord, nous pouvons envisager la mise en œuvre réelle de nos deux stratégies de gestion proposées dans de véritables systèmes multi/multi-cœurs basés sur des clusters, tels que la plate-forme ARM big.LITTLE (Odroid XU3) et Kalray MPPA. Deuxièmement, nous pouvons envisager d'autres critères d'optimisation (*e.g.*, fiabilité thermique, sécurité) dans les stratégies de gestion. Troisièmement, il est possible d'étendre l'approche de simulation que nous proposons (pour l'évaluation des stratégies de gestion) à d'autres cadres de simulation de niveau système.

Acknowledgments

During the last three years of my PhD life, I am lucky to have received a lot of help from many nice people. Here, I would like to express my gratitude to all of them.

First of all, I would like to express my gratitude to my supervisor and co-supervisor: Sébastien Pillement and Sébastien Le Nours. I thank them for accepting me as one of their students and for giving me the opportunity to engage in this interesting research topic. In these recent years, I have benefited greatly from their in-depth discussions and critical remarks.

Secondly, my heartfelt gratitude goes to my dear Maria Mendez Real, who had also advised me during the last year of my PhD work. She was always willing to share her vast knowledge with me and to motivate me in my work. I was so lucky to meet Maria and have the opportunity to work with her. I think I will never forget the days when we worked in the same office. Besides, I sincerely thank Prof. Tao Su from Sun Yat-sen University in China. In recent years, when I was frustrated with my work, I sent him some emails. He can always look at my difficulties from a broader perspective and give me the strength to move forward.

Thirdly, I would like to express my appreciation to the reviewers and defense committee, particularly, Prof. Andy D. Pimentel, Prof. Frédéric Rousseau, Prof. Nathalie Julien, for their efforts and time invested in reviewing my dissertation and attending my defense.

Moreover, I would like to thank Sandrine Charlier, Marc Brunet, and Guillaume Lirzin. Their effective administrative and technical supports have greatly facilitated our research. I also thank my colleagues for sharing their academic and life experiences. They are jingjing Pan, Parth Raj Singh, Yunniel Arencibia Noa, Hai Dang Vu, Tien Thanh Nguyen, Safouane Noubir, Alexis Duhamel, Irfan Ali Tunio, Gatien Septembre, Xiao Yang and others. I will never forget the many friends I met in France, especially Yao Ma, Zijian Li, Ziwei Xu, Zhongchao Qiao. They have greatly enriched my daily life in the past three years.

Finally, I would like to extend my deep gratefulness to my father, my mother and my two brothers, for their endless love and trust in me. Particularly, I would like to thank my boyfriend, Zhe Fu. He is always there, willing to listen to me, comfort me and support me.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

In this chapter, we first introduce the context of this dissertation. Then, we discuss about the research problems and present our contributions. Finally, the organization of the document is given.

1.1 Context

1.1.1 Technology Trends

Based on the emerging trend of chip manufacturing, Gordon Moore predicted in 1975 that the number of transistors on a chip doubles approximately every two years [15]. This prediction is called *Moore's Law*, and the period of doubling the integrated transistors on chips is often

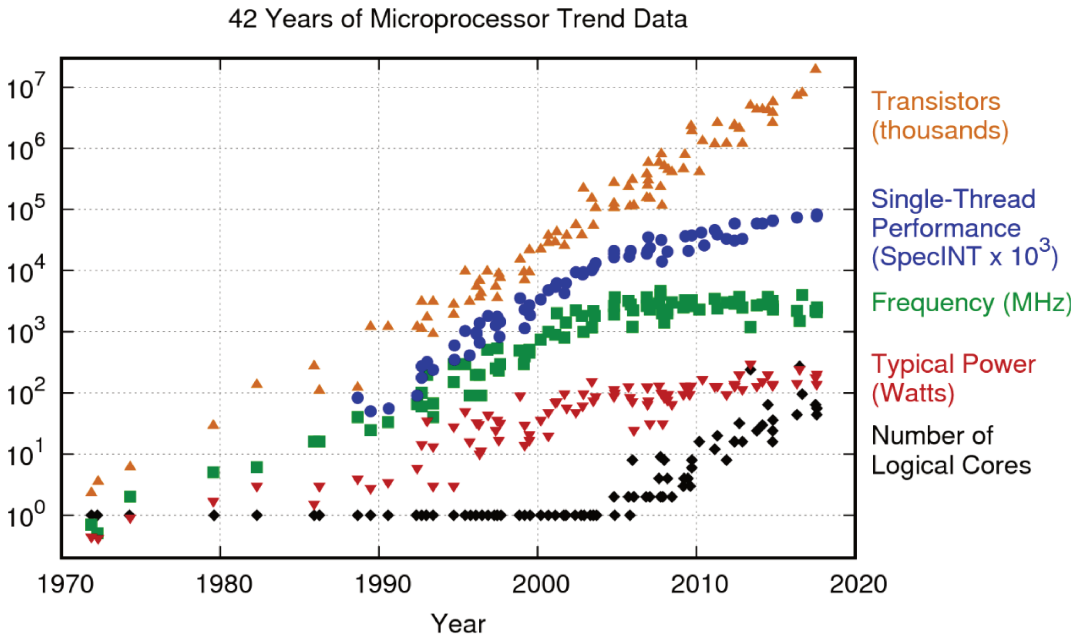

Figure 1.1: Microprocessor trend from 1970 [1].

quoted as 18 months. The *Moore's Law* can be seen in Figure 1.1, which shows that the number of microprocessor transistors (orange dots) increases linearly after 1970. The increase in on-chip transistors is driven by the continued success of reducing the technology size of transistors in the semiconductor industry. In the meantime, scaling down the device dimensions reduces supply voltage and increases frequency by the same degree, thus the power density can be kept constant. This is known as *Dennard scaling* [16]. Unfortunately, the leakage power continues to rise due to the influence of quantum effects on new technology nodes. To keep the leakage power manageable, the supply voltage cannot be reduced anymore suggesting the end of *Dennard scaling*. The operating frequency also cannot continue to increase (green dots in Figure 1.1) due to some potential problems. On one side, higher frequencies increase the power consumption, and the circuit may eventually self-destruct once the power consumption exceeds 100W (*Power Wall Effect*) [17] (red dots in Figure 1.1). On the other side, the power density also increases and leads to higher temperatures, while the higher temperatures result in higher leakage and in turn lead to higher temperatures again. This phenomenon would also damage the circuit.

Power and power density problems show the importance of not increasing the operating frequency. Traditionally, single-core systems satisfy the ever-increasing application demands by primarily increasing the operating frequency, which is not practical due to the breakdown of *Dennard scaling*. As also shown in Figure 1.1, the circuit designers shift their focus on integrating multiple cores into a single chip around 2006 (black dots). Compared with single-core counterparts, multi-core systems are able to deliver the same computing performance at lower frequencies and low power consumption. With the continuous progress of technology,

many-core systems have emerged that integrate tens or hundreds of cores in a single-die. The integration of more than 1000 cores in a chip has been demonstrated feasible in recent years. For example, Epiphany-V [18] has 1024 cores developed by *Adapteva* on the 16-nm technology node in 2016. Multi-core and many-core systems have become efficient solutions to deliver good trade-offs between performance and power. They have thus progressively emerged as possible solutions in current and future embedded systems for mobile phones, high-performance computing, and automotive domains as examples.

1.1.2 Cluster-based Multi/Many-core Platforms

Multi/many-core platforms contain a set of processing elements (*e.g.*, programmable cores or dedicated hardware units), memories, and communication resources (*e.g.*, bus or network-on-chip (NoC)).

A homogeneous platform contains multiple processing elements of the same type. Such platforms deliver design scalability and there exists industrial homogeneous many-core platforms, such as *Intel* Single-chip Cloud Computer (SCC) 48-cores [19] and *Kalray MPPA2®-256* [20]. In these platforms, one or several cores and some communication and memory resources are grouped into identical clusters (or tiles) that are connected via communication resources. Each cluster is a *Voltage Frequency Island* (VFI) [4], inside which the cores share the same voltage/frequency (v/f) level. These platforms can be regarded as homogeneous cluster-based platforms.

A heterogeneous platform is composed of different types of processing elements that provide different performance, power and energy characteristics. The distinct features of different core types can be exploited by various applications to achieve performance and power/energy efficiency trade-offs. In heterogeneous cluster-based platforms, cores within each cluster can have the same type and share the same v/f level, while core types can be different from one cluster to another. One example of such platforms is the Samsung Exynos 5 Octa (5422) processor [21], as shown in Figure 1.2.

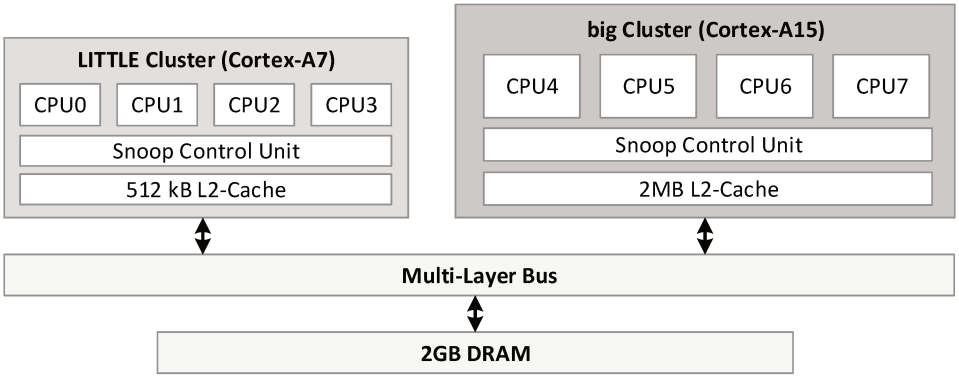

Figure 1.2: Exynos 5 Octa (5422) multi-core platform based on ARM's big.LITTLE architecture.

The Exynos 5 Octa (5422) is based on the Arm big.LITTLE architecture [22]. It integrates the so-called big cluster and little cluster in two different VFIs. The big cluster consists of four high-performance cores (ARM Cortex-A15), while the little one has four low-power cores (ARM Cortex-A7). Each cluster has its own L2-Cache to support the communication between cores. MediaTekHelio X30 (MT6799) [23, 24] is another example that integrates clusters with ARM Cortex-A73, Cortex-A53 and Cortex-A35. ARM big.LITTLE cluster-based platforms are widely used in the field of mobile phones to achieve performance and energy trade-offs.

Heterogeneous cluster-based platforms (*e.g.*, ARM big.LITTLE) have the potential to be scaled to many cores, due to its inheritance of the design scalability feature of homogeneity (inside a cluster). It can be expected that there can be more different clusters and more cores inside each cluster in future systems, such as the platform studied in [4, 25]. Our work focuses on cluster-based multi/many-core platforms, that consist of different numbers of clusters and different numbers of cores within each cluster. We consider both homogeneous and heterogeneous cluster-based platforms in our experimental evaluations. It is worth noting that the cluster-based platform having one cluster is equivalent to a homogeneous platform, while the heterogeneous cluster-based platform having one core in each cluster is equivalent to a generic multi-core platform.

1.1.3 Task-dependent Application

In the scope of this dissertation, we consider the execution of task-dependent applications on cluster-based platforms. A task-dependent application consists of a set of tasks, and each task represents an atomic, non-preemptive, code. In such an application, the output data of one task can be the input data of another one. It indicates that there exists precedence constraints between tasks, where the execution of one task may depend on the completion of other tasks. Figure 1.3 gives one example of a task-dependent application app_1. This application has four tasks and its $task2$ executes after $task1$. According to the discussion in [26], task-dependent applications with precedence constraints are the most typical application model. Compared to the application model that considers tasks independent of each other, a task-dependent application model is more realistic to represent application behavior. In addition, we consider each application has a timing constraint. Each application executes periodically, and all application tasks have to finish their executions within a predefined *period*. It is possible to apply our work to sporadic tasks (*i.e.,* tasks with irregular arrival time [27]), which would be further discussed in Section 7.1.

For all application tasks, their executions are fulfilled by using processing and communication resources of platforms. Task mapping defines the allocation of application tasks on platform resources and the execution order (*i.e.*, scheduling) of tasks on a given core. When mapped on platform resources, each application task takes execution time to process data. Besides, each task consume power to finish its execution. Power consumption can be categorized into dynamic and static parts. Dynamic power is proportional to the frequency and the square of the voltage (*i.e.*, $\propto v^2 f$), which is consumed in charging and discharging the transistors associated with the

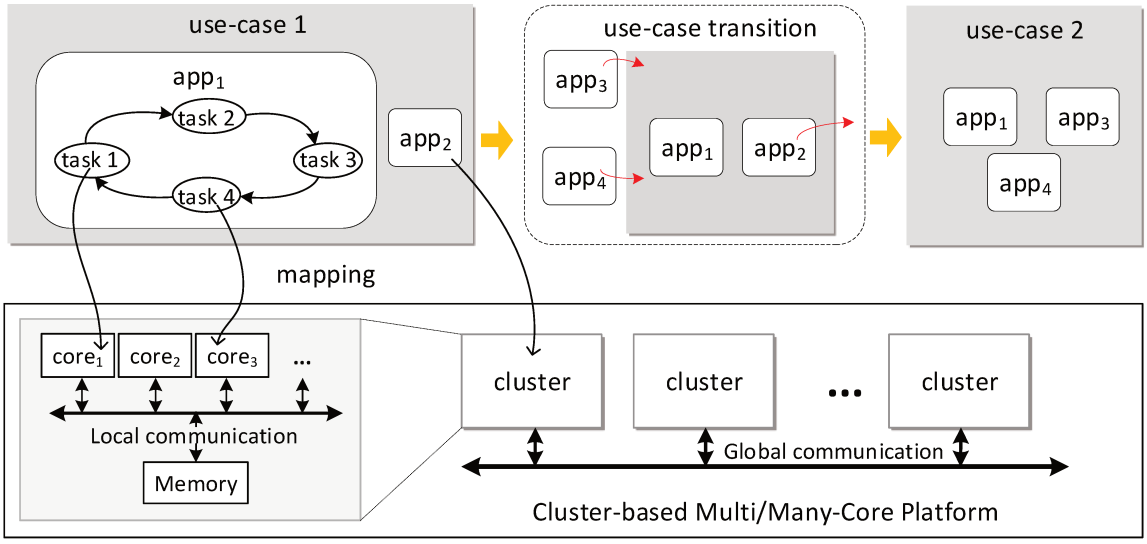

Figure 1.3: Dynamic execution of multiple applications in two different use-cases on a cluster-based multi/many-core platform.

task execution. Static power is consumed due to the always present sub-threshold and gate oxide currents [28]. At a fixed v/f level, this dissertation considers that different tasks have different power consumption when executed on the same core. This is a more realistic consideration compared to the works [4, 29] that assume all tasks have the same power consumption on the same core.

1.1.4 Run-Time Management of Multiple Task-dependent Applications

The increasing number of cores on multi/many-core platforms provides possibilities to execute more applications at the same time. In the context of multiple task-dependent applications executed in cluster-based multi/many-core systems, the set of simultaneously active applications may vary over time as some applications can be active or inactive during system execution. We define a set of simultaneously active applications as a use-case (*i.e.*, scenario) according to [30]. Figure 1.3 depicts two different use-cases. In use-case 1, app_1 and app_2 are active. Due to the state modification of some applications (*e.g.*, app_3 and app_4 are newly active, app_2 becomes inactive), active applications in use-case 2 are different from those in use-case 1. The state modification (*i.e.*, active/inactive) of applications, hereinafter referred to as application execution dynamism, can be caused by user decisions.

Application execution dynamism indicates that the application workloads supported by the platform can vary dynamically, which arises the need for run-time management of systems. As previously discussed, application execution can only be supported after application tasks are mapped on platform resources. However, as active applications can be different from one use-case to another, a new use-case needs the previous mapping to be partially or

completely redefined to support the execution of newly active applications. Therefore, run-time management has to adopt *Dynamic Task Mapping* techniques to adapt application task mapping to different use-cases accordingly.

1.1.5 Run-time Management for Energy Efficiency

Run-time management has to take *energy efficiency* into account in today's multi/many-core systems. Energy efficiency refers to using less energy consumption to execute the same applications. Energy consumption is the integration of its power consumption over time (*i.e.*, $Energy = Power \times Time$). Reducing energy consumption is significantly important to extend the battery life of systems. The increasing number of active applications and the rising complexity of platforms (*e.g.*, increasing heterogeneity in resources, more v/f domains) make system energy efficiency a crucial optimization target. *Dynamic Task Mapping* plays an important role in energy efficiency. Applications mapped on platform resources consume power to finish their computational and communication activities. Due to the platform heterogeneity, tasks executed on different core types may have different execution time and power consumption, thus their energy consumption can also change. It means that appropriate utilization of platform resources can result in better energy efficiency for all application tasks in the system.

Additionally, energy efficiency is often achieved through *Dynamic Voltage Frequency Scaling* (DVFS) [31, 32]. It refers to the technique that dynamically modifies the operating frequency and voltage of cores that executes application tasks. According to [28, 33], there exists a relationship between operating frequency and voltage in CMOS circuits, which suggests that higher frequencies require the support of higher voltages. In the Samsung Exynos 5 Octa (5422) processor [21], users can only scale the operating frequency, based on which the operating voltage is automatically adjusted. Lower v/f levels can lead to the reduction of dynamic power consumption (due to $\propto v^2 f$) and increase the execution time of tasks (due to degraded processor performance) at the meantime. Dynamic energy consumption can be reduced if the decrease in power consumption is greater than the increase in execution time [4, 34]. Note that static power/energy can be mitigated by shutting down certain platform resources but it is difficult to do so in the active mode of systems [28, 35].

DVFS can be applied at different granularities, depending on the supports of different platforms. *Global DVFS* allows all the cores in a system to share the same v/f level, while *Per-core DVFS* allows each core to have its own distinct v/f level. *Per-cluster DVFS* (*i.e.*, VFI) is a compromise solution that allows several cores in a cluster to share the same v/f level, and different clusters can support different v/f levels. As the discussions in [31, 36], *Per-cluster DVFS* is widely used in advanced many-core systems, making a trade-off between the feasibility of *global DVFS* and the efficiency of *per-core DVFS*.

The objective of this dissertation is to propose run-time management strategies, employing dynamic task mapping and DVFS (*i.e.*, per-cluster) together, to achieve energy efficiency of

multiple task-dependent applications executed dynamically on a cluster-based multi/many-core platform. Energy efficiency can be characterized by energy consumption or average power consumption in a certain period of time. In the experimental evaluations of this work, we focus on the optimization of the average dynamic power consumption of active applications.

1.2 Problem Statement

For the run-time management purpose of energy efficiency of task-dependent applications executed in cluster-based multi/many-core systems, we study two research problems on run-time management decisions (*i.e.*, dynamic task mapping and DVFS control) and run-time management evaluation.

1.2.1 Coordination of Dynamic Task Mapping and DVFS Control

The first research problem studied in this dissertation work is: how to appropriately apply dynamic task mapping and DVFS to achieve energy efficiency of *task-dependent applications* in cluster-based systems. Energy efficiency of cluster-based systems has been studied for independent tasks, while it remains an open and complex question for task-dependent applications. On one hand, obtaining the optimal application mapping (for energy efficiency) is a NP-hard problem [37, 38], where the solution space exploration increases with the number of applications and the number of cores. The space exploration becomes even larger when DVFS is taken into account. On the other hand, different task mappings lead to different DVFS possibilities within a cluster and in the overall system, which complicates run-time management decision issues. This work aims to achieve near-optimal management solutions for both local optimization within a cluster and global optimization in the overall system.

Local optimization within a cluster

We first study the local optimization of energy efficiency within a cluster. At the cluster level, we consider homogeneous cores. For the purpose of energy efficiency, it is important to execute active applications (in a use-case) at a low cluster v/f level. This is a challenging mission due to the fact that simultaneous active applications in the same cluster compete for platform resources but share the same cluster v/f. The more used cores for one application means fewer cores for other applications, which may consequently worsen the performance of other applications. The poor performance of an application can significantly increase the cluster v/f, thereby increasing the power/energy consumption of other applications in the same cluster. However, existing works address the mutual influence between application mappings and cluster v/f configurations at the cost of increased strategy complexity. A management strategy that offers a good trade-off between energy efficiency and complexity is thus required.

Global optimization in the overall system

Then, we study the global optimization or energy efficiency in the overall system. At the chip-level, application-to-cluster assignment plays a crucial role to achieve energy efficiency. In the scope of this work, we assume that each application can be executed in different clusters, but all tasks of an application are assigned to the same cluster to avoid communication costs among clusters. Different application-to-cluster assignments can lead to different possibilities of cluster v/f configurations and eventually change the system energy efficiency. In the case of heterogeneous platforms, the application-to-cluster assignment problem becomes even more complex because the performance and power characteristics vary from one cluster to another. Therefore, it is required to assign applications to clusters carefully to achieve the global optimization target. However, most existing application-to-cluster assignment solutions focus on 2-clusters platforms (*e.g.*, ARM big.LITTLE). They may have limitations to deal with platforms with more different clusters or with more cores in a cluster.

1.2.2 Evaluation of Run-Time Management Strategy

With our proposed run-time management strategies, an interesting research problem arises: how to evaluate run-time management strategies to guarantee that non-functional requirements (*e.g.*, application latency, resource usage, energy efficiency) will be respected during system execution. System-level modeling and simulation approaches allow early detection of potential design issues. However, most of the existing system-level frameworks only support a static mapping of applications on platform resources without considering the run-time management effects. It is required to extend system-level simulation-based approaches for run-time management strategy evaluations.

1.3 Main Contributions

Towards the above-mentioned research problems, this dissertation makes the following three main contributions.

Contribution 1: Run-time management for local optimization within a cluster

We propose a run-time management strategy to optimize average dynamic power consumption of multiple applications executed dynamically within a cluster. The overview of the run-time management for local optimization is presented in Figure 1.4.

For active applications within a cluster, the local management strategy is responsible for determining task-to-core allocation and scheduling and setting the cluster v/f level. The proposed strategy can achieve near-optimal energy efficiency of a cluster while meeting

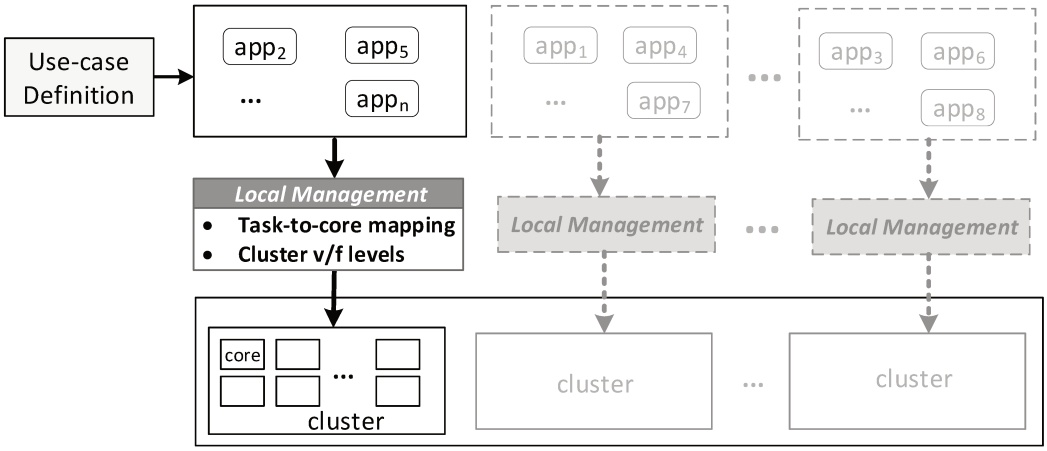

Figure 1.4: Overview of the run-time management for local optimization within a cluster.

application timing constraints and platform resource usage constraints. Compared to state-of-the-art strategies, our strategy has lower complexity to achieve near-optimal solutions.

Contribution 2: Run-time management for global optimization of the overall system

We introduce run-time management strategies to optimize average dynamic power consumption of multiple applications executed dynamically in the overall system globally. The overview of the run-time management for global optimization is presented in Figure 1.5.

For all active applications in the overall system, a global management strategy determines the application-to-cluster assignments and sets cluster v/f levels. Then, a local management strategy optimizes the task-to-core allocation and scheduling within each cluster. The proposed management strategies are effective to achieve a near-optimal result for energy efficiency of

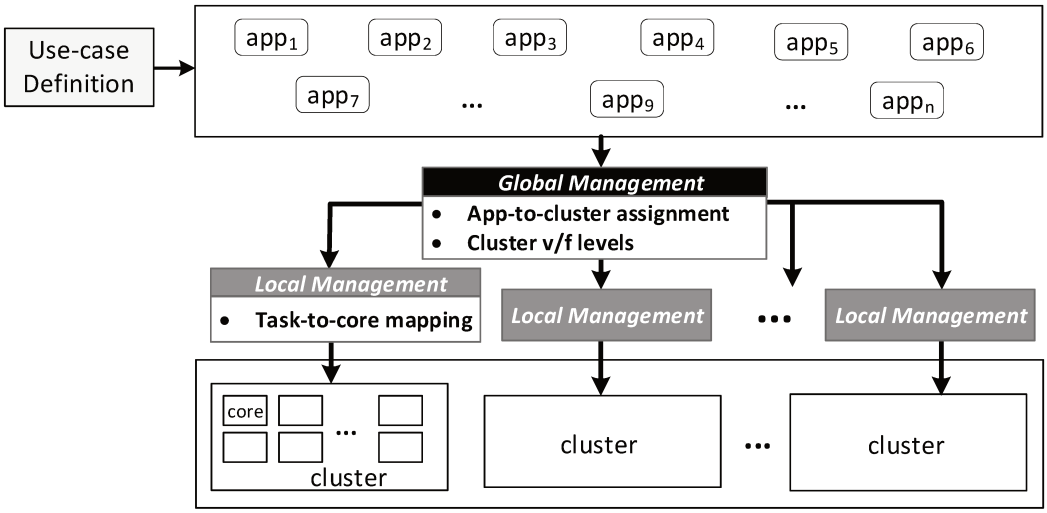

Figure 1.5: Overview of the run-time management for global optimization in the overall system.

the overall system, satisfying application timing constraints and platform resource constraints. The proposed strategies have reasonable strategy complexity and have potential scalability to manage systems with different numbers of applications/clusters/cores.

Contribution 3: System-level run-time management strategy evaluation

We presents a novel system-level modeling and simulation approach to allow the evaluation of run-time management strategies on multi/many-core systems. Figure 1.6 depicts the overview of the modeled and simulated components in our approach, including application models, a platform model and run-time management modules.

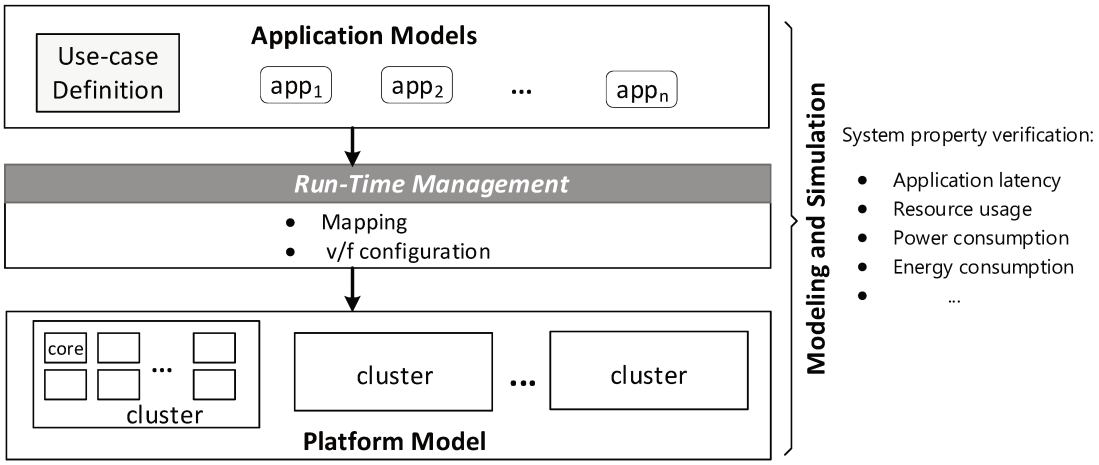

Figure 1.6: Overview of modeled and simulated components with application models, a platform model and run-time management modules.

Our proposed simulation approach can be used to evaluate different run-time management strategies (*e.g.*, dynamic task mapping, DVFS) on multi/many-core systems. The proposed approach has been integrated in an industrial modeling and simulation framework [39].

1.4 Dissertation Organization

The remainder of the dissertation is organized as follows.

- Chapter 2 presents state-of-the-art researches related to run-time management strategies. Firstly, it focuses on the existing strategies using dynamic task mapping. Then, it summaries the related strategies that apply both dynamic task mapping and DVFS. After that, it discusses the existing management structures and the existing system-level evaluation approaches for run-time mapping strategies.

- Chapter 3 introduces the system models, including application, platform, and mapping models, used throughout the dissertation. Besides, in order to evaluate system energy efficiency, the power/energy models are presented and the models are validated by some measurements in the ARM big.LITTLE platform.

- Chapter 4 begins with an overview of run-time management for the local optimization of energy efficiency within a cluster. It then gives a summary of the existing strategies, which are particularly used as counterparts to our proposed management strategy. After that, the details of our proposed management strategy are demonstrated. Finally, the advantages of our proposed strategy are shown in the experimental results.

- Chapter 5 focuses on run-time management for the global optimization of energy efficiency in the overall system. It first presents an overview of this work. Then, it highlights the existing strategies that are particularly used as counterparts to our proposed application-to-cluster assignment strategy. Furthermore, the global management problem is defined, followed by the details of proposed management strategies. Finally, the experimental results are given.

- Chapter 6 concerns the system-level evaluation of run-time management strategies. First, it presents the motivation of this work. Then, it compares our proposed simulation approach to the existing trace-driven simulation approach. After, it presents the proposed system-level simulation approach and the experimental evaluations.

- Chapter 7 concludes the contributions presented in previous chapters and discusses some possible improvements in future work.

Chapter 2

State-of-the-art

Contents

Many efforts have been done to cope with run-time management of multiple applications executed dynamically on multi/many-core systems. This chapter first summarizes the existing works concerning dynamic task mapping strategies. Then, we discuss how state-of-the-art approaches apply dynamic mapping and DVFS together in cluster-based multi/many-core systems. After that, the existing management structures are compared. Finally, the state-of-the-art system-level evaluation approaches for run-time management strategies are presented.

2.1 Dynamic Task Mapping

Task mapping can be performed either statically or dynamically. *Static task mapping* defines a mapping of application tasks on platform resources at design-time, and the mapping does

not change during run-time system execution. Generally, the applied mapping is an optimized solution that is obtained through extensive *Design Space Exploration* (DSE) using simulated annealing [40] or genetic algorithm [41]. As discussed in [14], static mapping techniques are suitable for a predefined set of applications executed on multi-core systems. However, it cannot support system execution with different sets of active applications in various use-cases. Note that a use-case refers to a set of simultaneously active applications (first defined in Chapter 1).

To overcome the limitation of static task mapping, *dynamic task mapping* is proposed. It can adapt task mappings of active applications in different use-cases. The existing strategies of dynamic task mapping can be classified as *on-the-fly* and *hybrid*. On-the-fly mapping strategies establish a mapping for a set of active applications without previously analyzed results. [42–45] employ heuristic algorithms to allocate and schedule application tasks onto platform resources, taking the system current states into account. However, as the number of cores and the number of active applications increase, on-the-fly mapping strategies may not guarantee the mapping scalability due to the heavy processing bottlenecks at run-time.

In order to release the run-time computation burdens but still achieve dynamic system configurations, hybrid mapping strategies establish a mapping for a set of active applications based on some design-time optimized mappings. Hybrid mapping strategies are usually applied for the mappings of complex applications, such as task-dependent applications with precedence constraints. In this case, application precedence constraints are considered at design-time and retained at run-time. Inspired by the survey of [2], Figure 2.1 gives an overview of hybrid mapping strategies.

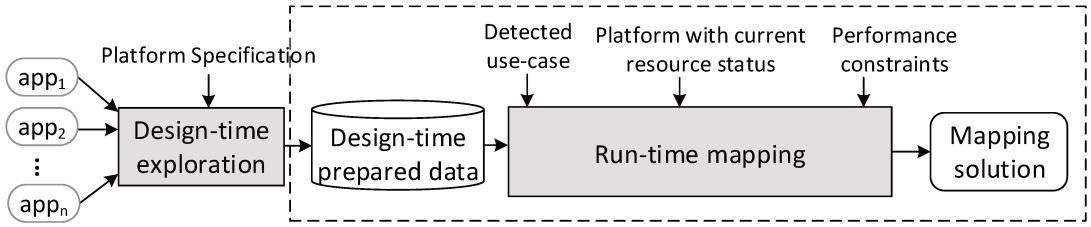

Figure 2.1: Overview of hybrid mapping, including design-time preparations and run-time mapping, inspired by [2]. The main focus of this dissertation work is highlighted by dotted box.

At design-time, DSE is performed to explore some optimized mappings, taking into account the run-time executing applications, platform specifications, and design objectives. The explored mappings are stored as design-time prepared data and then used to establish different mappings at run-time. Run-time mapping is performed based on the detected use-case (with a set of active applications), the current resource status of platform and the performance constraints. The amount of design-time prepared data influences the complexity of the run-time

mapping. The more information prepared at design-time, the less computation intensity at run-time, but it consumes more storage space. It is important to consider the trade-off between the design-time storage overhead and the run-time computation effort. Design-time prepared data and run-time mapping are the main focus (highlighted by dotted box) of this dissertation work.

Among the existing hybrid mapping strategies, the design-time preparation can be classified into *use-case-based* or *application-based*. Use-case-based preparation refers to preparing design-time optimized mappings for a set of active applications in *each use-case*, while application-based preparation refers to preparing design-time optimized mappings for *each application*.

2.1.1 Hybrid Mapping with Use-case-based Preparation

The overview of hybrid mapping strategies with use-case-based preparation is illustrated in Figure 2.2. At design-time, optimized mappings are prepared for a set of active applications in *each use-case*. The prepared mappings can be obtained by any static mapping strategies. For example, work [46] and [47] employ such hybrid strategies, and they employ integer programming (ILP) and a genetic algorithm respectively to achieve prepared mappings. At run-time, the prepared mapping for the detected use-case is applied accordingly without any modification. Efficient run-time mapping can be achieved with less computation effort.

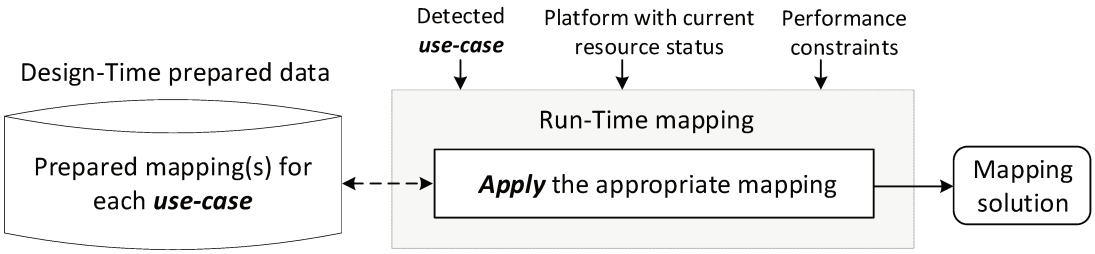

Figure 2.2: The overview of hybrid mapping strategies employing use-case-based design-time preparation.

However, a lot of storage and computation efforts are paid at design-time. Considering different combinations of n applications that might be executed on multi/many-core systems, the number of possible use-cases is $2^n - 1$. This is because each application has two states (*i.e.*, active/inactive), and we eliminate the use-case with no active application. When the parameter n is large, it could cost a lot of memory space to store at least one optimized mapping for each use-case. Besides, some design-time DSE approaches have to evaluate a significant number of mappings until the optimized mapping required for all the possible use-cases is obtained. Therefore, preparing design-time mappings for all the possible use-cases is not scalable.

2.1.2 Hybrid Mapping with Application-based Preparation

In recent years, hybrid mapping strategies with application-based preparation has attracted more attention. The overview of such strategies is presented in Figure 2.3. At design-time, one or several optimized mappings can be prepared for *each application*. For example, the work in [14] prepares the best performance mapping for each application. In [3, 48], the prepared mappings have different trade-offs between application performance and resource usage (or energy efficiency). Then, run-time mapping can be performed only for newly detected active applications (*i.e.*, in a new use-case) or all active applications in the current use-case. Run-time mapping consists of two steps. First, *application mapping selection* is performed to select one of the prepared mappings for each active application. If there is only one prepared mapping for an application, the prepared mapping is directly used. Second, *application mapping combination* is performed to obtain a so-called combined *mapping* by processing the selected *mappings*.

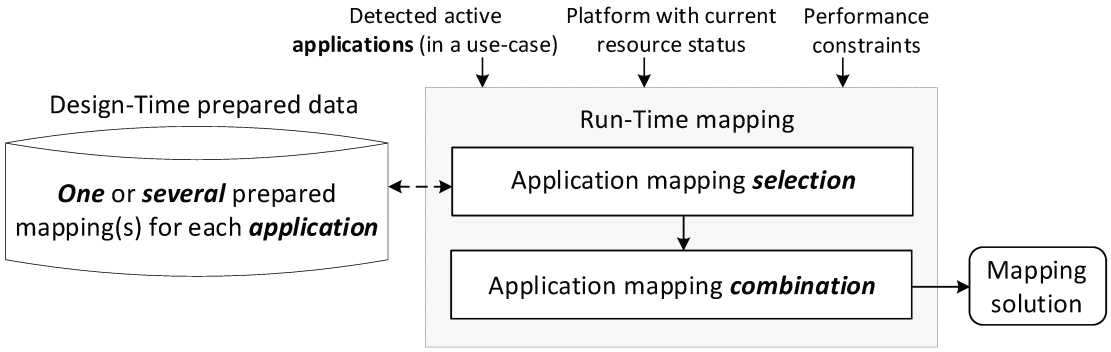

Figure 2.3: The overview of hybrid mapping strategies employing application-based design-time preparation.

In hybrid mapping strategies with application-based preparation, the total number of prepared mappings is highly dependent on the number of supported applications, not the number of possible use-cases. Compared to use-case-based preparation, application-based preparation can greatly reduce the number of prepared mappings that are explored and stored at design-time. On the other hand, more calculation might be required to obtain good-quality run-time mapping solutions.

For a running use-case, the quality of its mapping solution is highly determined by the *application mapping selection* and *application mapping combination* steps. These two run-time steps have been studied to optimize active applications *individually* and *holistically*. The former considers the optimization for each individual application. For a set of active application, it finishes the optimization for one application then another one. The latter considers the optimization of all active applications at the same time.

Optimizing Applications Individually at Run-Time

The overview of hybrid mapping strategies that optimize applications individually is illustrated in Figure 2.4. These strategies assume that the run-time active applications wait in a ready queue and that optimization is performed for each application successively. For an active application, one of its design-time prepared mappings is selected and then combined with the mapping of already existing applications on the platform. Note that system status (*e.g.*, application performance, number of used cores) is updated each time a new application is mapped. Based on the updated system status, the next application in the ready queue (depending on the adopted ordering strategy) is considered until all applications are mapped to the platform. This *application mapping combination* processing is regarded as First-Come-First-Served (FCFS), which maps active applications one after another. Particularly, the FCFS applied in [7, 13, 48] allows only one application to be mapped to each core, which means that there is no resource competition between different applications in a core.

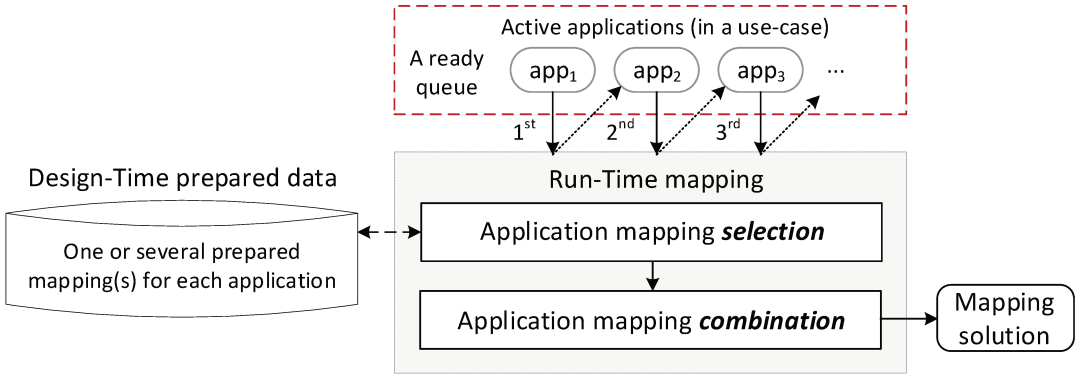

Figure 2.4: Overview of hybrid mapping strategies optimizing applications individually at run-time.

In [13], the authors aim to map applications on homogeneous multi-core platforms meeting application performance constraints. At design-time, this work evaluates an application mapped on different numbers of cores in order to obtain optimized mappings which have different trade-offs between resource usage and throughput. At run-time, for any given active application, it selects the prepared mapping that satisfies the throughput constraint with the least number of cores. The selected mapping is then applied to the processing cores that are topologically close to each other to reduce communication costs between cores. A similar hybrid strategy is used in [48] to optimize the energy consumption of a heterogeneous system under application performance constraints. In the considered heterogeneous platform, there are different core types, general-purpose processor (GPP), including digital signal processor (DSP), Accelerator (ACC) and reconfigurable hardware (RH). At design-time, the prepared mappings for each application depend not only on the number of cores but also on the different core types. For

example, an application can be mapped on 2GPP&1ACC or 1GPP&2RH at design-time to prepare for different availability of core types at run-time. At run-time, application mapping selection is performed according to resource availability and application throughput.

As highlighted in [14], hybrid mapping strategies that map individual applications one after another might not achieve efficient mappings for all active applications. As the availability of platform resources varies for different applications, more platform resources or more efficient cores (in heterogeneous platforms) can be occupied by first considered applications. As a result, fewer efficient resources are left for later considered applications, which can degrade their performance. That means first considered applications have higher optimization priorities than later ones. Therefore, optimizing applications individually might not lead to the optimization of the overall system.

Optimizing Applications Holistically at Run-Time

The overview of hybrid mapping strategies that optimize applications holistically is illustrated in Figure 2.5. Such strategies optimize for all active applications as a whole, even sacrificing the performance of some applications to achieve overall optimization of all active applications. In this case, hybrid mapping strategies of [3, 10, 14, 34] allow one core to execute tasks of different applications to achieve less resource usage.

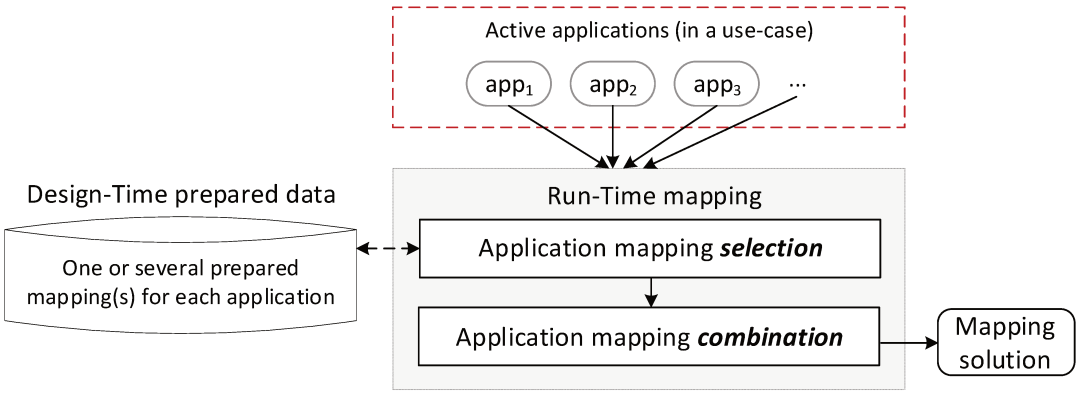

Figure 2.5: Overview of hybrid mapping strategies optimizing applications holistically at run-time.

In [10, 14], the *application mapping selection* step is first accomplished for all the active applications, and the *application mapping combination* step is then performed to combine all the selected mappings together. These two works gives more focus on *application mapping combination*. The work in [10] aims to maximize the application throughput under a predefined energy budget of a heterogeneous multi-core platform. For any active application in a new use-case, the prepared mapping with the maximum throughput under the energy budget is selected.

All the selected mappings of active applications are simply merged to form the initial combined mapping, thus the tasks of different applications can be mapped onto the same core. To avoid a heavy computation burden in one core and achieve better application throughput, a heuristics is then performed to iteratively migrate tasks from one core to another for further mapping optimization. In [14], since only one design-time mapping is prepared for each application, the prepared mapping of each active application is directly selected at run-time. Two heuristic algorithms (*i.e.*, Longest Available Slot Packing (LASP) and Best Fit Slot Packing (BFSP)) have been proposed to achieve a combined mapping with minimized resource usage on homogeneous platforms.

The hybrid mapping strategies of [3, 34] closely involve *application mapping selection* and *application mapping combination* steps. The work in [3] aims to minimize the energy consumption of heterogeneous multi-core systems under application performance constraints. At design-time, it explores different possibilities of task allocation and scheduling to generate multiple optimized mappings for each application. Each prepared mapping is better than another in terms of application performance or energy efficiency. Figure 2.6 illustrates the run-time application mapping and combination of this work. Based on multiple prepared mappings for each active applications (app_1, app_2 and app_3 ...), there can be different possibilities of mapping selection. For each set of selected mappings, application mapping combination is performed to find a mapping solution meeting application timing constraints and minimizing energy consumption. After mapping combination is finished for all sets of selected mappings, the mapping solution with the best energy efficiency is preferred.

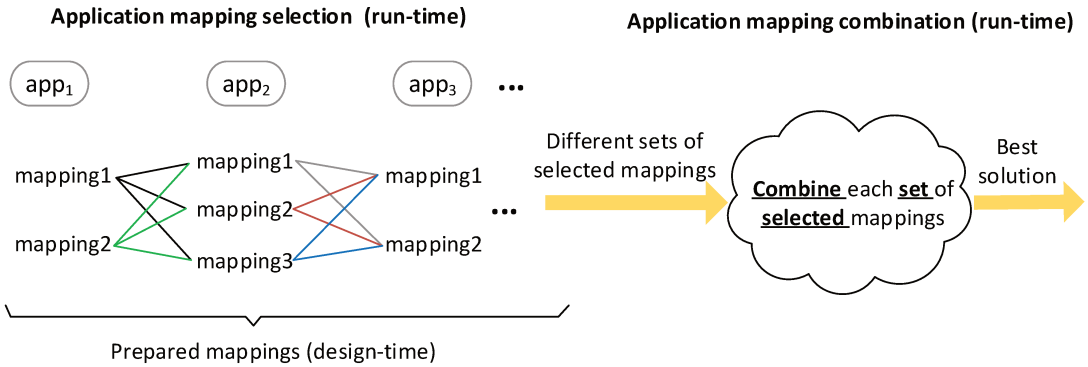

Figure 2.6: Run-time application mapping selection and combination in [3].

A similar approach is used in [34] to consider application mappings in ARM big.LITTLE cluster-based systems. The design-time prepared mappings for each application can use different types of cores. According to active applications at run-time, different prepared mappings are selected and combined in different cores to estimate energy efficiency.

Compared to optimizing applications individually, optimizing applications holistically

can achieve overall optimization for all simultaneously active applications. However, more evaluation efforts should be paid to compromise the resource competition among applications, which can increase the strategy complexity of *application mapping selection* and *application mapping combination* at run-time. Therefore, the main concern of holistically optimizing applications is to reduce the complexity of run-time mapping strategies. Moreover, run-time strategies can become more complex when DVFS is taken into account. In this case, the performance an energy consumption of all active applications depend not only on the run-time mapping solution, but also on the DVFS control. Our dissertation work focuses on reducing the complexity of run-time strategies that apply dynamic mapping and DVFS together.

2.2 Applying Dynamic Task Mapping and DVFS

As previously introduced in Chapter 1, cluster-based multi/many-core systems support *per-cluster DVFS*, where the core in each cluster shares the same v/f level. In such systems, dynamic task mapping (*e.g.*, on-the-fly, hybrid) and DVFS techniques are often applied to reduce power/energy consumption. In the literature, the two techniques have been applied separately or coordinately, as depicted in Figure 2.7.

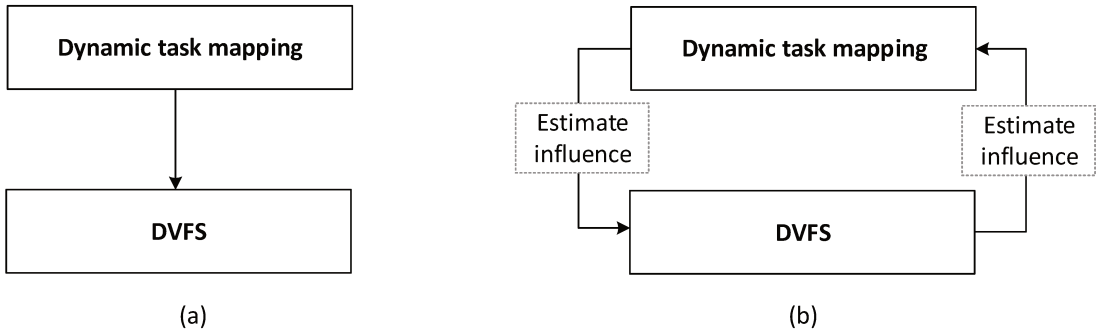

Figure 2.7: Applying dynamic mapping and DVFS (a) separately and (b) coordinately.

Applying dynamic task mapping and DVFS separately (see Figure 2.7.(a)) refers to the two techniques applied separately into two independent steps. It considers one technique and then considers the other one. Typically, task mapping is performed first without considering its influence on v/f configurations, and DVFS is then applied based on the obtained mapping. On the other hand, applying dynamic task mapping and DVFS coordinately (see Figure 2.7.(b)) refers to considering the mutual influence between the two techniques. The potential influence from one technique on the other is estimated during the decision-making process.

2.2.1 Applying Dynamic Task Mapping and DVFS Separately

In our considered cluster-based multi/many-core systems, existing works usually fix dynamic task mapping first then perform DVFS to achieve power/energy goals. The work in [4] aims to reduce the energy consumption of *independent periodic tasks* in heterogeneous cluster-based multi/many-core systems (up to 4 clusters, 24 cores). The management overview of this work is given in Figure 2.8. This work first maps all independent tasks to different clusters (see step (1) in Figure 2.8). It uses a Low-Energy-First (LEF) based mapping strategy, which assigns tasks successively to the cluster that can achieve the lowest energy consumption for each individual task at the maximum cluster v/f level. This work holds the hypothesis that the energy consumption of a task depends on the used core type and the configured cluster v/f level, meaning that different priorities are given to different clusters (at the maximum cluster v/f levels) when performing task mapping among clusters. Then, workload balance is targeted inside each cluster. Finally, after completing the mapping of all tasks, cluster v/f levels are decreased as much as possible under timing constraints (see step (2)).

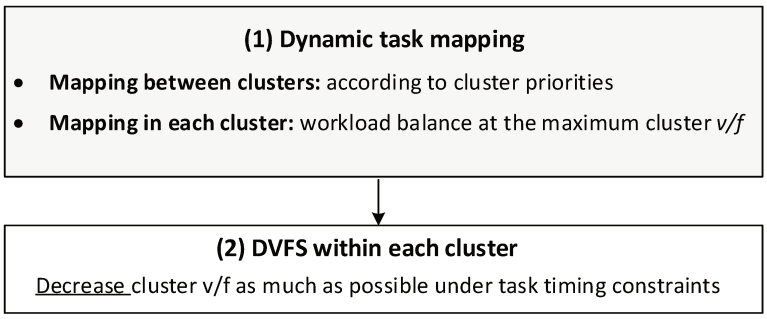

Figure 2.8: Overview of applying dynamic mapping and DVFS separately in [4].

Similarly, the work in [49] also gives different mapping priorities to the clusters in the Arm big.LITTLE system (*i.e.*, 2 clusters, 4 cores in each cluster). The strategy of this work can be understood as Low-Power-First (LPF) based mapping strategy, where each application attempts to be assigned to the lowest power cores (*i.e.*, little cores) first. If the performance constraint is not satisfied, high-performance cores (*i.e.*, big cores) are used. After applications assignment, the cluster frequency is reduced with respect to a pre-defined power budget. The two previously mentioned works give the highest priority to one cluster when performing application-to-cluster assignments. For platforms with more cores in the cluster, we can predict that the workload of a cluster can be very heavy with increasing numbers of active applications. Consequently, the frequency and the energy consumption of one cluster can be very high, while other clusters can be empty without any executing application. Since platform resources in low-priority clusters may not be fully used, opportunities for further energy optimization may be missed.

For the Arm big.LITTLE system, the work in [34] first performs hybrid mapping then executes DVFS. It prepares several design-time mappings for each application, and each

prepared mapping can use different types of cores. For run-time active applications, the prepared mappings are selected and combined such that the overall energy consumption is minimized without violating the application timing constraints and platform resource constraints. After that, DVFS is performed based on the already fixed mapping of active applications for further energy optimization. This work classifies application workloads into compute-intensive, memory-intensive and mixed-intensive based on Memory Reads Per Instruction (MRPI=$\frac{\text{L2 cache read refills}}{\text{instructions retired}}$). Low and high MRPI values represent compute-intensive and memory-intensive workloads, respectively. In this work, application workloads (MRPI values) are predicted dynamically in every time interval. The prediction is based on the observed workloads in the previous time interval. According to various MRPI ranges, cluster v/f level is set to different values. Note that there is a table classifying the MRPI range and the corresponding v/f level based on design-time analysis.

On the other hand, also for the Arm big.LITTLE system, the work in [5] not only performs DVFS based on fixed application mappings but also performs mapping migration based on fixed v/f. The management overview of this work is illustrated in Figure 2.9. This work isolates multimedia applications from non-multimedia applications. During task mapping (see step (1) of Figure 2.9), the authors aim to map non-multimedia applications to the big cluster and multimedia applications to the little cluster, due to the dedicated hardware decoders in the considered board (*i.e.*, Odroid XU3 [50]). Workload balance is targeted in each cluster. Then, the v/f level of each cluster is set by CPU frequency scaling governor (see step (2)), which is an operating system module that adjusts CPU frequency and voltage depending on core utilization. According to the cluster v/f, applications can be migrated from one cluster to the other (see step (3)). If the performance of multimedia applications cannot be satisfied at the highest v/f level in the little cluster, some multimedia applications can be migrated to

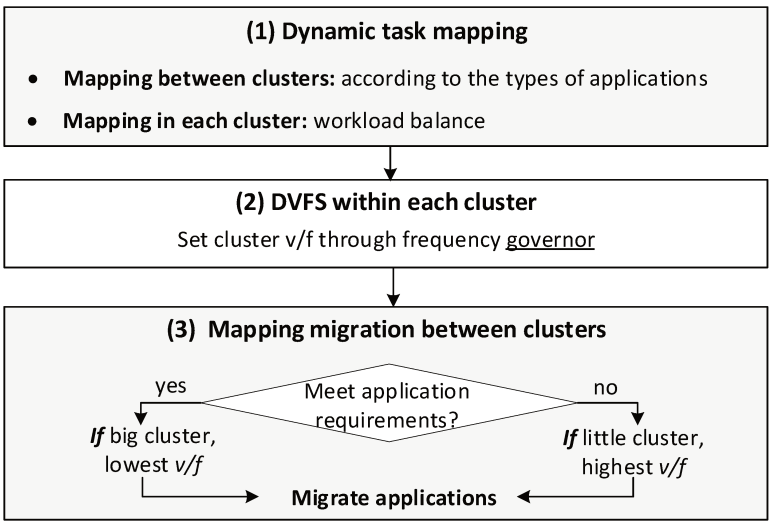

Figure 2.9: The overview of applying dynamic mapping and DVFS separately in [5].

the big clusters. In contrast, for the multimedia applications executed in the big cluster, if a multimedia applications can be executed at a lower v/f level than the minimum v/f level of the big cluster, the multimedia application can be migrated to the little cluster. Similarly, the work in [51] also supports application migration after DVFS. However, this work allows only one cluster to be activated at the same time due to the limitation of software supports. Therefore, when application requirements cannot be satisfied at the highest v/f level in the little cluster, this work migrates all applications to the big cluster.

The above-mentioned works aim to achieve energy efficiency while guaranteeing system performance in cluster-based multi/many-core systems. The first mentioned work consider *periodic independent tasks* executed in a system with 4 clusters, while the other mentioned works focus on the ARM big.LITTLE system with 2 clusters. These works apply dynamic mapping and DVFS in separated steps. Application mapping determines application performance, which can further affect the cluster v/f level required to meet performance constraints. Some mappings may have better energy efficiency at one v/f level but it may result in higher v/f configurations after DVFS, which ultimately leads to high energy consumption. Better management results can be achieved if the influence between the two techniques is taken into account.

2.2.2 Applying Dynamic Task Mapping and DVFS Coordinately

In recent years, the mutual influence between dynamic task mapping and DVFS in cluster-based multi/multi-core systems has received more attention. The coordination between the two techniques can be considered through estimations or iterative evaluations.

In [6], the authors aim to satisfy application performance requirements without violating the given power budget in the Arm big.LITTLE system. Unlike most existing works, this work establishes coordination between application mapping and DVFS by estimating performance gain/loss of applications. The management overview of this work is depicted in Figure 2.10. The work applies Low-Power-First strategy (*i.e.*, LPF discussed in Section 2.2.1) during dynamic mapping (see step (1) of Figure 2.10). Every new active application is first mapped onto the little cluster. When the application does not meet the performance requirement with the highest resource usage in the little cluster, the application is mapped to the big cluster. Each time when a certain application is active or inactive, application remapping is performed to make utilization uniform on cores within a cluster. This work coordinates dynamic mapping and DVFS to respect the power budget within each cluster (see step (2)). On one hand, when the power budget is violated in a cluster, applications can be migrated to the other cluster. After estimation (see step (2.a)), if no application can meet its performance requirement in the *new* cluster, DVFS is used to reduce power consumption in the *current* cluster. On the other hand, when the power budget is honored in a cluster, the cluster v/f level can be reduced in the *currently* considered cluster. After estimation (see step (2.b)), if the new v/f level causes performance violation of an application, more resources are allocated to the application.

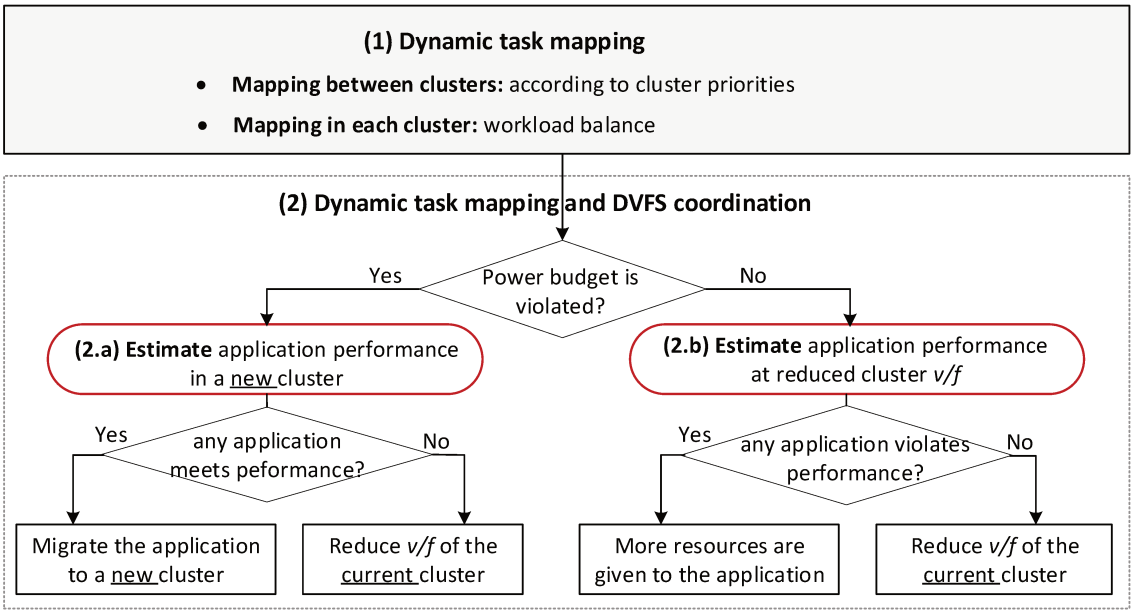

Figure 2.10: Overview of applying dynamic mapping and DVFS coordinately in [6].

Through dynamic estimations of application performances, this work is able to minimize the penalty of management decisions (*i.e.*, on-the-fly mapping, DVFS) and reduce application performance loss.

Particularly for the energy efficiency within a cluster of the ARM big.LITTLE system, the works in [7, 8] coordinate hybrid mapping and DVFS through iterative evaluations. Figure 2.11 illustrates the iterative loops of these two works. Based on several design-time prepared mappings for each application, the two works perform mapping optimization (*i.e.*, including mapping selection and combination) and cluster v/f configuration iteratively at run-time.

The authors of [7] aim to maximize system performance under power density[1] constraint, which is set to guarantee avoiding thermal violations within a cluster. At design-time, it stores two parameters for each application, including average power consumption and application execution time that are dependent on the used cluster, the number of threads and v/f levels. For active applications at run-time, as illustrated in Figure 2.11 (a), mapping optimization and cluster v/f configuration are performed iteratively (highlighted in red) for each application until all active applications are considered. This work optimizes applications individually. For an active application, it selects the thread number and cluster v/f level that can lead to the best throughput of the application under power density constraint. As the available processing resources reduce, the last considered application can only be mapped onto few cores, resulting in poor application performance. Eventually, we could foresee that the cluster v/f level and cluster power consumption can be very high.

The authors of [8] aim to map active applications at a low cluster v/f level within a cluster.

[1]Power density refers to power consumption per unit area.

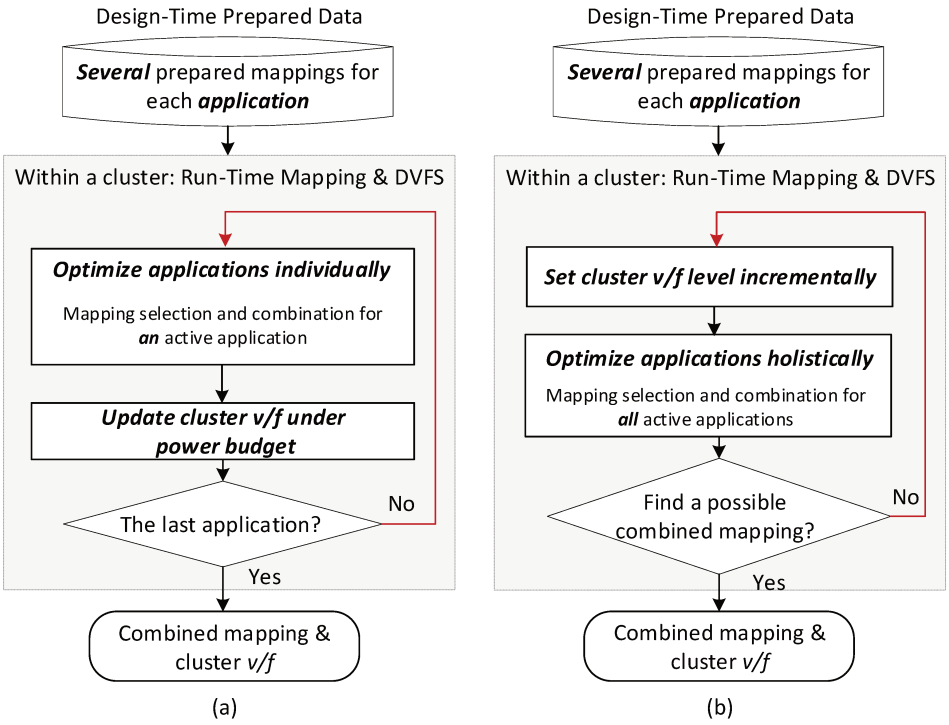

Figure 2.11: Overviews of coordination between hybrid mapping and DVFS within a cluster in (a) [7] and (b) [8].

Its design-time prepared mappings for each application have different trade-offs between the number of used cores and application performance. For the run-time active applications, as illustrated in Figure 2.11 (b), it iteratively (highlighted in red) increments cluster v/f from the lowest level and performs mapping selection and mapping combination at each new v/f level. The exploration stops when a possible mapping solution is found through holistically application optimization, platform resource constraints. This approach can obtain an energy-efficient mapping supporting low cluster v/f. However, a long time can be taken to reach convergence when the frequency increment is small. Therefore, it is important to reduce the number of estimation iterations when coordinating hybrid mapping and DVFS.

To summarize, the above-discussed works coordinate dynamic task mapping and DVFS techniques to achieve a good energy and performance trade-off. Compared to the strategies that separate dynamic task mapping and DVFS into different steps, coordinating the two techniques needs more computation at run-time (due to estimation or iterative evaluations). More attention should be paid to reducing the strategy complexity of dynamic mapping and DVFS coordination. Note that the above-discussed works focus on the *local optimization* within a cluster. On the other hand, for the *global optimization* of a system with multiple clusters, the work in [29] outlines the optimal task mapping and cluster v/f configurations for *periodic independent tasks*. This work assumes that tasks can be assigned continuously in each core and that each task has the same power consumption in the same core. Based on the two

assumptions, this work builds a core utilization relationship between the highest v/f level and the optimal scaled v/f level. For the system energy optimization objective, this work tries to equalize task workloads of all clusters when executed at the highest v/f levels. To the best of our knowledge, this work is the only one aimed at achieving the optimal energy optimization of the overall cluster-based multi/many-core systems. However, this work fails to handle the assignment of task-dependent applications (*i.e.*, with task precedence constraints). Further efforts should be made for the global energy optimization of task-dependent applications on cluster-based multi/many-core systems.

2.3 Management Structure

As the number of cores in modern systems increases, run-time management strategies have to pay more attention to *management scalability*, which considers whether run-time management strategies have the potential to deal with a large number of applications in multi/many-core systems.

Management scalability has been studied in terms of management structures. The possible management structures are shown in Figure 2.12. In *centralized management* (see Figure 2.12.(a)), one manager monitors and controls applications execution in the whole system. Managing a system from one central point provides management simplicity, but it may also impose severe performance bottleneck and heavy computation burdens in many-core systems. To allow the management to be scalable with the size of the many-core system, *distributed management* and *hierarchical management* are proposed.

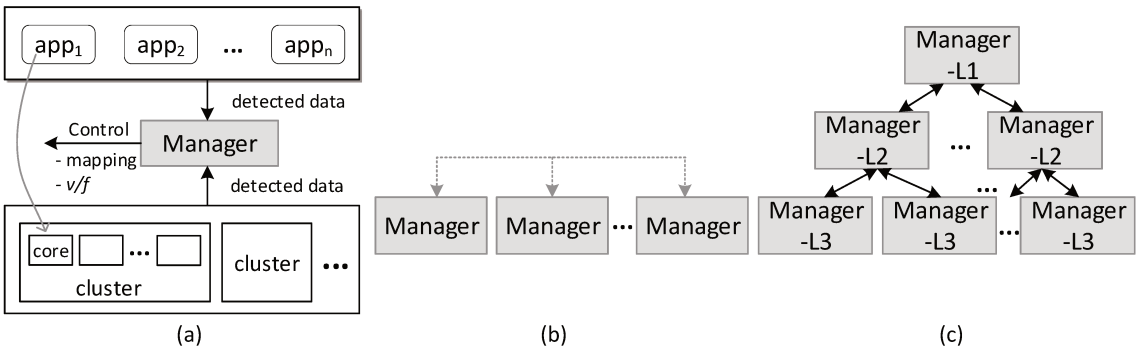

Figure 2.12: Run-time managers in different structures (a) centralized management, (b) distributed management and (c) hierarchical management.

2.3.1 Distributed Management

Distributed management divides the entire management problem into several sub-problems, each of which is handled by a manager. As shown in Figure 2.12 (b), the distributed managers

can be independent or can communicate with each other. For independent distributed managers, it is assumed that some management decisions (*e.g.*, applications and cores considered by each manager) are known at design-time. The distributed manager can be created individually for an application, or for a core, or a cluster, in order to achieve local optimization of a sub-system.

One Manager for One Application / One Core

In [52], each distributed manager is created for a certain active application in homogeneous multi/many-core systems. Each manager (or agent) is created to map an application onto cores for better performance. When a new application is active, its manager randomly selects a region (with some close available cores) on the chip to run the application. Then the application manager starts to communicate with another application manager over a short distance to request some resources. If the performance gain of the requesting application is greater than the performance loss of the answering application, some resources are released from one application to another. The resource bargain between different application managers can be applied over larger distances (among cores) to achieve a wider range of coverage.

The work in [53] presents a distributed management approach based on each core. A run-time heuristic algorithm is proposed to run on each processing core in a homogeneous multi-core system (*i.e.*, 9 cores). To reduce communication overheads, the heuristic migrates some tasks from one core to its neighbors based on its local workload. If no improvement is achieved from the migration, a larger neighborhood is considered. The algorithm stops when there is no more improvement after a certain number of repetitions (set by users).

The work in [54] creates a manager for each application and for each core. Among these works, the number of managers is highly dependent on the number of active applications or the number of cores. As the number of supported applications and the number of cores increase, creating a manager for each application or each core is not scalable.

One Manager for One Cluster

The authors of [55] create one manager for one cluster that executes multiple applications on several processing cores. It aims to optimize the communication energy of homogeneous multi/many-core systems. At system startup, it divides the platform into several fixed-size clusters and creates a manager for each cluster. During system execution, each cluster manager heuristically maps active applications within the cluster. When the resources are not sufficient, the cluster manager can borrow some resources from neighbor clusters. Thus the cluster sizes can change dynamically. The work in [56] presents another example that also supports dynamic size of cluster.

As previously discussed in Section 2.2.2, the works in [8, 12] respectively aims to achieve energy and power optimization within a cluster. These two works manage the executions of multiple applications on a cluster, assuming that the application-to-cluster allocation is known

in advance. It means that the two works create a distributed manager for a fixed size cluster in the Arm big.LITTLE systems. There is no communication between the distributed managers in the little cluster and the big cluster.

Compared to the distributed management approaches that are based on one application or one core, creating one manager for one cluster helps to reduce the number of managers. Since the main feature of distributed management is its local optimization of the system, the division of sub-systems (*e.g.*, in terms of an application, a core or a cluster) determines the scope of local optimization.

2.3.2 Hierarchical Management

Hierarchical management can provide management scalability with both local and global optimization. As shown in Figure 2.12 (c), managers are created to deal with a system at two or more levels of abstraction. According to [57], managers at different levels consider local optimization of each sub-system and global optimization of the overall system.

2-Level Hierarchical Management

In 2-level hierarchical management approaches, a global manager and several local managers are created to perform run-time management. The global manager not only serves as a communication center for different local managers but also highly determines the management quality of the entire system.

The work in [58] presents a 2-level hierarchical resource allocation framework on a heterogeneous platform, in which there are identical clusters on a chip (*i.e.*, homogeneous at the chip-level) and each cluster contains different core types (*i.e.*, heterogeneous at the cluster-level). In the management framework, a global manager monitors the system workload and assigns active applications to clusters for *workload balance* at the chip-level. An application is allocated to one cluster to reduce the task communication overhead between different clusters. At the cluster-level, a local manager is created to allocate application tasks to cores within each cluster. The local management strategy is based on some prepared mappings of each application. For multiple active applications in a cluster, the local strategy first merges the prepared mapping of each active application together and then iteratively migrates tasks between cores to minimize core usage variation among cores.

The work in [59] presents 2-level hierarchical managers to enable multi-objective optimization in a homogeneous many-core platform, which has multiple clusters and supports per-core DVFS. The managers aim to reduce energy consumption, improve application performance and guarantee the power constraint. At the chip-level, the global manager verifies the power and resource requirements of a new active application and then chooses a cluster for the application. The global manager can also dynamically change the operation mode of a cluster according to the workload behavior. A cluster is set to *energy mode* or *performance mode* depending

on whether the power budget is violated or not. Then, at cluster-level, a local manager is responsible for mapping or remapping tasks and setting v/f level of each core according to the operation mode.

More-Level Hierarchical Management

The work in [60] presents hierarchical organized run-time controllers to deal with application dynamism and architecture failures (temporary or permanent) of many-core systems with multiple clusters. The created controllers aim to deal with the behavioral or fault events in three different levels, which are core-level, cluster-level, and chip-level. The events that can not be handled at a low level can be delivered to a higher level.

As previously discussed in Section 2.2.1, the work of [49] applies dynamic mapping and DVFS separately to manage the ARM big.LITTLE system under restricted power budget. Here, we discuss the management structure of this work. This work presents a 3-level hierarchical management framework, which consists of different levels (*i.e.*, chip-level, cluster-level, task-level) of controllers. The chip-level power allocator triggers the cluster frequencies and the quality of service (QoS) of the tasks. The per-cluster DVFS controller sets the cluster frequencies. The per-task QoS controller sets the task performance constraint, based on which per-task resource control determine resource allocation. There is a load balancer and migrator at the cluster-level, which migrates tasks between big and little cluster according to the performance requirements.

Compared to distributed management, hierarchical management has more flexibility due to its capability of local and global optimizations. Different optimization targets can be set to managers at different levels. The difficulty of hierarchical management is how to coordinate management between different management levels [57].

2.4 Run-Time Management Strategy Evaluation at System-Level

In the state-of-the-art of system-level modeling and simulation approaches, a system model is captured according to Y-chart design methodology [9, 61, 62], where application models and a platform model are built independently and further combined by mapping rules. As illustrated in Figure 2.13, application models capture the functional behavior of applications, while the platform model describes the hardware resources and hardware performance characteristics. After the application models are mapped onto and then simulated with the platform model, the platform model accepts the computation and communication activities of applications as workloads [62]. As a consequence, non-functional characteristics of applications can be estimated under different situations. The resulting performance may lead to the improvement of platforms, the adaptation of applications or the modification of mapping strategies. The

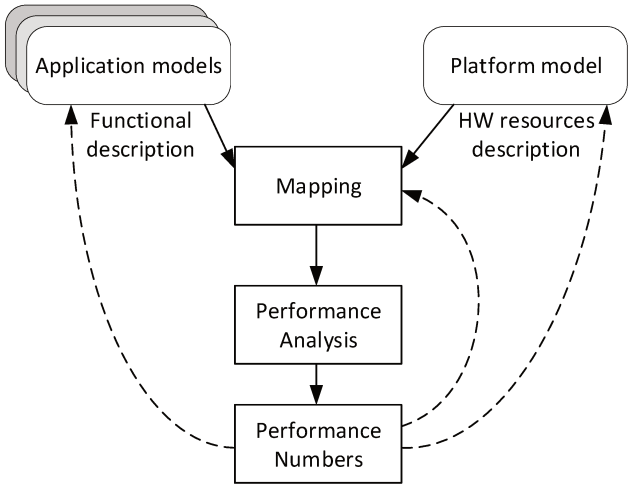

Figure 2.13: Y-chart-based design methodology [9].

process of system modeling and simulation are performed at a high-level of abstraction, which minimizes modeling effort and optimizes simulation speed [61].

To the best of our knowledge, only two related works support dynamism in system-level simulation. As previously discussed, the work of [10] presents a run-time mapping strategy to deal with multiple applications dynamically executed on a heterogeneous multi-core platform. This work evaluates the run-time mapping strategy on an extended Sesame system-level modeling and simulation framework. Figure 2.14 illustrates the extended Sesame framework, which is based on the trace-driven simulation approach [62, 63]. Each application model records its action by a set of event traces (*i.e.* computation and communication events). To simulate the performance consequences of application events, a platform model is parameterized by an event table containing operation latencies. The extended Sesame introduces three modules (in gray in

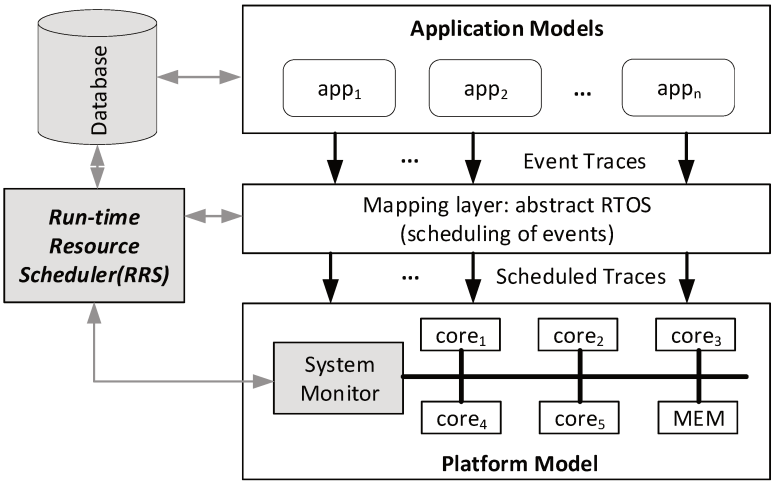

Figure 2.14: Extended Sesame framework for run-time resource scheduling [10].

Figure 2.14) to support the simulation of run-time mapping strategies. The *database* stores some prepared mappings that are optimized at design-time, and these prepared mappings are used to establish mappings for active applications at run-time. The *system monitor* aims to identify the current states of application workloads and platform resources. The *Run-time Resource Scheduler (RRS)* is especially introduced to control the mapping of active applications in each simulated use-case. During system simulation, RRS dynamically dispatches the event traces to an architecture model at the intermediate mapping layer. During event dispatching, some synchronizations have to be done to ensure the availability of the allocated hardware resources. Our simulation approach aims to avoid such model synchronization by computing the instants when platform resources are used.

In [11], an extension of Intel CoFluent Studio is proposed to support dynamic application mapping but this proposal is not currently supported in the available tool. Figure 2.15 presents the extended framework, where the newly introduced modules are highlighted in gray. The introduced modules have similar functions as those ones in the previously discussed *Sesame* framework. The *database* stores some design-time mappings. The *Performance-Aware Supervisor (PAS)* has two missions. First, *PAS* identifies the system violations (*e.g.*, violations of timing constraint or power budget) by some agents, which are added for each application and each platform component to monitor the non-functional properties (*e.g.*, latency, resource load, and power consumption. etc.). Secondly, based on the monitored information, it dynamically changes the allocation of tasks on platform resources during system simulation by a certain run-time mapping strategy. The run-time mapping is fulfilled by additional SystemC code that is instrumented in the framework. In contrast to this approach, we aim to implement a new simulation approach in Intel CoFluent Studio without any modification of the used framework, making our proposition portable to other simulations environments.

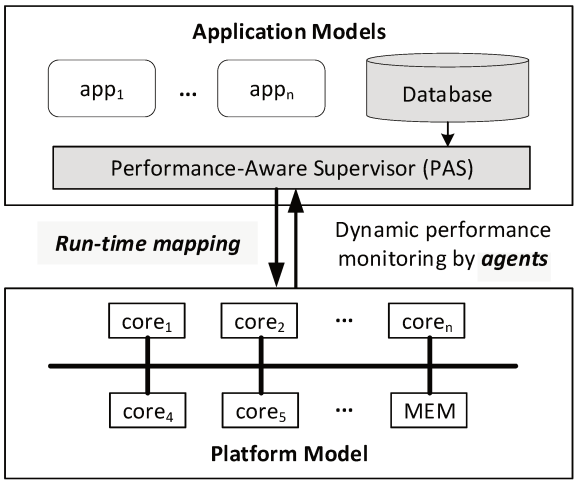

Figure 2.15: Extended CoFluent framework for run-time resource scheduling [11].

2.5 Summary and Discussion

This dissertation focuses on managing the energy efficiency of multiple applications executed dynamically on cluster-based multi/many-core platforms. In this chapter, related works are presented. The presented literature is summarized in Table 2.1.

Firstly, we discussed the existing works about dynamic task mapping, a necessary technique to deal with application execution dynamism. We concentrate on hybrid mapping strategies, which realize dynamic mapping based on some design-time prepared mappings to reduce run-time computation burden. Generally, to reduce storage space, some optimized mappings are prepared for each application (highlighted in gray in Table 2.1) at design-time. At run-time, application mappings can be optimized independently or holistically. Our work focuses on holistic optimization, due to its capability of overall optimization for all active applications. We also apply independent optimization to perform application-to-cluster assignment in the overall system, because independent optimization has better feasibility and lower complexity.

Secondly, we discussed the existing approaches that apply dynamic mapping and DVFS techniques on cluster-based multi/many-core systems. Mapping and v/f configurations can be applied independently or coordinately, based on whether the mutual influence between the two techniques is predicted or evaluated during the decision-making process. Applying the two techniques coordinately can achieve better management results. However, more computation efforts have to be paid at run-time. The management problem becomes even more complex when hybrid mapping targets holistic optimization for all applications. Our work addresses the difficulties of coordinating hybrid mapping and DVFS to obtain run-time mappings for task-dependent applications with optimized cluster v/f levels. In contrast to the previous works, we aim at reducing strategy complexity of local optimization, and exploring management strategies to achieve global optimization of systems with different numbers of clusters (*e.g.*, more than 2).

Thirdly, we discussed different management structures that can be used to realize hybrid mapping and DVFS techniques. Our work considers distributed and hierarchical structures to allow our management strategies to be scalable in large systems where many applications can be executed simultaneously on a large number of cores. On one hand, for cluster-based multi/many-core systems, we adopt the existing distributed management structure where a local manager is created for each cluster. In this dissertation, we propose a new local management strategy to achieve local optimization in one cluster, assuming the same strategy is applied in every cluster. Compared with existing related work, the proposed local management strategy requires fewer search iterations to achieve energy efficiency within a cluster. On the other hand, for global optimization in the overall cluster-based system, we apply the existing hierarchical management structure where the global manager is created at chip-level and local managers are created at cluster-level. Unlike most existing hierarchical management approaches, our work considers hybrid mapping and per-cluster DVFS holistically. The proposed approach can be scalable to homogeneous and heterogeneous cluster-based platforms with different numbers of

clusters and different numbers of cores in each cluster.

From one use-case to another, our proposed run-time management strategies could update application mapping and cluster frequency configurations for energy optimization. During the system reconfiguration process, some time and energy would be spent to allow task/application migrations from one core to another (or from one cluster to another). Thus it requires weigh reconfiguration costs and benefits to make a reasonable run-time decision. Notice that system reconfiguration costs are highly dependent on the current use-case duration [64]. The work of [64] predicts use-case duration based on historical records. It stores 3 history samples (use-case duration) for each use-case and computes the probabilities of possible predictions. For a new use-case, this work checks its matched history pattern and then sums the probabilities of some promising predictions (where use-case duration is large enough) together. If the sum probability value is large than a predefined value, this work performs migration. For simplicity, our work assumes that each use-case executes long enough, and the system reconfiguration costs can be neglected compared to the reconfiguration benefits. Use-case duration prediction would be addressed in future work.

Finally, we discussed the state-of-the-art system-level modeling and simulation approaches that support the evaluation of run-time management strategies. These approaches use some design-time prepared mappings to guide run-time mapping simulation. This dissertation work presents a new system-level simulation approach, which is also based on design-time prepared data. Compared to the existing trace-driven simulation approach, our approach does not dispatch trace events and avoids model synchronization by computing the instants when application tasks are run on platform resources. Besides, our approach can be implemented without any modification of the used framework.

Table 2.1: Comparison of state-of-the-art run-time management approaches

Ref	Platform	Dynamic task mapping		Apply with DVFS	Mechanism
		Design-time preparation	Run-time configuration		
[46]	Generic heterogeneous [2]	Use-case-based	Apply optimized mapping		Centralized
[47]	Heterogeneous cluster-based	Use-case-based	Apply optimized mapping		Centralized
[60]	Homogeneous cluster-based	Use-case-based	Apply optimized mapping		Hierarchical (3-level)
[7]	ARM big.LITTLE	Application-based	Optimize apps individually	Per-cluster DVFS (separately)	Distributed (cluster)
[13]	Homogeneous	Application-based	Optimize apps individually		Centralized
[48]	Homogeneous	Application-based	Optimize apps individually		Centralized
[3]	Generic heterogeneous	Application-based	Optimize apps holistically		Centralized
[8]	ARM big.LITTLE	Application-based	Optimized apps holistically	Per-cluster DVFS (coordinately)	Distributed (cluster)
[14]	Homogeneous	Application-based	Optimize apps holistically		Centralized
[34]	ARM big.LITTLE	Application-based	Optimize apps holistically	Per-cluster DVFS (separately)	Centralized
[10]	Generic heterogeneous	Application-based	Optimize apps holistically		Centralized
[58]	Special heterogeneous [3]	Application-based	Optimized apps holistically		Hierarchical (2-level)
[4]	Heterogeneous cluster-based	On-the-fly mapping		Per-cluster DVFS (separately)	Centralized
[5]	ARM big.LITTLE	On-the-fly mapping		Per-cluster DVFS (separately)	Centralized
[6]	ARM big.LITTLE	On-the-fly mapping		Per-cluster DVFS (coordinately)	Centralized
[29]	Heterogeneous cluster-based	On-the-fly mapping		Per-cluster DVFS (coordinately)	Centralized
[51]	ARM big.LITTLE	On-the-fly mapping		Per-cluster DVFS (separately)	Centralized
[49]	ARM big.LITTLE	On-the-fly mapping		Per-cluster DVFS (separately)	Hierarchical (3-level)
[59]	Homogeneous	On-the-fly mapping		Per-core DVFS (separately)	Hierarchical (2-level)
[52]	Homogeneous	On-the-fly mapping			Distributed (app)
[53]	Homogeneous	On-the-fly mapping			Distributed (core)
[55]	Homogeneous	On-the-fly mapping			Distributed (cluster)
[56]	Homogeneous	On-the-fly mapping			Distributed (cluster)

[2] Generic Heterogeneous: a platform contains different type of cores showing different power/performance characteristics (*e.g.*, GPP, DSP).

[3] Special Heterogeneous: a platform has the same clusters (*i.e.*, homogeneous) in the system and different core types ((*i.e.*, heterogeneous) within a cluster.

Chapter 3

System Models

Contents

This chapter presents the application, platform, application mapping and power and energy models. The models and notations defined in this chapter are used throughout the dissertation work.

3.1 Application Models

In this dissertation, we target data-flow applications (*i.e.*, task-dependent applications), where the output of one task might be the input of other tasks [65]. Some examples include multimedia and Digital Signal Processing (DSP) applications. Figure 3.1 gives an illustration of an application, denoted by app_i. The application app_i consists of a set of H computation tasks (or nodes): $T_{app_i} = \{t_{1,i}, t_{2,i}, \cdots, t_{H,i}\}$ and a set of G communication edges (or arcs): $E_{app_i} = \{e_{1,i}, e_{2,i}, \cdots, e_{G,i}\}$ representing dependencies among the tasks. Task and edge in app_i are respectively indexed by $t_{h,i}$ and $e_{g,i}$. This work focuses on periodic real-time data-flow applications and each application has a period $Period_{app_i}$, denoting the application execution deadline. The application execution time shall be within its corresponding period time.

In the scope of this work, Synchronous Data Flow (SDF) semantics [66] is used to capture the data-flow activity of applications by specifying the number of data samples (or tokens)

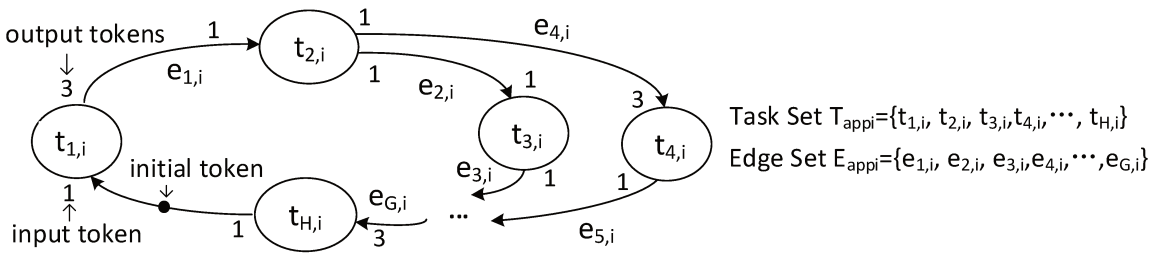

Figure 3.1: Application model of app_i.

produced and consumed by each task. As shown in Figure 3.1, input tokens define the number of tokens that are read from the edge before executing a task and the output tokens define the number of tokens that are written through the edge after the task execution. The reason for choosing SDF is that it is a commonly adopted model of computation for data-flow applications, offering both good expressiveness and analysis capability. Our work is not limited to this specific model of computation and it could be extended to other data-flow models.

Particularly, Figure 3.2 gives the general SDF descriptions of H.263 decoder, H.263 encoder and JPEG decoder according to *SDF3* [67]. These three applications are typical multimedia applications that are widely used as benchmarks in works such as [48, 68, 69]. In our work, we will use these three applications to derive other applications with different numbers of tokens and different period constraints. The derived applications will serve as representatives of different computation and communication activities in the experimental evaluations of Chapters 4, 5 and 6.

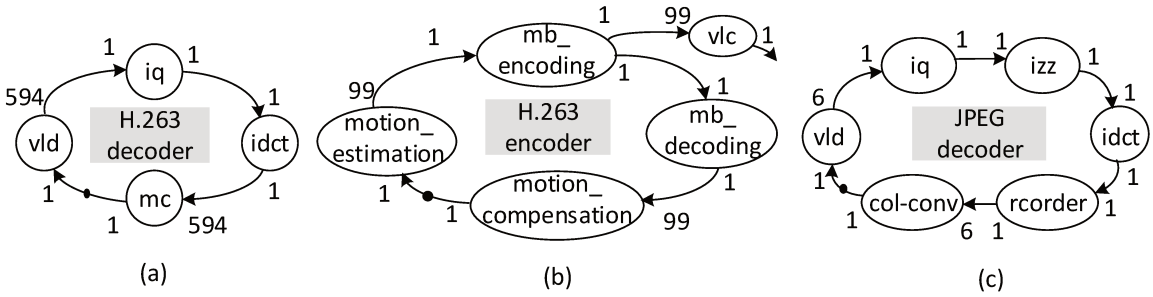

Figure 3.2: SDF descriptions of (a) H.263 decoder, (b) H.263 encoder and (c) JPEG decoder.

As previously discussed in Section 1.1.5, this work considers application execution dynamism in which use-cases can change over time. As previously defined, a use-case refers to a set of simultaneously active applications [30]. Here, we denote a use-case by $u_m = \{app_1, app_2, \cdots, app_I\}$, where I is the total number of active applications in the considered

use-case. Different use-cases might exist due to different combinations of multiple applications. The execution duration of each use-case depends on application activities. Without loss of generality, this work makes the same assumption as in [10] that each use-case runs for a long time, to justify run-time configurations (*e.g.*, a new mapping or v/f settings).

3.2 Platform Model

As introduced in Chapter 1, this dissertation focuses on cluster-based multi/many-core platforms. Figure 3.3 shows the platform model, which consists of multiple clusters associated with a shared memory and communication resources. Let J be the total number of clusters and each cluster is indexed by $cluster_j$. Each cluster is composed of multiple or many cores. The number of cores in $cluster_j$ is denoted by N_j, and different clusters can have different numbers of cores. It is assumed that the cores in each cluster are of the same type, while the core type can be different from one cluster to another.

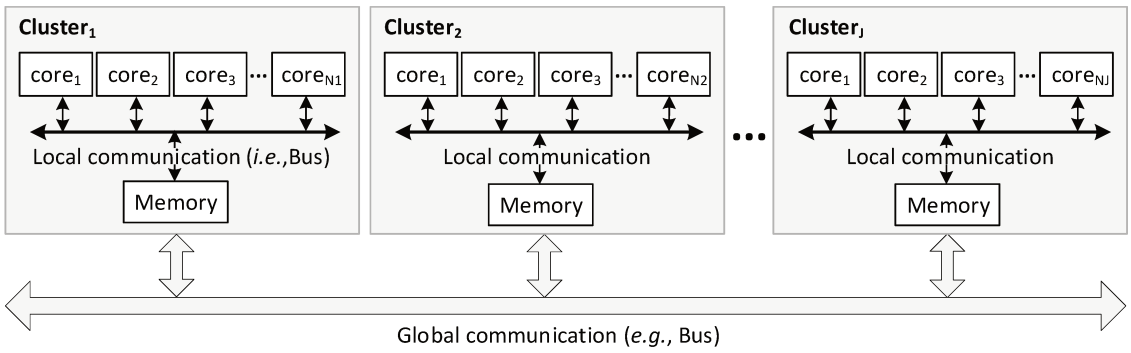

Figure 3.3: Platform model of cluster-based multi/many-core systems.

The heterogeneity of the cores among clusters can lead to different characteristics of latency and power consumption. To illustrate these differences, the core type can be characterized by a performance ratio and a power ratio normalized against a reference cluster $cluster_r$. According to [25], these ratios represent the performance and power improvements (or degradation) to perform the same system executions as compared to the reference cluster. For a given $cluster_j$, its performance and power ratios are indexed by R_j^{perf} and R_j^{power} respectively.

In addition, the considered platform supports per-cluster DVFS, each cluster being a voltage/frequency island (VFI) [70]. All the cores within a cluster share the same voltage/frequency (v/f) level, while each cluster has its own v/f ranges. The available discrete frequency levels of $cluster_j$ are denoted as $\{f_{j,1}, f_{j,2}, \ldots, f_{j,max}\}$ and operating voltages are adapted to the frequency settings. The relationship between operating voltage and frequency will be further illustrated in Section 3.4.

Table 3.1 presents some examples of cluster-based multi-core/many-core platforms. *Intel* Single-chip Cloud Computer (SCC) [19] and *Kalray MPPA2®-256* [20] have many homogeneous clusters (*i.e.*, more than 10). Their homogeneous clusters support the same frequency range. On the other hand, Exynos 5 Octa [21] and MediaTekHelio [23] have several heterogeneous clusters. Their clusters are based on the ARM Cortex family and different clusters support different frequency ranges.

Table 3.1: Examples of cluster-based multi/many-core platforms

Examples	**Architecture**	**Nb of clusters:** J	**Cluster description with supported operating frequency ranges**
SCC [19]	Homogeneous	24	2 Pentium cores: $125MHz \sim 1GHz$
MPPA2®-256 [20]	Homogeneous	16	16 RISC cores: $600MHz \sim 800MHz$
Exynos 5 Octa [21]	Heterogeneous	2	4-core ARM Cortex-A7: $0.2GHz \sim 1.4GHz$ 4-core ARM Cortex-A15: $0.2GHz \sim 2.0GHz$
MediaTekHelio X30 (MT6799) [23]	Heterogeneous	3	2-core ARM Cortex-A73: up to $2.6GHz$ 4-core ARM Cortex-A53: up to $2.2GHz$ 2-core ARM Cortex-A35: up to $1.9GHz$

3.3 Mapping Model

As previously defined in Section 1.1.5, task mapping refers to the allocation of application tasks on platform resources and the execution order (*i.e.*, scheduling) of tasks on a given core. When applications are mapped on platform resources, their computation and communication activities can be fulfilled after a certain time of execution. The time used to finish computation activities of a task ($t_{h,i}$) is defined as computation time ($CompTime_{h,i}$), while the time used to finish communication activities between dependent tasks via an edge ($e_{g,i}$) is defined as communication time ($CommTime_{g,i}$). $CompTime_{h,i}$ and $CommTime_{g,i}$ can be different due to different processed data, mapping strategies and platform configurations (processing element, v/f level, $\cdots$). This work holds the same assumption as [10], that is, communication time within a core is very short and can be neglected.

In this work, we characterize a mapping by an execution trace, which comprises a set of instants defining the start time (x_s) and the end time (x_e) of each task when executed on platform resources. For a given task, $x_{s\_t_{h,i}}(k)$ and $x_{e\_t_{h,i}}(k)$ refer to the k^{th} start and end instances of the task $t_{h,i}$ respectively. Figure 3.4 gives two examples of execution traces for app_1 mapped on two cores and four cores. In these examples, $t_{2,1}$ and $t_{3,1}$ are executed three times at each iteration. Due to the different mappings between Figure 3.4 (a) and (b), their obtained instants (*i.e.*, from $x_{s\_t_{1,1}}(1)$ to $x_{e\_t_{4,1}}(1)$) are different the change of task execution orders. Moreover, since

different platform configurations (processing element, v/f level, $\cdots$) can change $CompTime_{h,i}$ and $CommTime_{g,i}$, the instants of execution traces will change accordingly.

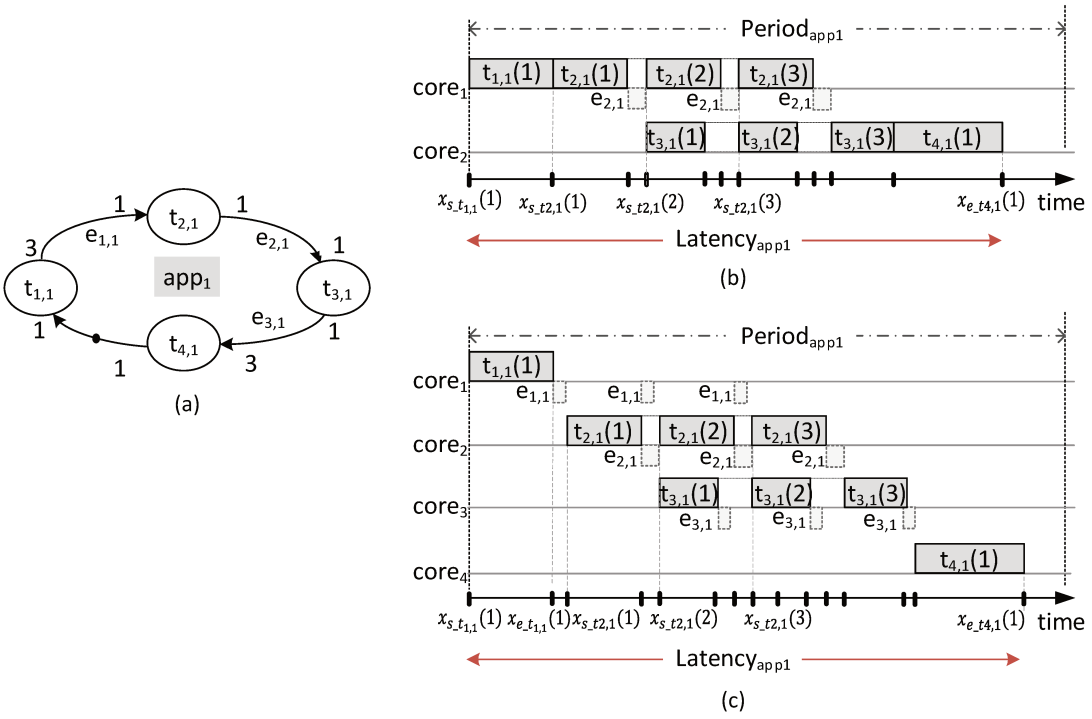

Figure 3.4: (a) SDF description of app_i. Examples of execution traces of app_1 mapped on (b) two cores and (c) four cores.

The application latency, denoted by $Latency_{app_i}$, refers to the execution time of an application from its first input to the last instant (*i.e.*, $x_{e\_t_{4,1}}$ in Figure 3.4) within one period. To guarantee timing constraints are met, application latency should be smaller than its period (*i.e.*, $Latency_{app_i} <= Period_{app_i}$).

3.4 Models for Energy Efficiency Evaluation

The section presents the models to evaluate the energy efficiency of executing applications on platform resources. Energy efficiency can be characterized by *energy consumption* or *average power consumption* in a certain period of time. In this work, we focus on the optimization of *average dynamic power consumption* within a *hyper-period*. According to the definition of [4], *hyper-period* refers to the Least Common Multiple (LCM) among periods of the considered periodic applications. In this work, the *average dynamic power consumption* is optimized through dynamic task mapping and DVFS. Shutting down some clusters/cores to reduce *static*

power/energy is beyond the scope of this work. The energy and the average power consumption mentioned in the following refer to the dynamic part.

For multiple applications executed on a cluster-based multi/many-core platform, the system average power (P_{sys}^{avg}) can be expressed as the sum of the average power of all active applications in all clusters as follows.

$$P_{sys}^{avg} = \sum_{i=1}^{I} \sum_{j=1}^{J} P_{app_i}^{avg}(cluster_j, f_j) \tag{3.1}$$

where I and J are the total number of active applications and the total number of clusters, respectively. $P_{app_i}^{avg}$ is the average power of app_i, and the power value depends on the cluster (*e.g.*, $cluster_j$) where the application is executing and the cluster frequency configuration (*e.g.*,f_j).

For an application, its $P_{app_i}^{avg}$ within its application period ($period_{app_i}$) can be computed as the amount of energy (E_{app_i}) consumed in a unit of time as follows.

$$P_{app_i}^{avg}(cluster_j, f_j) = \frac{E_{app_i}(cluster_j, f_j)}{Period_{app_i}} \tag{3.2}$$

The energy of an application (E_{app_i}) can be computed as the sum of computation energy ($E_{app_i}^{comp}$) and communication energy ($E_{app_i}^{comm}$) as expressed in Eq.(3.3).

$$E_{app_i}(cluster_j, f_j) = E_{app_i}^{comm}(cluster_j, f_j) + E_{app_i}^{comp}(cluster_j, f_j) \tag{3.3}$$

In this work, we focus on data-flow applications like multimedia applications. We assume that the communication cost (*e.g.*, time and power) of an application is *much smaller* than its computation cost. Chapter 4 will use *communication energy ratio* (R_{comm}) to indicate the small ratio between communication energy and computation energy within a period for each application mapping. On the other hand, the $E_{app_i}^{comp}$ of an application can be estimated as the sum of energy consumed by all tasks, which can be expressed into Eq.(3.4).

$$E_{app_i}^{comp}(cluster_j, f_j) = \sum_{h=1}^{H} E_{h,i}(cluster_j, f_j) \tag{3.4}$$

where $E_{h,i}$ refers to the computation energy of $t_{h,i}$. As the energy of a task is the integration of its power ($P_{h,j}$) overtime, $E_{h,i}$ can be further estimated as follows.

$$E_{h,i}(cluster_j, f_j) = P_{h,i}(cluster_j, f_j) \times CompTime_{h,i}(cluster_j, f_j) \tag{3.5}$$

For CMOS circuits, the works in [71–73] define the dynamic power model by the square of the voltage (v) and the frequency(f), as shown in Eq.(3.6).

$$P = \epsilon \times v^2 \times f \tag{3.6}$$

where ϵ is a constant coefficient that depends on the technology used to manufacture the circuits. Furthermore, according to [35, 72, 73], it exists an approximate relationship between the operating frequency and supply voltage, as shown in Eq.(3.7).

$$f = \frac{(V - V_{th})^{\alpha}}{K \times L_d} \tag{3.7}$$

where V_{th} is the threshold voltage, L_d is the logic depth, K is a constant, while α is a technology dependent parameter. Based on Eq.(3.6) and Eq.(3.7), the dynamic power can be expressed as a polynomial of frequency of degree λ (*i.e.*, f^{λ}) [8]. λ is generally set to 3 in related works [28, 32, 33, 73] due to that fact that they assume there is an approximate linear proportional relationship between frequency and voltage.

Upon convenience, this work reuses the power model (Eq.(3.8)) proposed in [8], which considers the dynamic power of an application mapped on the Exynos.5422 [21] Arm big.Little cluster-based multi-core platform. Here, the dynamic power is estimated by the cubic of frequency (*i.e.*, f^3).

$$P = \xi \times f^3 \tag{3.8}$$

where ξ is a coefficient that is dependent on the task and the allocated core type.

To describe the evolution of computation time with operating frequency, we use the traditional performance model [4, 32, 72] as shown in Eq.(3.9). W is the total number of execution cycles, which can be understood as the amount of work that has to be done. W can be known at a reference frequency (f_0) [8]. f_0 can be one any frequency level that is commonly supported in all clusters. Eq.(3.9) will be further verified in the next section.

$$\begin{aligned} CompTime(f) &\approx \frac{W}{f} \\ &\approx \frac{CompTime(f_0) \times f_0}{f} \end{aligned} \tag{3.9}$$

From Eq.(3.8) and Eq.(3.9), the dynamic computation energy in Eq.(3.5) can be written into Eq.(3.10). Let $\xi_{h,i,j}$ be the power coefficient of $t_{h,i}$ executed on $cluster_j$. Note that now the energy equation has f_j^2 instead of f_j^3.

$$E_{h,i}(cluster_j, f_j) = \xi_{h,i,j} \times CompTime_{h,i}(cluster_j, f_0) \times f_0 \times f_j^2 \tag{3.10}$$

Moreover, the work of [25] summarized the performance and power consumption ratios based on the publicly available information of ARM-cortex processors. Let R_j^{perf} and R_j^{power} be the performance/power ratios of $t_{h,i}$ executed on $cluster_j$ respectively, while $cluster_r$ is

defined as a reference cluster.

$$R_j^{perf} = \frac{CompTime_{h,i}(cluster_j, f_0)}{CompTime_{h,i}(cluster_r, f_0)} \tag{3.11}$$

$$\begin{aligned} R_j^{power} &= \frac{P_{h,i}(cluster_j, f_0)}{P_{h,j}(cluster_r, f_0)} \\ &= \frac{\xi_{h,i,j}}{\xi_{h,j,r}} \end{aligned} \tag{3.12}$$

Based on Eq.(3.10), (3.11) and (3.12), the average dynamic energy of a task (*e.g.*, $t_{h,i}$) can be further written into Eq.(3.13).

$$E_{h,i}(cluster_j, f_j) = R_j^{power} \times \xi_{h,i,r} \times R_j^{perf} \times CompTime_{h,i,r}(f_0) \times f_0 \times f_j^2 \tag{3.13}$$

As a consequence, the system average power (P_{sys}^{avg}) in Eq.(3.1) can be further written into Eq.(3.14), if communication energy is neglected.

$$\begin{aligned} P_{sys}^{avg} &= \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{E_{app_i}(cluster_j, f_j)}{Period_{app_i}} \\ &= \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{\sum_{h=1}^{H} E_{h,i}(cluster_j, f_j)}{Period_{app_i}} \\ &= \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{\sum_{h=1}^{H} R_j^{power} \times \xi_{h,i,r} \times R_j^{perf} \times CompTime_{h,i,r}(f_0) \times f_0 \times f_j^2}{Period_{app_i}} \end{aligned} \tag{3.14}$$

Eq.(3.14) reveals that system average power (P_{sys}^{avg}) is highly dependent on the hardware features (*e.g.*, R_j^{power}, R_j^{perf} and f_j) and application characteristics (*e.g.*, $\xi_{h,i,r}$, $CompTime_{h,i,r}$, and $Period_{app_i}$).

3.5 Model Validation on ARM big.LITTLE platforms

In this section, ODROID XU3 board [50] consisting of a Samsung Exynos 5422 [21] ARM big.LITTLE clusters is used as experimental platform to verify our applied models. The verified models include v-f model (*i.e.*, Eq.(3.7)), power model (*i.e.*, Eq.(3.8)) and performance model (*i.e.*, Eq.(3.9)). The ODROID XU3 board embeds *INA231 current-shunt and power sensors* [74] to allow the measurement of the instant current and power consumption in the little cluster and the big cluster. The function $/clock()$ provided by linux can be used to measure application execution time.

Our experiment uses *matrix multiplication* as a representative application. We map the application onto different numbers of cores within a cluster. There is no data communication from one core to another. The application cost (*e.g.*, time and power) is mainly spent on computation activities and memory accesses. Figure 3.5 shows the voltage-frequency evolution of *matrix multiplication* application mapped on 1, 2 and 4 cores. The measurement results are given for both clusters. During the measurement, we increment cluster frequency levels. Based on these frequencies, voltage information can be read from the *INA231* sensor embedded in each cluster.

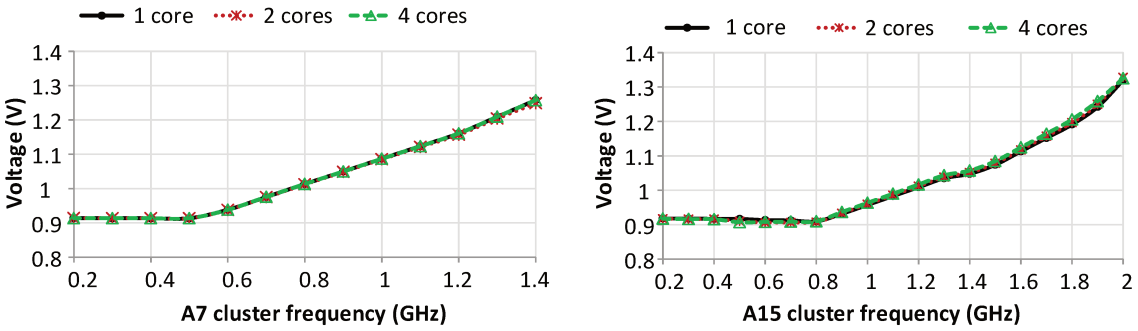

Figure 3.5: Dependence between voltage and frequency for *matrix multiplication* executed on 1, 2 and 4 cores in (a) the little cluster and (b) the big cluster.

From Figure 3.5, we can observe the voltage and frequency relationship is not dependent on application workloads (*e.g.*, due to different mappings). The cluster voltage increases linearly with frequency within a certain range (*i.e.*, from $0.5GHz$ to $1.4GHz$ in the little cluster, from $0.8GHz$ to $2.0GHz$ in the big cluster). This confirms the assumption made for Eq.(3.7) that there is an *approximately* linear proportional relationship between voltage and frequency in a specific frequency range.

In addition, Figure 3.6 presents the average dynamic power consumption and the measured latency according to increasing cluster frequency for *matrix multiplication* application mapped on 1, 2 and 4 cores in the two different clusters. Part (a) of Figure 3.6 shows the average dynamic power consumption increases with frequency. The depicted power in the figure is composed of the average dynamic power of the application and of memory accesses, which are obtained by excluding the "System Default Power" from the sensor values. "System Default Power" refers to the measured power of the cluster without any mapped application. The measured power indicates that an application mapped to more cores can achieve better performance, which is consistent with the observation in [75]. Part (b) of the figure shows the application latency decreases as frequency increases. The measured latency mainly consists of task computation time and memory access time. The measurement indicates that better application performance can be achieved by using more cores.

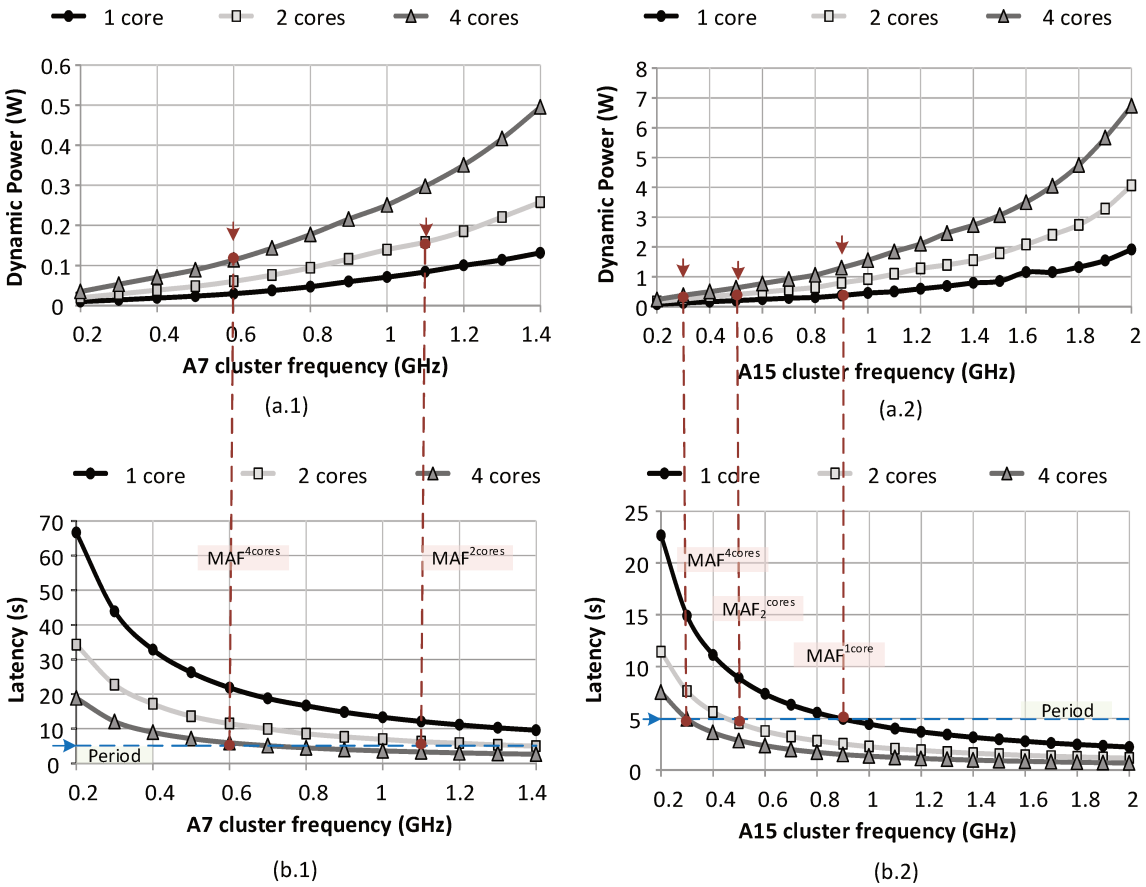

Figure 3.6: Evolution of (a) latency and (b)average dynamic power consumption based on frequency for *matrix multiplication* executed on 1, 2 and 4 cores in the little and big clusters.

Table 3.2: Fitting results for the models and measurements

Fitting Equation			Cortex-A7			Cortex-A15		
			1 core	2 cores	4 cores	1 core	2 cores	4 cores
Extended Eq.(3.8)	$Power(f) = \xi \times f^3 + const_2$	R-squared	0.977	0.978	0.982	0.987	0.989	0.990
Extended Eq.(3.9)	$Latency(f) = W/f + const_1$	R-squared	0.999	0.999	0.999	0.999	0.999	0.999
		W value	13.260	6.819	3.753	4.521	2.279	1.457
	$CompTime(f_0) \times f_0$ value[4]		13.284	6.925	3.615	4.436	2.266	1.375

[4] $CompTime(f_0) \times f_0$ is the average value obtained when f_0 is set to different frequencies.

We try to fit the measured power and latency of the application with our models. For the *matrix multiplication* application, we can use Eq.(3.8) and Eq.(3.9) to estimate the computation power and computation time. As inspired by [8] which uses a constant value to present the memory access cost, we add a constant value to each of the two equations to represent the memory access power and memory access time. Table 3.2 shows the fitting results for the

models and measurements (in Figure 3.6) and the R-squared [5] values are highlighted in gray. The obtained R-squared values are close to 1, indicating that our models fit well with the measurements. Particularly for the fitting of the extended Eq.(3.9), the fitting W values are close to the values of $CompTime(f_0) \times f_0$. This confirms that we could approximately predict task computation time based on Eq.(3.9).

Moreover, since this dissertation work considers periodic applications, we arbitrarily set a period (*i.e.*, timing constraint) for the application. The application period ($5s$) is marked by a blue dashed line in Figure 3.6 (a.2) and (b.2). As can be seen, application mappings using different numbers of cores require different frequencies to meet the timing constraint. The minimum required frequency is defined as *Minimum Allowed Frequency* (MAF) in this work. For the example in the A7 cluster (see part (a.1) and (a.2)), MAFs are marked (in red dashed lines) for the application mappings using 2 cores (MAF^{2cores}) and 4 cores (MAF^{4cores}). MAF^{4cores} is lower than MAF^{2cores} because the performance of the 4-core mapping is better. On the other hand, the 1-core mapping cannot satisfy the timing constraint at any frequency level, and thus no MAF is marked for this mapping. Similar observations can be found in the A15 cluster (see part (b.1) and (b.2)). These observations suggest that the MAF of application mappings using more cores may be lower than application mappings using fewer cores, but an application mapping can have different MAF values in different heterogeneous clusters (*e.g*, compare MAF^{4cores} in A7 and A15). The MAF parameter will be used in Chapter 4 and Chapter 5 to characterize each design-time application mapping.

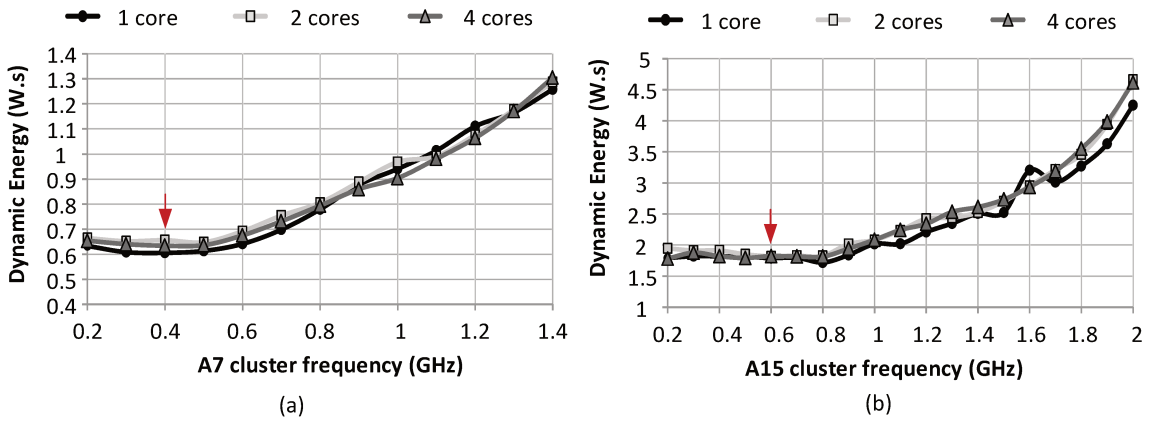

Figure 3.7: Evolution of dynamic energy consumption based on frequency for *matrix multiplication* executed on 1, 2 and 4 cores in the little and big clusters.

Finally, we compare the evolution of dynamic energy consumption of the application mapped on different numbers of cores. Figure 3.7 gives the energy evolution in the little cluster and the big cluster. The depicted values are obtained by multiplying the power values and

[5] R-square refers to coefficient of determination. The closer the value of R-square is to 1, the better the fit between the measured data and the model used.

latency values together at different frequencies (from Figure 3.6). We achieve the same energy evolution as [4]. It can be observed that there exists a *critical frequency* (f_{crit})" [4, 76] which minimizes the energy consumption of application execution. The arrow in figure (a) and (b) highlight the f_{crit} for the little cluster and the big clusters respectively. The f_{crit} values do not depend on the application, but depends on the platform characteristics. The f_{crit} exists due to the fact that cluster voltage does not decrease with frequency (see Figure 3.5), which makes energy cannot keep decreasing with frequency. According to [4], it should avoid executing an application at frequencies below f_{crit} for better system energy efficiency. In the scope of our work, we assume that scaled frequencies are above f_{crit}.

3.6 Summary

In this chapter, we presented the system models that are used throughout this dissertation. We consider data-flow applications (*i.e.*, with task dependence). Each application is captured by a SDF description. The targeted cluster-based multi/many-core platforms support per-cluster DVFS, and each cluster can be set to discrete frequency levels. The mapping of applications executed on platform resources is characterized by an execution trace, which defines the start instant and end instant of task executions. We also presented the power and energy models that are required for the evaluation of energy efficiency. Our used models are verified by some measurements on the ODROID XU3 board with the ARM big.LITTLE architecture. The measurement results show the same trend as the measurements in existing works. The power and energy models presented in this chapter will be used to evaluate our proposed management strategies in Chapter 4 and 5.

Our work verifies the power and energy models in ARM-based platforms by using the *matrix multiplication* application. Future work could perform more extensive validation experiments for different applications (*e.g.*, computation-intensive, communication-intensive workloads) and for different platforms (*e.g.*,x86, RISC-V). Moreover, our power/energy models are built for each independent application. These models can be inaccurate when multiple applications executing simultaneously on the same platform. This is because the fixed coefficients of the models cannot capture run-time workload variations. The establishment of run-time power/energy models can be addressed in our future work.

Chapter 4

Run-Time Management for Local Optimization

Contents

4.1 Overview

For cluster-based multi/many-core systems that support per-cluster DVFS, mapping application tasks and setting cluster frequencies play crucial roles to achieve energy efficiency, as discussed previously in Chapter 1. This chapter focuses on the energy efficiency of active applications executed locally in a given cluster.

For the local optimization in a cluster, executing multiple applications at a low cluster frequency for energy optimization is a challenge. Applications mapped into the same cluster influence each other due to possible platform resource competition and the common cluster frequency configuration. For an application with timing constraint, increasing the number of used processing cores allows a lower operating frequency level. However, using more cores for some applications leaves fewer available cores for other active applications. Consequently, the application that has the worst performance might determine the cluster frequency level and significantly increase the energy consumption. A management strategy is required for each cluster to determine the mapping of applications and set the cluster frequency level. Implementing such a management strategy can become very complex with the increasing application dynamism. Furthermore, the management strategy itself, as the mapping phase should not jeopardize the timing constraints of the applications. Therefore, an efficient management strategy with reduced complexity is of great importance.

In this chapter, we propose a new run-time management strategy to optimize energy consumption in a cluster while preserving the application timing and platform resources constraints. Like [8, 12], this chapter assumes that each application is known to be executed in which cluster beforehand. The assumption is practical with reference to some mapping strategies of Arm big.LITTLE platforms. For example, according to resource demands of applications, we can assign applications that require more CPU resources to the big cluster, and applications that require fewer CPU resources to the little cluster [5]. Once certain applications are assigned to a certain cluster, the work presented in this chapter deals with task-to-core mapping (*i.e.*, allocation and scheduling) and cluster frequency configuration within the cluster.

The main contributions of this work are:

- We propose a new run-time selection strategy to select the best prepared mapping of each active application. This strategy relies on hybrid mapping with application-based preparation (see Section 2.1.2). The design-time prepared data includes possible mappings for each application as well as the Minimum Allowed Frequency (MAF). This latter defines the minimum required frequency for a given prepared mapping in order to meet the application timing constraint. The selection management then tries to minimize the required frequency of the clusters according to the selected mappings.

- To further improve the energy efficiency, a new run-time mapping combination strategy is proposed to effectively combine the selected mappings of all active applications. The combination strategy can imply that some tasks can be merged on a core to reduce the resource usage of active applications. This combination strategy considers holistically the optimization for all active applications in the cluster (see Section 2.1.2).

- Several use-cases including up to 9 active applications (45 concurrent tasks in total) have been considered in order to evaluate the benefits of the proposed management approach

in terms of energy efficiency and utilization of resources. Results show up to 206% improvements of energy efficiency when compared with state-of-the-art approaches.

4.2 Summary of Related Work on Local Optimization

As previously discussed in Chapter 2, this dissertation work focuses on hybrid mapping strategies with application-based preparation. At design-time, different optimized mappings can be prepared for each supported application. At run-time, *mapping selection* and *mapping combination* are performed to achieve a mapping solution for active applications on available platform resources. The following summarizes the existing *mapping selection* and *mapping combination* strategies, which are particularly used as counterparts to our proposed management strategy.

In multi/many-core systems that support per-cluster DVFS, different *mapping selection* strategies can have different impacts on cluster frequency configuration. A *mapping-based selection* strategy is presented in [3], where all the possible prepared mappings are exhaustively explored to find the best set for the active applications. Theoretically, this strategy could be extended to support frequency reduction for the best energy efficiency. However the complexity of this approach increases with the number of prepared mappings, the number of applications and the number of cores. A *frequency-based selection* strategy is presented in [8], where different available frequency levels are explored in an iterative process. The platform frequency is incrementally set until a possible set of prepared mappings are found for all the active applications. This approach can take a long time to reach convergence when the frequency increment is small. In this work, we introduce a new parameter defined as *Minimum Allowed Frequency* (MAF) as a guideline to make the selection at run-time among all the design-time prepared mappings and the cluster frequency for the active applications. This approach reduces the searching iterations of run-time selections while offering energy efficiency.

Once mapping selection for all active applications is accomplished, *mapping combination* is performed under platform resource constraints. The authors of [3] explore all the possible task allocations in an exhaustive approach to find the best energy efficiency. This approach presents a scalability limitation. On the other extreme, a First-Come-First-Served (FCFS)-based strategy is proposed in [12, 13] in order to increase the speed of the mapping combination process. However, each core is only used by one application and thus some processing resources can be wasted. For less resource usage, the Largest Available Slot Packing (LASP) strategy is introduced in [14]. It allows different applications to be allocated to the same core at the cost of possible degradation of application performance. In this work, we propose a new run-time mapping combination strategy that reduces resource usage without sacrificing the performance of applications. Further comparisons of FCFS and LASP with our proposed mapping combination strategy are provided in Section 4.4.3.

4.3 Problem Definition

The aim of this chapter is to achieve local optimization of energy efficiency for multiple applications executed dynamically in a single cluster. For that purpose, a run-time management strategy is introduced to coordinate hybrid mapping and per-cluster DVFS. The run-time management environment is depicted in Figure 4.1.

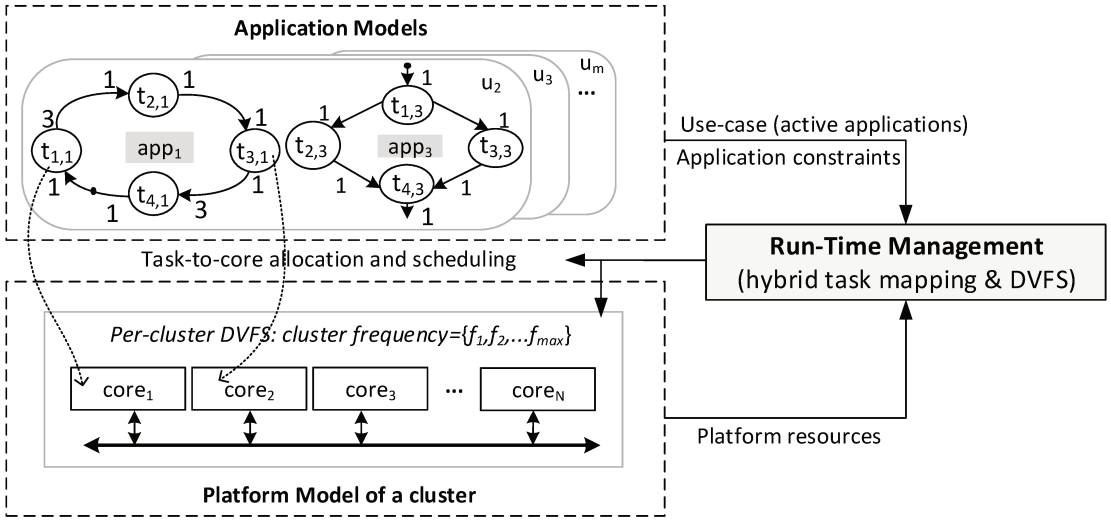

Figure 4.1: Run-time management of multiple applications active dynamically onto a cluster.

As shown in Figure 4.1, when a new use-case (*e.g.*, u_m) is detected in the cluster, the run-time management strategy identifies the set of active applications in the use-case. Then, the system constraints, including application timing, the cluster resource and the cluster frequency constraints are taken into account. According to the previous definitions in Chapter 3 (Section 3.3), a cluster is indexed by $cluster_j$. The index j will be used in Chapter 5 to index different clusters. Since this chapter focuses on a single cluster, the index j is not used in the scope of this chapter for clarity reason. That is, the considered cluster is indexed by $cluster$, which consists of N cores and supports discrete frequency levels $\{f_1, f_2, \ldots, f_{max}\}$. The objective of the run-time management strategy is to determine the task-to-core mapping (allocation and scheduling) and set the cluster frequency to achieve energy efficiency. This work characterizes energy efficiency by the average dynamic power consumption of active applications in a single cluster ($\sum_{i=1}^{I} P_{app_i}(cluster, f)$, see Eq.(3.1) for one cluster in Chapter 3). The management problem can be further stated by the following inputs, constraints and objective.

- **Inputs:**

  A use-case defining the active applications in the cluster: $u_m = \{app_1, app_2, \ldots, app_I\}$

- **Constraints:**

  Application timing constraint: $Period_{app_i}$ for each active application

  Cluster resource constraint: N available cores $\{core_1, core_2, \ldots, core_N\}$

  Cluster frequency constraint: $\{f_1, f_2, \ldots, f_{max}\}$

- **Objective:**

  Minimize the average dynamic power consumption of all the active applications executed in a single cluster: $\min \sum_{i=1}^{I} P_{app_i}(cluster, f)$.

4.4 Proposed Management Approach

The overview of the proposed run-time management strategy is depicted in Figure 4.2. The proposed strategy can be considered in three main steps. First, in the *A. Design-Time Prepared Data* step, a set of prepared mappings and their minimum required frequencies are stored for each application. Then, at run-time, Steps B. and C. are performed iteratively for each new use-case. In the *B. Run-Time Selection* step, a cluster frequency level is first selected from the design-time prepared frequencies for all the active applications. Then, the selected frequency level determines the selected mapping of each active application. Finally, in the *C. Run-Time Mapping Combination* step the selected mappings are combined. After this step, if the available resources are not enough, Steps B. and C. are repeated within an increased selected frequency level until a possible solution is found. These steps will be detailed in the next subsections.

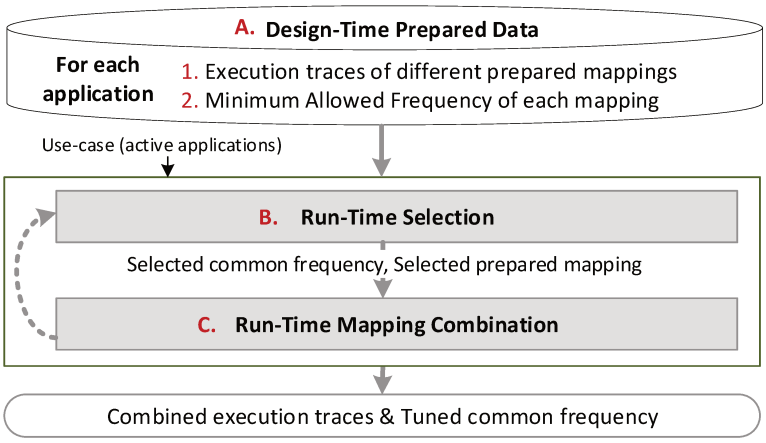

Figure 4.2: Overview of the proposed hybrid management approach, including design-time and run-time steps.

4.4.1 Design-time Prepared Data

At design-time, different information require to be prepared for each application.

(1) Multiple Prepared Mappings

For each application, a limited set of mappings requiring different numbers of cores and achieving different performance are prepared. Any static mapping strategy (*e.g.*, simulated annealing [40] or genetic algorithm [41]) can be used to explore design-time optimized mappings using different numbers of cores. For a prepared mapping, the communication cost (*e.g.*, time and power/energy) is neglected and only inter-core communication is taken into account, taking the same assumption as in [10]. The inter-core communication time highly depends on communication resources (*e.g.*, shared-memory or NoC). Our work assumes that the memory size and communication bandwidth are sufficient for inter-core communication activities, and communication time is much shorter compared to task computation time.

A mapping is characterized by its execution trace, *i.e*, a set of instants defining the start $x_s(t_{h,i})$ and end time $x_e(t_{h,i})$ of each task executed at a given frequency on the targeted platform. Here, for each prepared mapping, the instants within a period are prepared at f_{max}. The design-time execution trace of an application app_i mapped on c cores is defined by $X^c_{app_i} = \{x_{s\_t_{h,i}}(1), x_{e\_t_{h,i}}(1), \cdots x_{s\_t_{h,i}}(k), x_{e\_t_{h,i}}(k)\}$, where k refers to the k^{th} instance of a given task. Figure 4.3 illustrates two possible prepared execution traces of app_1, using (a) one core and (b) two cores respectively. Here, these two prepared execution traces are obtained through design-time analysis. In each execution trace, slots are formed by combining multiple adjacent executions of tasks on the same core. As illustrated in Figure 4.3, a slot is defined by

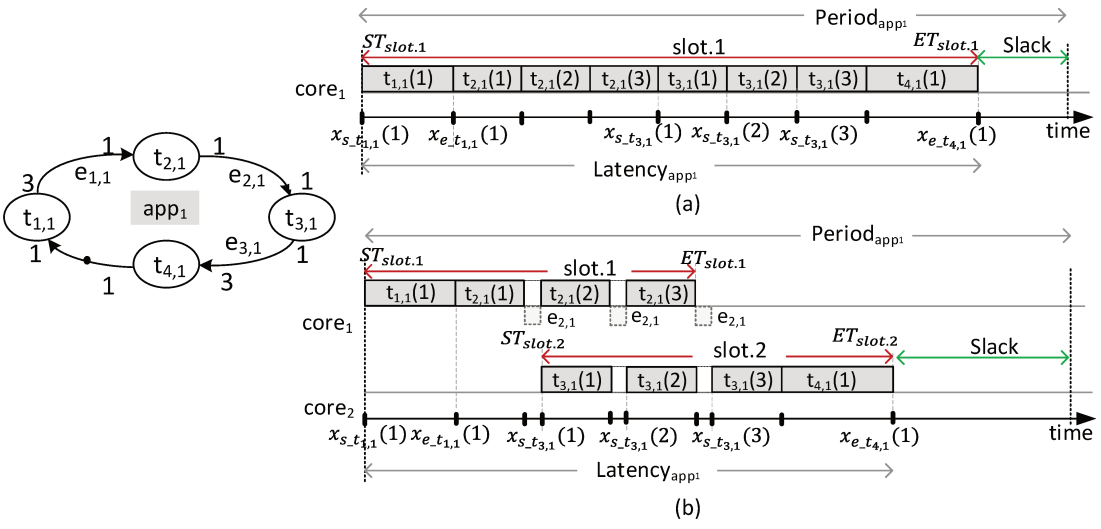

Figure 4.3: Design-time prepared execution traces for app_1 (a) mapped on one core ($X^1_{app_1}$) and (b) mapped on two cores ($X^2_{app_1}$).

its start time (ST) and end time (ET). $X^1_{app_1}$ is thus characterized with only one slot whereas $X^2_{app_1}$ contains two slots.

Different mappings can lead to different application latency. Let $Latency_{app_i}$ refers as the latency of app_i. It can be observed from Figure 4.3 that the latency ($Latency_{app_1}$) of $X^2_{app_1}$ is smaller than that of $X^1_{app_1}$ due to the execution parallelism. As a result, the slack time of the two prepared execution traces are different with respect to $Period_{app_1}$. The slack time can be used by DVFS to achieve lower frequencies but still meeting the considered timing constraint.

(2) Minimum Allowed Frequency (MAF)

Allowed frequencies for a given mapping are the frequencies that allow the mapped applications to respect their timing constraints. The allowed frequencies can be obtained by both, experimental measurements or application performance model (see Eq.(3.9) [6]). In this work, only the minimum level of the allowed frequencies, defined as Minimum Allowed Frequency (MAF), of each prepared mapping is stored. Let MAF^c_i be the MAF of app_i mapped on c cores (*i.e.*, $X^c_{app_i}$). Different mappings of an application can lead to different MAF with respect to the same timing constraint. For the example in Figure 4.3, MAF^2_1 can be smaller than MAF^1_1 because the longer slack time of $X^2_{app_1}$ allows the application execution at a lower frequency level without timing violation.

4.4.2 Run-time Selection

For every new use-case, the run-time selection is performed to explore a set of prepared mappings and a cluster frequency level to optimize the energy efficiency of the active applications. As all active applications within a cluster share the same frequency level, the objective is to find the lowest common allowed frequency under application timing and platform resource constraints. The selected frequency level corresponds to one of the prepared MAFs for all the active applications. Once this common frequency is selected, it further determines the application mappings that are selected. The selection process can be illustrated by the example of $u_2 = \{app_1, app_3\}$ in Figure 4.4.

Figure 4.4.a shows the average dynamic power evolution of the design-time prepared mappings for app_1 and app_3 (see Figure 4.1 for the SDF description). These curves are built based on the previously introduced power model (Eq.(3.8)) in Chapter 3. In this example, only two mappings per application are prepared and their associated MAFs are depicted in the figure. As previously discussed in Section 4.4.1, MAF^2_1 is smaller than MAF^1_1 due to different execution parallelism. Similarly, MAF^2_3 is smaller than MAF^1_3. As indicated from the figure, at different frequencies, the design-time mapping that should be selected for each application

[6] Eq.(3.9) can be approximately written into $Latency_{app_i}(f) \approx \frac{Latency_{app_i}(f_0) \times f_0}{f} \leq Period_{app_i}$, thus $MAF = f \leq \frac{Latency_{app_i}(f_0) \times f_0}{Period_{app_i}}$

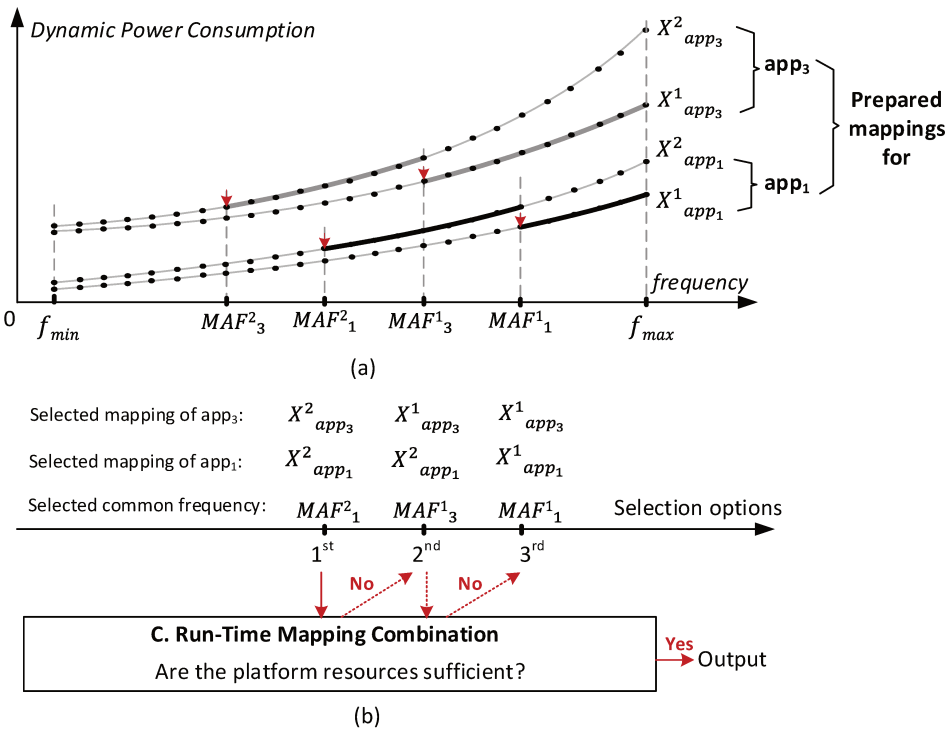

Figure 4.4: (a) Power evolution of the prepared mappings for app_1 and app_3. (b) Different mapping and frequency selection points formed by the prepared MAFs.

can be different. Take app_1 as an example, from MAF^2_1 to MAF^1_1, $X^2_{app_1}$ should be selected as it allows for lower operating frequencies. From MAF^1_1 to f_{max}, $X^1_{app_1}$ should be selected due to its less power consumption. The figure highlights the designated design-time mapping (bold line) selection for each application at different frequencies.

Part (b) of Figure 4.4 shows different selection options for app_1 and app_3. Each selection option contains the selected prepared mapping for each active application, and the selected cluster frequency, which is one of the prepared $MAFs$. In this example, the first selected frequency would be the lowest common allowed frequency MAF^2_1, as a consequence, the selected mappings will be $X^2_{app_1}$ and $X^2_{app_3}$. Finally, notice that after the run-time mapping combination step explained in the next subsection, if the applications timing constraints cannot be met due to insufficient platform resources, the next lowest MAF (MAF^1_3 in our example) will be selected and the process will be restarted again with the corresponding application mappings: $X^2_{app_1}$ and $X^1_{app_3}$. This approach aims to achieve a viable option with few iterations, without having to explore all feasible frequencies. The viable option corresponds to a lower MAF and lower dynamic energy consumption.

4.4.3 Run-Time Mapping Combination

After the active application's mappings selection ($X^c_{app_i}$), the selected mappings need to be combined in order to obtain a combined execution trace. In the mapping combination step, the selected design-time execution traces are processed in terms of slots. Our new combination strategy denoted *Grouped Applications Packing under Varied Constraints* (GAPVC) considers the advantages of the state-of-the-art strategies FCFS [12, 13] and LASP [14].

In the following, we first introduce FCFS and LASP strategies through simple examples. Then, based on the trade-off between FCFS and LASP, the GAPVC strategy is presented.

Examples of FCFS and LASP

For simplicity of presentation, we consider the mapping combination of app_1 and app_3 (in $u_2 = \{app_1, app_3\}$) onto a cluster. Suppose the selected prepared execution traces are $X^2_{app_1}$ and $X^2_{app_3}$, their slot-level execution traces are shown in Figure 4.5 (a) and (b) respectively. Notice that each design-time mapping contains information of task executions in one period. Figure 4.5 (a) extends the design-time prepared mapping $X^2_{app_1}$ (see Figure 4.3 (b)) within two $Period_{app_1}$, namely a hyper-period. As previously defined, a hyper-period refers to the least common multiple (LCM) of application periods. The same slots executed in different periods are defined as periodic slots (*e.g.*, $slot.1$ and $slot.5$ in Figure 4.5). The periodic slots are normally packed onto the same cores, in order to reduce the overhead of migrating slots from one core to another during execution.

Based on the two design-time execution traces in Figure 4.5 (a) and (b), part (c) shows the mapping combination result of FCFS. It merges the selected execution traces for individual applications successively. It first allocates the slots of $X^2_{app_1}$ onto $core_1$ and $core_2$ (in Figure 4.5 (c)). After that, it allocates the slots of $X^2_{app_3}$ onto $core_3$ and $core_4$. The combined execution trace uses 4 cores and keeps the application performance.

Figure 4.5 (d) shows the combined execution trace of LASP. The slot packing is restricted by the availability of each processing core (AC_{core_i}), which defines the time at which $core_i$ is available to pack a slot. Slots are packed according to the ascending order (marked by the slot index) of the slot end time (ET). The first slot ($slot.1$) is packed to an empty core ($core.1$) and then AC_{core_1} is updated to $ET_{slot.1}$. From the second slot, each slot tries to be packed in the used core with the least AC_{core_i}. If no used core is available, an empty core is chosen. Thus, $slot.4$ is packed to $core_1$, which has the least AC_{core_i} among the used cores. This can entail the execution delay of the periodic slots $slot.5$ and $slot.6$. This execution delay reduces the slack time of the combined execution and can even result in application deadline violations. This performance sacrifice can reduce resource usage (3 cores).

In a combined mapping, application performance and resource usage are the two important criteria to achieve energy efficiency in a cluster. On one hand, FCFS has better application performance, which allows for a lower frequency configuration. On the other hand, LASP has

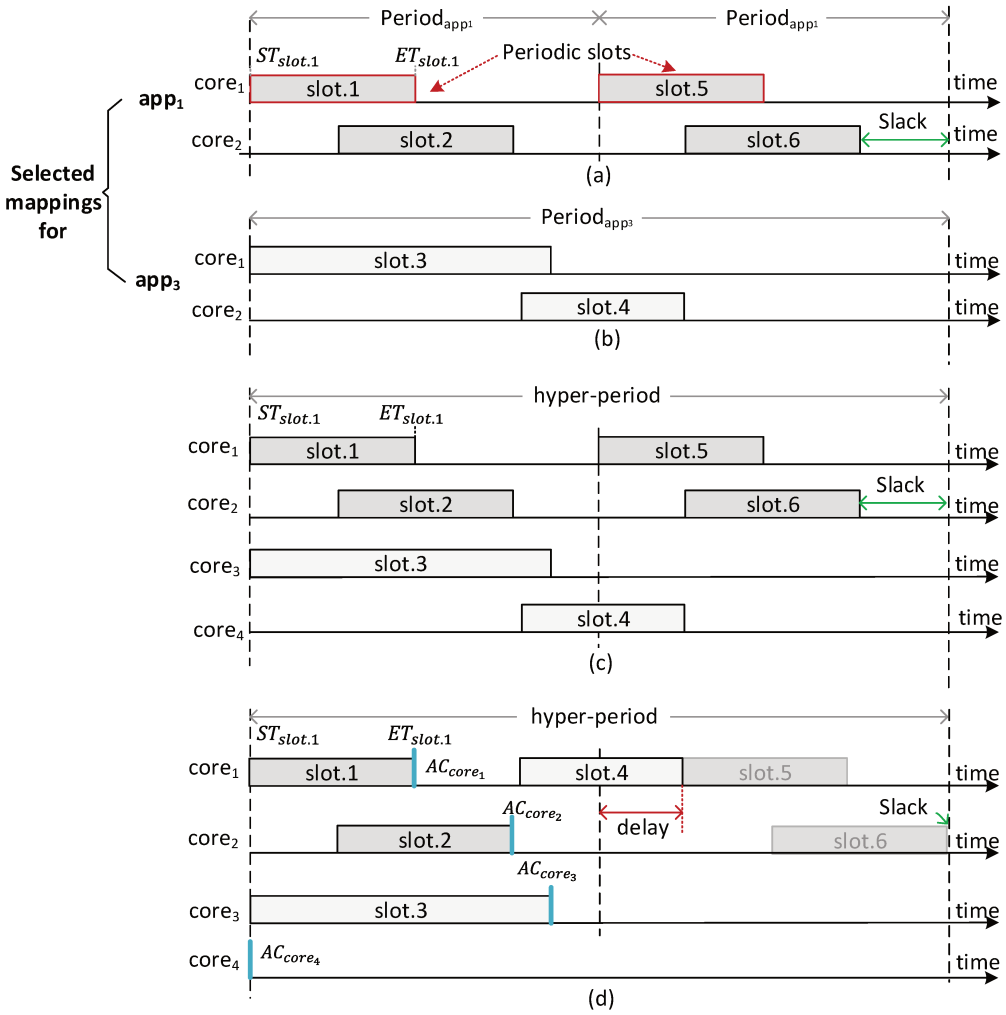

Figure 4.5: Extended the selected $X^2_{app_1}$ and $X^2_{app_3}$ within a hyper-period for app_1 and app_3; combined execution traces in a cluster by (c) FCFS [12, 13] and (d) LASP [14].

less resource usage, which allows a combined mapping to be achieved with fewer iterations (at lower MAF). To inherit the benefits of both strategies FCFS and LASP, GAPVC is proposed.

Proposed GAPVC strategy

The GAPVC strategy is proposed to obtain a combined execution trace that reduces resource usage without degrading application performance. Algorithm.1 presents the GAPVC strategy. The algorithm takes the run-time active applications (in u_m), their timing constraints (*i.e.*, $Period_{app_i}$), the selected design-time execution traces ($X^c_{app_i}$), the number of available cores (N) and available cluster frequencies as input to establish a run-time combined execution trace using a maximum of N cores.

In the first step, we divide active applications into different groups (lines 2-10). Grouping

applications can reduce the strategy complexity of mapping combination compared to LASP. In LASP, it aims to achieve a combined mapping for all active applications within a hyper-

Algorithm 1: Grouped Applications Packing under Varied Constraints (GAPVC) Strategy

Input:
$u_m = \{app_1, app_2 \cdots, app_I\}$: active applications in a running use-case
$Period_{app_i}$: period of each active application
$X^c_{app_i}$: selected execution trace of each active application
N: available number of cores in the cluster
$\{f_1, f_2, \ldots, f_{max}\}$: available cluster frequencies
Output:
X'_{Apps}: a run-time combined execution trace using a maximum of N cores

Run-Time Mapping Combination;
//Step 1: Group active applications in several groups
Sort active applications in the ascending order of $Period_{app_i}$;
for *each application* $app_i \in u_m$ **do**
  if $Period_{app_i} : Period_{app_{i+n}} = 1 : \mathbb{N}^+$,$app_i, app_{n+1} \in u_m$ **then**
    $Group_y = \{app_i, app_{i+n}\}$; //group app_i and app_{i+n} together;
  else
    $Group_y = \{app_i\}$;
  end
end
//Step 2: Pack slots of grouped applications to cores Initialize $nEmptyCore = N_j$;
for *each* $Group$ **do**
  Extend $X^c_{app_i}$ of each paired applications within their $hyper - period$;
  Sort all slots in ascending order based on ET_{slot} ;
  Set $AC_{core} = 0, UC_{core} = hyper - period$ for empty cores;
  for *each slot "*$slot.s$*"* $\in Apps$ **do**
    if *packing of slot* s *is not fixed* **then**
      if *(first slot* || *((*$ST_{slot.s} < AC_{UsedCore}$ || $ET_{slots.s} > UC_{UsedCore}$*)* && $nEmptyCore>0$*))* **then**
        Select $EmptyCore$;
        $nEmptyCore--$;
      else
        Select $UsedCore$;
      end
    else
      Select $FixedCore$;
    end
    Pack s to the selected core;
    Update $AC_{SelectedCore}$ and $DC_{SelectedCore}$;
  end
  Update $nEmptyCore$;
end

period of all active applications. The value of the hyper-period can be very huge due to various periods of several/many applications. As a result, after design-time prepared mappings are extended within a hyper-period, the total number of slots would be very huge, which add more computation burden at run-time. Such situations can be avoided by dividing applications into several groups. Different numbers of applications can be in a group. Here, we chose to put two applications per group according to the relationship $Period_{app_i} : Period_{app_{i+n}} = 1 : \mathbb{N}^+$. This relationship ensures that the *hyper-period* of the two active applications (in a group) equals to $Period_{app_{i+n}}$. It can consequently avoid an excessive increase in the number of extended slots. If an application cannot be grouped with another, only one application is kept in a group.

In the second step, each group of applications is successively packed to cores (lines 11-32). After completing the packing of one group then it goes to the next one. The slot packing results of all groups form the final combined execution trace.

To illustrate the slot packing of *each* application group, let's assume that app_1 and app_3 are grouped together and $X^2_{app_1}$ and $X^2_{app_3}$ (see Figure 4.5) are the selected prepared mappings. For a better explanation, Figure 4.6 presents the combined execution traces of GAPVC at two different packing moments. In GAPVC, the slot packing is restricted by two different constraints, including the availability of each processing core (AC_{core_i} used in LASP) and unavailability of each processing core (UC_{core_i}) defining the time at which $core_i$ is unavailable to pack the next slot. For example, in figure (a), AC_{core_1} corresponds to the End Time of last

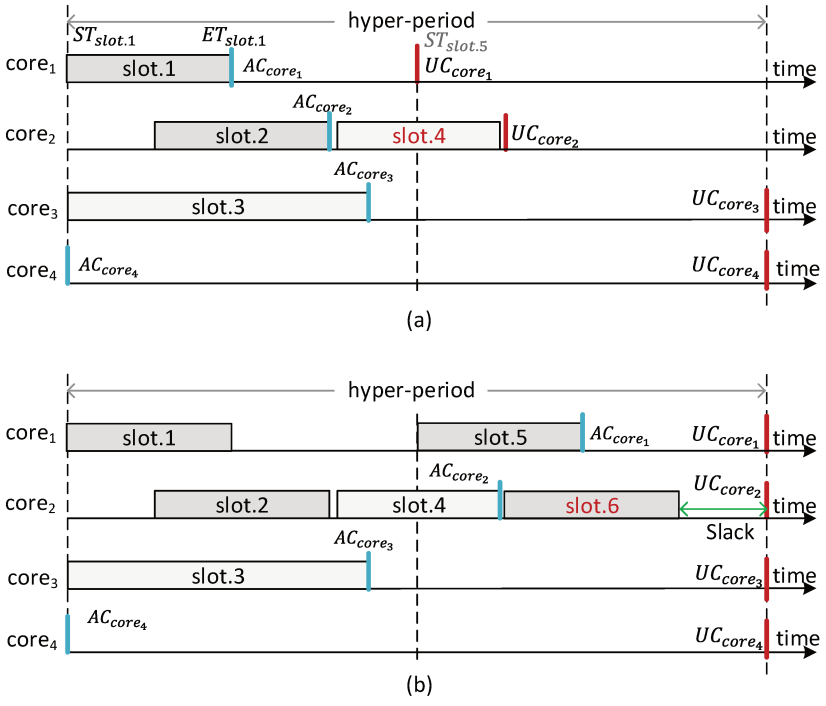

Figure 4.6: The slot packing results of the GAPVC strategy when packing (a) slot.4 and (b) slot.6, based on $X^2_{app_1}$ and $X^2_{app_3}$ of Figure 4.5.

already packed slot on the core ($ET_{slot.1}$), while UC_{core_1} defines the Starting Time of the next periodic slot $ST_{slot.5}$ on the core. To pack a slot (*e.g.*, $slot.s$) on a core, $ST_{slot.s}$ of the slot should not be smaller than AC_{core_i}, while $ET_{slot.s}$ should not be larger than UC_{core_i}. UC constraints guarantee the availability of each core for all future periodic slots of packed slots. Following this rule, $slot.4$ is packed onto $core_2$ in GAPVC, while $slot.4$ is packed onto $core_1$ in LASP. After finishing the packing of $slot.4$ and $slot.5$, AC_{core_i} and UC_{core_i} are updated accordingly in figure (b). By comparing the mapping combination results of FCFS and LASP (Figure 4.5 (c) and (d)), we can observe that the packing of GAPVC (see Figure 4.6) reduces resource usage but keeps the application performance. Note that the example in Figure 4.6 illustrates the packing of one application group. The packing of another group is performed in the remaining empty cores, which means that each core is only packed with slots from the same application group.

4.5 Experimental Evaluations

4.5.1 Simulation Setup

The proposed strategy is evaluated by simulation on Visual Studio in C++. The evaluation is performed for multimedia applications running on a single cluster under different possible use-cases. As previously stated in Chapter 3, a cluster consists of homogeneous cores that support a specific range of frequencies. The cluster frequencies range from $0.2GHz$ to $1.4GHz$ with a step increment of $0.1GHz$. Different resource constraints (*i.e.,* N) are taken into account in the following evaluations. The simulated applications, from now on denoted by app_1 to app_9, are defined in Table 4.1. They were derived from reference applications (H263 encoder, H263 decoder, and JPEG decoder) with different input and output tokens. As indicated in Table 4.1, we prepared for each application multiple design-time mappings with different numbers of used cores and different MAF values. It might be possible that some mappings using different numbers of cores have the same MAF. In such cases, only the mapping requiring fewer core is kept to achieve less communication cost between cores (due to parallelism). Let app_2 be an example, the mappings using 1 core and 2 cores have the same MAF, and only 1-core mapping is kept in the design-time prepared data.

In this chapter, we use a parameter "communication energy ratio" (R_{comm}) to indicate the ratio between communication energy and computation energy within a period for each prepared mapping. Here, we set R_{comm} arbitrarily for the prepared mapping using the maximum number of cores (highlighted in gray in Figure 4.1) for each application. Then, R_{comm} of other prepared mappings of the same application are approximately set according to the proportional relationship[7] between communication tokens (*i.e.*, only counts communication

[7] $\frac{R_{comm}\text{of another mapping}}{R_{comm}\text{ of mapping}^*} = \frac{\text{communication tokens between cores of another mapping}}{\text{communication tokens between cores of mapping}^*}$, mapping* is the mapping with known R_{comm}.

Table 4.1: Design-time prepared information of the considered applications

Application				**Prepared Mapping** $X^c_{app_i}$		
Type	app_i	**Nb of tokens of each task**	**Period** (μs)	**Nb of used cores**	**MAF** (GHz)	R_{comm}[8]
H263 decoder :4 tasks :3 edges	app_1	$\{1, 6, 6, 1\}$	60	1	1.1	0
				2	1.0	1.1%
	app_2	$\{1, 4, 4, 1\}$	180	1	0.4	0
	app_3	$\{1, 264, 264, 1\}$	360	1	1.3	0
				2	0.8	6.68%
H263 encoder :5 tasks :4 edges	app_4	$\{1, 5, 5, 5, 1\}$	540	1	1.3	0
				2	1.2	1%
	app_5	$\{1, 15, 15, 15, 1\}$	1080	1	0.9	0
				2	0.7	2.25%
	app_6	$\{1, 45, 45, 45, 1\}$	1080	2	1.1	4.13%
JPEG decoder :6 tasks :5 edges	app_7	$\{1, 7, 7, 7, 7, 1\}$	180	1	1.3	0
				2	1.1	3.38%
				4	1.0	6.77%
	app_8	$\{1, 9, 9, 9, 9, 1\}$	360	1	0.8	0
				2	0.7	3.70%
				4	0.6	7.41%
	app_9	$\{1, 22, 22, 22, 22, 1\}$	1080	1	0.5	0
				2	0.4	4.61%

[8] R_{comm}: *communication energy ratio* indicates the ratio between communication energy and computation energy within a period for each prepared mapping.

Table 4.2: Considered use-cases

Use-case	**Active Applications**	**Use-case**	**Active Applications**
u_1	app_3, app_4	u_5	$app_1, app_3, app_5, app_6$
u_2	app_1, app_6, app_7	u_6	$app_5, app_6, app_7, app_8$
u_3	app_2, app_5, app_6	u_7	$app_1, app_2, app_3, app_7, app_8, app_9$
u_4	$app_1, app_2, app_3, app_4$	u_8	$app_1, app_2, app_4, app_6, app_7, app_8$

between cores). In this way, the communication energy varies for different prepared mappings of each application.

For the 9 considered applications, there can be 511 (*i.e.*, $2^9 - 1$) possible use-cases with different active applications for each one. Table 4.2 gives 8 example use-cases, which are used to specifically describe the characteristics of the evaluated strategies.

4.5.2 Evaluations of the proposed Hybrid Management Strategy

(1) Run-Time Mapping Combination

One main novelty of this work concerns the mapping combination. In this evaluation, the proposed mapping combination strategy GAPVC is compared to FCFS strategy [12, 13] and to LASP strategy [14]. For a fair comparison, the three strategies are performed based on the same selected mappings. In this experiment, we select the prepared mapping using the maximum number of cores (highlighted in gray in Figure 4.1) for each application in each use-case. Note that different prepared mappings can be selected for this comparison. Figure 4.7 shows the compared results in terms of latency and required cores for the 8 considered use-cases.

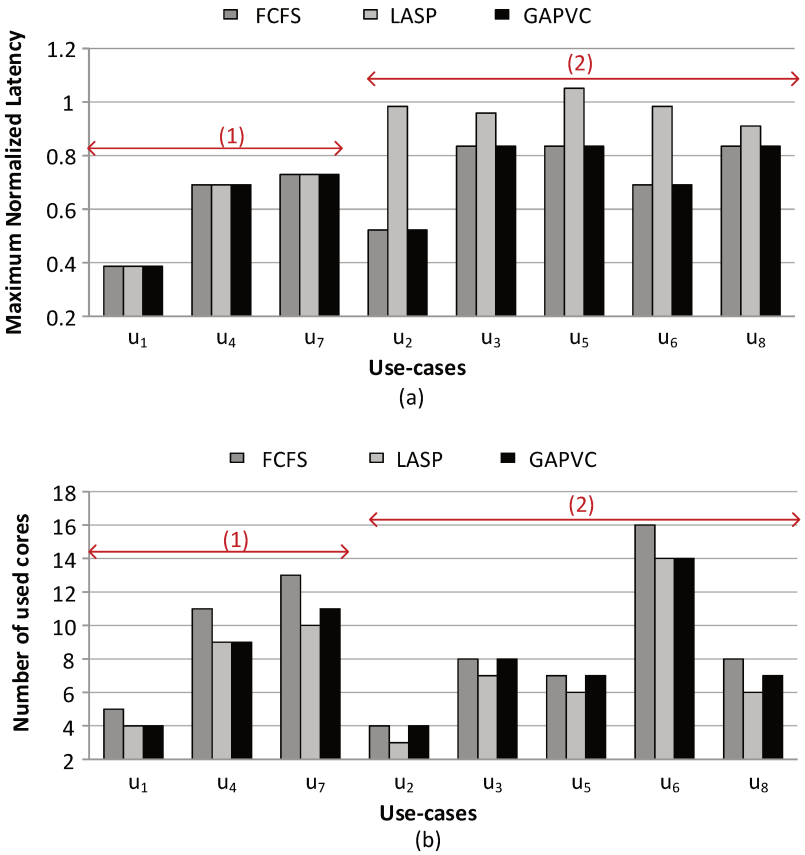

Figure 4.7: (a) The maximum latency normalized to application period and (b) the number of used cores for different use-cases. Results are given for FCFS, LASP and GAPVC.

After the mapping combination, the latency normalized to a period can be obtained for active applications. Part (a) of Figure 4.7 captures the maximum normalized latency in each use-case, which determines the tuned frequency and highly influences the energy efficiency. Part (b) of Figure 4.7 presents the number of required cores for each strategy. Index (1) refers

to the use-cases where the three mapping combination strategies achieve the same application latency, while FCFS uses more cores as it only executes tasks of one application onto one core. For the use-cases indicated by index (2), LASP leads to high application latency due to the execution delay of tasks. In u_5, such execution delay even violates the application period constraint. The proposed GAPVC strategy makes a trade-off for the used cores between FCFS and LASP without sacrificing application performance.

General evaluations are performed for all the possible use-cases (*i.e.*, 511) of the 9 considered applications. Compared to FCFS, GAPVC has less resource usage to achieve the same application latency in 287 (*i.e.*, 56.2%) use-cases. Compared to LASP, GAPVC avoids task delay in 228 (*i.e.*, 44.6%) use-cases and avoids timing violations in 21 (*i.e.*, 4%) use-cases. It means that the LASP strategy has 4% failed use-cases due to timing violation. As the experimental evaluation is performed under the constraint of sufficient cores, the three compared strategies have no failed use-case due to resource insufficiency.

(2) Run-time Selection and Run-time Combination

Our proposed run-time management strategy consists of the MAF-based selection strategy and the GAPVC mapping combination strategy (MAF-based&GAPVC). We compare our run-time management strategy with FCFS [12] and LASP [14] based on the same selection strategy (MAF-based&FCFS, MAF-based&LASP). Additionally, in order to compare our results with the optimal solution, we also implemented the exhaustive approach presented in [3]. The algorithm has been extended to support frequency tuning after the exhaustive mapping selection and combination to further optimize the average dynamic power of active applications.

For different management strategies, Figure 4.8 compares the obtained dynamic power consumption for 8 considered use-cases, taking into account a different number of cores in the targeted cluster. For each use-case, the power values are normalized to the result obtained by the exhaustive search [3] under the constraint of 8 available cores (*i.e.*, 4, 6, 8 cores). Observing the use-cases indicated by index (1), more available cores can lead to a lower dynamic average power of executed applications. This is because the availability of platform resources allows finding a lower common frequency with fewer iterations. We can also notice that in some cases (such as in u_8 under the constraint of 4 available cores) no solution can be found meaning that the mapping is not feasible for that platform in this execution scenario.

For the use-cases indicated by index (2), we can observe that our proposed strategy is able to achieve lower average dynamic power consumption (*i.e.*, better energy efficiency) than MAF-based&FCFS. Since FCFS uses mores cores than GAPVC to get a combined mapping, it may need more iterations to find the feasible selection, which makes the common frequency higher. As a consequence, MAF-based&FCFS achieves the highest average dynamic power in u_1 when compared to the others, and no solution can be found in u_7 under the constraint of 6 available cores.

The observations of use-cases indicated by index (3) illustrate the limitation of the

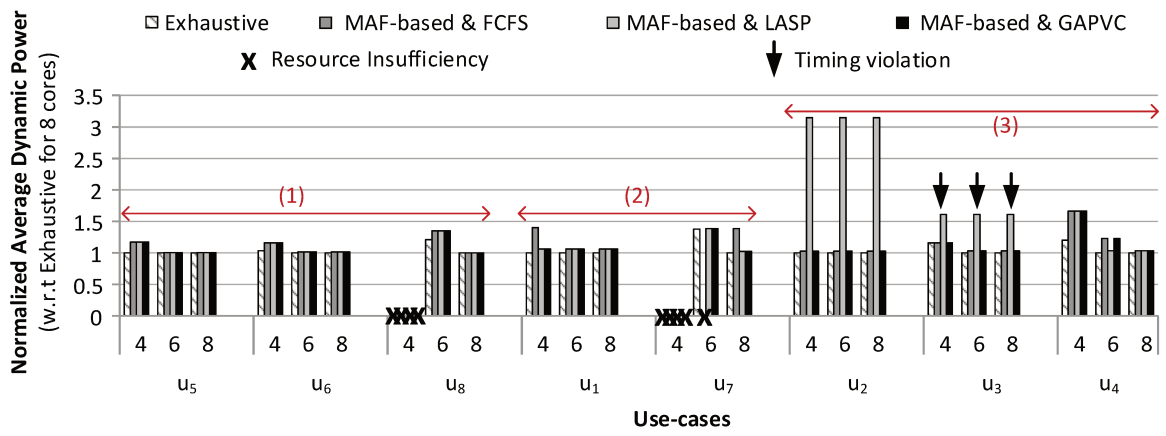

Figure 4.8: The average dynamic power at the tuned frequency within a hyper-period under different resource constraints. The power of each use-case is normalized to the result achieved by the Exhaustive strategy under the constraint of 8 available cores.

management strategy based on LASP. MAF-based&LASP leads to the most average power in u_2 and to timing violation in u_3. Because LASP can result in execution delay of tasks, it may make greater cluster frequency or even violate the timing constraint. Our run-time management strategy can optimize average power consumption without violating any timing constraint.

Moreover, Table.4.3 compares the proposed run-time management strategy (*i.e.*, MAF-based& GAPVC) with MAF-based&FCFS and MAF-based&LASP strategies among all 511 possible use-cases under different resource constraints (6, 8 and 10 cores). The exhaustive approach cannot be tested on these experiments due to scalability issues.

Table 4.3: Comparison of run-time management strategies among 511 use-cases

Cores	Strategy	Nb of failed u_m			Compared to the proposed strategy	
		Total	Due to resource insufficiency	Due to task delay	Nb of feasible u_m with more average power	Range of average power increase w.r.t proposed
6	MAF-based&FCFS	102	102	0	16	12.1% to 34.6%
	MAF-based&LASP	90	67	23	31	142.8% to 206.3%
	Proposed	76	76	0	-	-
8	MAF-based&FCFS	9	9	0	32	12.8% to 35.6%
	MAF-based&LASP	34	2	32	40	206.3%
	Proposed	3	3	0	-	-
10	MAF-based&FCFS	0	0	0	5	13.6% to 36.0%
	MAF-based&LASP	32	0	32	40	206.3%
	Proposed	0	0	0	-	-

The left part of the table compares the number of use-cases where no solution can be found.

It can be observed that the strategies using FCFS and LASP have more mapping failures than GAPVC. Due to the more resource usage, all the failures observed with MAF-based&FCFS come from resource insufficiency. The failures observed with MAF-based&LASP consist of resource insufficiency and timing violations caused by task delays. The resource failures can be avoided when the number of available cores are over 10. However, the failures due to task delay are still presented even when more cores are available.

The right part of the table lists the number of use-cases where the proposed strategy achieves better energy efficiency among the common feasible use-cases. Compared to MAF-based&FCFS and MAF-based&LASP, the proposed strategy is able to reduce average power consumption by approximately 36% and 206%, respectively. For the other use-cases, the three compared strategies find the same results.

4.5.3 Evaluations of the Strategy Complexity

(1) Complexity of Run-Time Selection

In Figure 4.9 we compare our selection approach with a mapping-based selection strategy [3] and a frequency-based selection strategy [8]. For a fair comparison, the selected mappings of each selection strategy are all combined with GAPVC.

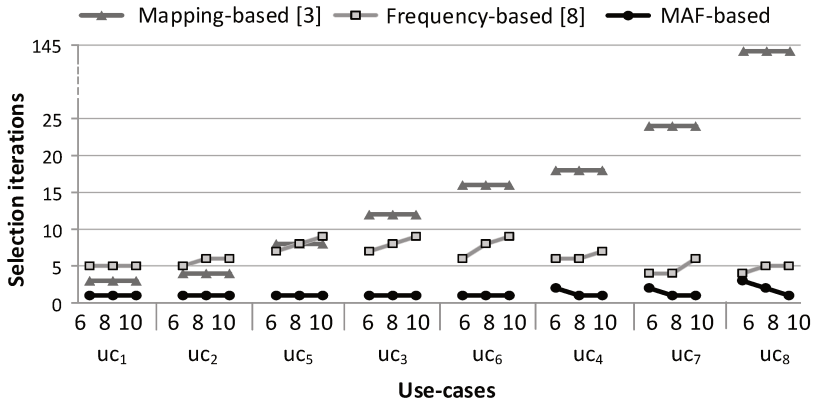

Figure 4.9: The number of selection iterations for 8 use-cases under different resource constraints.

Firstly, it can be observed that the Mapping-based strategy uses a large number of iterations. Since the selection strategy evaluates all possible prepared mappings for all active applications, the number of iterations can be large in the use-case (*i.e.*, u_8) with more available prepared mappings. Secondly, the iterations for the Frequency-based strategy are smaller than the total number of available platform frequencies (*i.e.*, 13). This strategy starts mapping exploration at a low estimated frequency level. The starting frequency level is estimated by allowing all the tasks mapped continuously without any space. Such estimations are optimistic because it

neglects the task dependencies, and consequently several explorations are needed to found a possible solution at a higher frequency level. In addition, more available cores can result in an even lower estimated starting frequency, which can also increase the number of exploration iterations in certain use-cases (*i.e.*,u_3). Finally, we can observe that the proposed MAF-based selection strategy requires the least number of iterations. Our proposed selection strategy reduces iterations by only exploring the frequency levels that are marked as common MAF (see Section 4.4.2).

(2) Complexity of Run-Time Mapping Combination

For GAPVC, FCFS and LASP, the strategy complexity of mapping combination highly depends on the number of allocated slots in a use-case. FCFS presents the lowest complexity and it deals with the slots of each active application within one period. LASP has the greatest complexity, as it maps all the slots extended within a hyper-period of all the active applications directly. For many active applications, the large value of their hyper-period can make the number of extended slots huge. As discussed in Section 4.4.3, by dividing active applications into groups, GAPVC extends the slots for each grouped applications within a hyper-period with a smaller value. The result of summing all the slots of all grouped applications can be much less than the considered slots of LASP.

We implemented the three mapping combination strategies on one little core and one big core of the Exynos5422 platform [21]. We measured the execution time of each use-case (among all the possible 511 use-cases). The average and the maximum values of the time required to obtain the combined mapping for a use-case are shown in Table 4.4.

Table 4.4: The average and maximum execution time of mapping combination strategies on one little core and one big core of Exynos 5422 big.LITTLE platform (among the 511 use-cases)

Cluster	Value	FCFS (ms)	LASP (ms)	GAPVC (ms)
Little	Average	0.043	2.225	0.137
	Maximum	0.177	10.015	0.417
Big	Average	0.011	0.693	0.037
	Maximum	0.030	3.211	0.135

From the comparison in Table 4.4, we can observe that the complexity of GAPVC is greater than the FCFS one, but much significantly less than the complexity of LASP. Let the average results on the little core as examples. The average result of GAPVC is 3.18 (*i.e.*, $\frac{0.137}{0.043}$) times of FCFS, while the result of LASP is 16.24 (*i.e.*, $\frac{2.225}{0.137}$) times of GAPVC.

4.6 Summary and Discussion

For the local optimization of energy efficiency (characterized by average dynamic power) for multiple task-dependent applications executed dynamically in a cluster, this chapter presents a new run-time management strategy to determine task-to-core mapping (*i.e.*, allocation and scheduling) and set the cluster frequency accordingly. The proposed management strategy is based on a design-time database, which includes multiple prepared mappings for each application and a new defined parameter Minimum Allowed Frequency (MAF) for each prepared mapping. For a set of active applications (in a use-case), a new run-time selection strategy is proposed to select a low cluster frequency and an efficient prepared mapping for each application based on MAFs. Moreover, a new run-time mapping combination strategy GAPVC is proposed to combine the selected mappings with less resource usage without sacrificing application performance. Combining the MAF-based selection strategy and GAPVC strategy it is possible to achieve lower average dynamic power with reduced complexity compared to state-of-the-art strategies. In our experimental evaluations, various case-studies with different sets of active applications are considered, demonstrating that our proposed run-time management strategy can reduce average power consumption by up to 206% when compared to the literature.

The goal of this work can also be understood as mapping the active applications onto limited processing resources at a low cluster frequency level. The applications active in a cluster are assumed to be known beforehand and they are not allowed to execute in other clusters. The way to reduce the cluster frequency level is to appropriately deal with the varying resource competition from dynamic active applications via appropriately selecting prepared mappings, which have different trade-offs between resource usage and performance. From one use-case to another, the selected mappings and the combined mapping can be totally different. Some additional overheads can include the cost of migrating tasks from one core to another. The consideration of migration overheads can be further addressed in our future work.

The proposed management strategy is expected to achieve local optimization in each cluster. As previously presented in Figure 1.4, local management in different clusters can be structured in distributed management in a cluster-based multi/many-core platform. Energy efficiency can be achieved locally in each cluster. However, global energy optimization in the overall system may not be achieved. Because this chapter holds the assumption that applications are not allowed to execute in other clusters even if there is intense resource competition in a cluster. This limitation is considered in the next chapter.

Last but not least, the proposed run-time management strategy in this chapter is possible to be used to in multi/many-core platforms that support Global DVFS (all homogeneous cores in a chip share the same v/f level). This situation can be seen as only one cluster in platforms. Note that our proposed strategy does not consider the case of having heterogeneous cores in a cluster. Because different core types use different voltages at the same frequency level, having different types of cores in the same voltage/frequency island is not power/energy efficiency [77].

Chapter 5

Run-Time Management for Global Optimization

Contents

5.1 Overview

The run-time management strategy introduced in Chapter 4 concerns local optimization of the energy efficiency within a cluster. In this chapter, we aim to achieve global optimization of the overall cluster-based multi/many-core system. As applications can be allocated to different clusters, application-to-cluster assignment plays an important role to determine the energy efficiency of the overall system. As previously discussed in Chapter 4, the cluster frequency is determined by the application with the strictest timing constraint in the cluster. Different application-to-cluster assignments lead to different frequency configurations among clusters, and thus to different energy efficiency in the whole system. It is required to assign applications to clusters carefully to achieve the global optimization target. However, this optimization problem is challenging because obtaining the optimal mapping solution is NP-hard [4, 29], and the optimization problem becomes even more complex when taking DVFS into account. In some state-of-the-art strategies, application mapping and DVFS are separated into two independent steps, without considering the mutual influence between the two techniques. Moreover, most existing application-to-cluster assignment solutions [49, 78] are proposed for small size platforms (*e.g.*, ARM big.LITTLE with 4 cores within a cluster), which might not be efficient for many-core platforms encompassing more clusters (*i.e.*, homogeneous/heterogeneous) or more cores within each cluster (*e.g.*, Kalray MPPA2®-256 [20] with up to 16 cores in a cluster).

In this chapter, we present a run-time management strategy for different platform sizes. The proposed strategy is used to manage the active applications in each use-case. For a better management scalability, a 2-level hierarchical management structure is adopted. Figure 5.1 shows the overview of our hierarchical management. Based on design-time prepared data, the

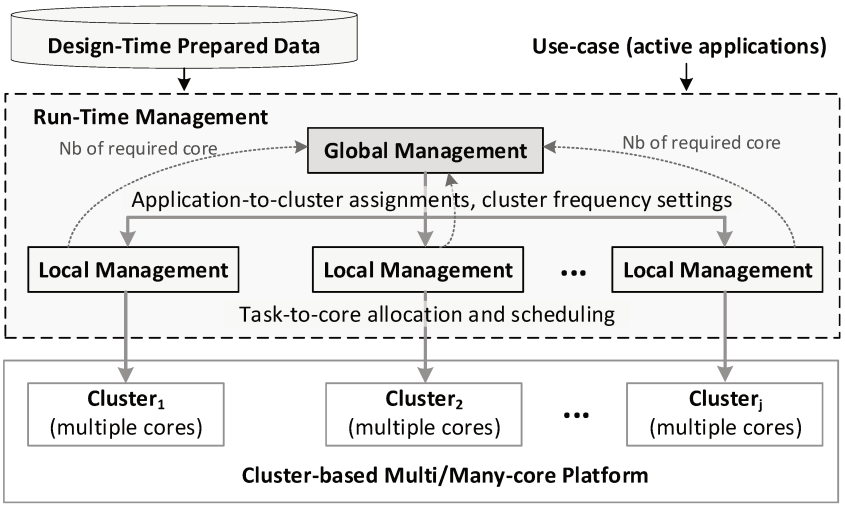

Figure 5.1: The run-time hierarchical management for multiple applications executed dynamically on a cluster-based multi/many-core platform.

global management determines application-to-cluster assignments and sets cluster frequency levels, while the local management optimizes task-to-core allocation and scheduling in each cluster. The global management decisions are influenced by the local management in terms of resource usage in each cluster.

Based on the 2-level hierarchical management, this chapter makes the following contributions.

- For the global management, we present a 0-1 Integer Programming (IP) model[9] that integrates application-to-cluster assignments and cluster frequency configurations to formulate the average dynamic power of the overall system at optimized cluster frequencies. The formulation relies on some design-time prepared data (application-based preparation, see Section 2.1.2), including one optimized mapping (*i.e.*, with the best application performance) and its corresponding MAF for each application. The prepared MAFs are used to estimate the optimized cluster frequencies based on different application-to-cluster assignments.

- To achieve the solution of the 0-1 IP optimization problem, it is not feasible to exhaustively search for the optimal solution in a reasonable time. Motivated by reducing strategy complexity, we propose a first global management strategy that aims to deliver near-optimal solutions. The strategy considers application-to-cluster assignments for all applications *holistically* in a use-case, allowing all migrations[10].

- We propose a second global management strategy that considers assignments for applications *individually*. It assigns only the newly arrived applications and allows application migrations to a certain extent in each use-case. Compared to the first strategy, this strategy can reduce the number of application migrations with a limited impact on energy efficiency.

- Like most related works, these two proposed global management strategies use a simple mapping combination strategy (FCFS) to estimate the number of used cores in each cluster locally. To further reduce the number of used cores in each cluster without degrading application performance, we consider also the benefits of using the GAPVC combination strategy (see Chapter 4, Section 4.4.3) for the local management.

- We evaluate our proposed management strategies in different use-cases (*i.e.*, different sets of active applications) and different platform configurations (*e.g.*, different numbers of heterogeneous/homogeneous clusters, different numbers of cores in each cluster, ...).

[9] 0-1 IP model: a specific type of integer programming where the decision variable is 0 or 1.

[10] application migration: from the current use-case to the next, the number of the current active applications that change their assignments from one cluster to another.

5.2 Summary of the Related Work on Global Optimization

As previously discussed in Chapter 2 (Section 2.2), there has been some prior researches about the energy optimization of cluster-based multi/many-core systems. Most state-of-the-art strategies perform application-to-cluster assignment and cluster frequency configuration into two separated steps. Generally, application mapping is performed first, then cluster frequencies are decreased as much as possible under timing constraints. In this work, we formulate the relationship between application mapping and optimized cluster frequencies into a 0-1 IP model.

For Arm big.LITTLE cluster-based platforms, applications are usually assigned according to their performance and resource demands. Particularly, the work of [49] employs a Low-Power-First (LPF) based strategy, where each application is attempted to be assigned to the low-power cores (*i.e.*, little cores) first. If the performance constraint is not satisfied, high-performance cores (*i.e.*, big cores) are used. After application assignments are finished, the cluster frequency is adapted with respect to a pre-defined power budget. For independent periodic tasks on a platform with more clusters, the work of [77] presents a Low-Energy-First (LEF) based mapping strategy. At fixed cluster frequencies (*e.g.*, maximum level), the LEF strategy assigns periodic tasks successively to the cluster that can achieve the lowest energy consumption for each individual task. Then, each cluster reduces the frequency level as much as possible.

These two works give the highest priority to one core type when performing application assignments. For platforms with more cores in the clusters, we can predict that in the case of many active applications, the workload of the top-priority cluster can be very heavy. Consequently, the frequency and the energy consumption of one cluster can be very high, while other clusters can be empty without any application. Due to the workload imbalance between clusters, platform resources may not be fully utilized, thus missing out opportunities for further energy optimization. In contrast to previous works, we propose global management strategies (*i.e.*, application assignments and cluster frequencies) to achieve near-optimal energy efficiency of task-dependent applications on cluster-based multi/many platforms. The near-optimal solutions are evaluated based on our formulated 0-1 IP model. The proposed global management strategies can scale to platforms with more different clusters and more cores inside each cluster.

5.3 0-1 IP Formulations of Global Management

Based on the hierarchical management structure shown in Figure 5.1, this section focuses on the global management decision for a set of active applications (*i.e.*, in a certain use-case) executed on a cluster-based multi/many-core platform. Here, we assume that all tasks of each application are assigned to the same cluster, but an application can change its execution in different clusters

for different use-cases. Additionally, global management depends on design-time prepared data. We consider that one optimized mapping is prepared for each application. The optimized mapping has the best application performance and it can be prepared by any static mapping algorithm (*e.g.*, simulated annealing [40] or genetic algorithm [41]). The Minimum Allowed Frequency (MAF) is prepared for each design-time mapping as well. MAF is the minimum required frequency of an application mapping that satisfies application timing constraint (first introduced in Section 3.5).

According to the previously definitions in Chapter 4 (Section 4.4.1), the MAF of app_i mapped on c cores (*i.e.*, $X^c_{app_i}$) is indexed by MAF^c_i. In this chapter, the global management considers one design-time prepared mapping for each application. For the sake of clarity, let c_i denote the number of used cores of each considered application mapping (app_i) in global management. In addition, the global management aims to assign applications among multiple clusters, the MAF notation should be extended into consider the impact of different clusters. From here on, the considered mapping of each application is indexed by $X^{c_i}_{app_i}$, and its MAF in a certain cluster ($cluster_j$) is indexed by $MAF^{c_i}_{i,j}$ in global management.

The global management problem of application-to-cluster assignments and cluster frequency configurations is formulated into a 0-1 IP model. In the formulation, the prepared MAFs are used to estimate the optimized cluster frequencies. Figure 5.2 gives one example to explain how cluster frequency is determined for the assigned applications in a certain cluster ($cluster_j$). This figure depicts the dynamic power consumption evolution of the considered prepared mappings ($X^2_{app_1}$ and $X^2_{app_3}$) for the active applications (app_1 and app_3) in a certain cluster. The allowed frequencies of each application mapping is highlighted (in bold line). In this case, the global management sets the cluster frequency to the minimum common frequency that support the execution of all active applications, which corresponds to the maximum MAF ($MAF^2_{1,j}$) of the considered prepared mappings.

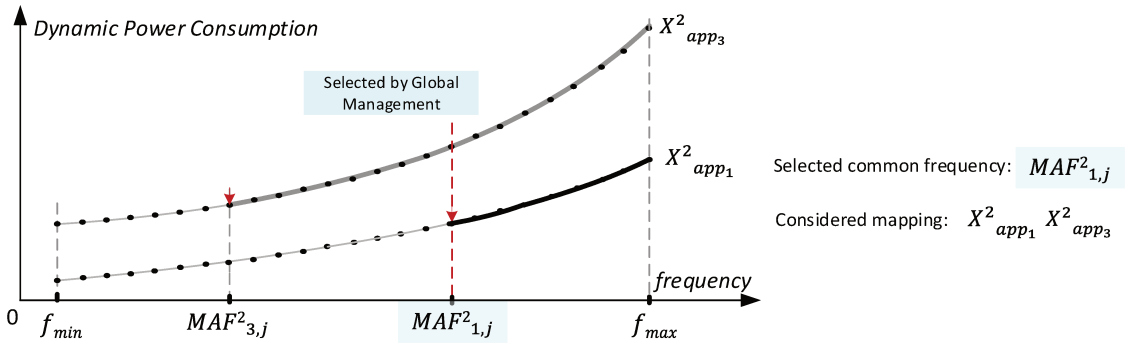

Figure 5.2: The global management selected frequency ($MAF_{1,j}$) for the active applications (app_1 and app_3) in a cluster.

In the following subsections, the 0-1 IP problem can be stated as input, variables, constraints and the objective for a set of active applications in any use-case.

5.3.1 Input

The inputs of the 0-1 IP formulation include the active applications in the current use-case, the design-time prepared data, and the performance/power characteristics of the cluster-based platform.

- **Active applications:** for the current use-case $u_m = \{app_1, app_2 \cdots, app_I\}$

  (1) I: the number of active applications

  (2) $Period_{app_i}$: the period of each active application

- **Design-Time Prepared Data:** for each application (app_i)

  (1) c_i: the number of used cores of the prepared execution trace ($X^{c_i}_{app_i}$)

  (2) $MAF^{c_i}_{i,r}$: the minimum allowed frequency of each application (app_i) executed on c_i cores in the reference cluster ($cluster_r$)

  (3) $CompTime_{h,i,r}(f_0)$: the computation time of each task ($t_{h,i}$) executed on $cluster_r$ at f_0 (see Eq.(3.10) in Chapter 3)

  (4) $\xi_{h,i,r}$: the power coefficient of each task ($t_{h,i}$) executed on $cluster_r$ (see Eq.(3.10) in Chapter 3)

- **Cluster-based multi-core/many-core platform:**

  (1) J: the number of different clusters

  (2) R^{power}_j: power ratio of each cluster ($cluster_j$) against the reference cluster (see Eq.(3.12))

  (3) R^{perf}_j: performance ratio of each cluster ($cluster_j$) against the reference cluster (see Eq.(3.11))

  (4) N_j: the number of cores in each cluster ($cluster_j$)

  (5) $\{f_{j,1}, f_{j,2}, \ldots, f_{j,max}\}$: the supported frequencies of each cluster ($cluster_j$).

5.3.2 Variables

The application-to-cluster assignment problem is represented as a matrix.

- **Application-to-cluster assignment matrix:** As an active application can be freely assigned or migrated among different clusters, we define a $(0, 1)$ variable $a_{i,j}$.

$$a_{i,j} = \begin{cases} 1, & \text{if } app_i \text{ is assigned onto } cluster_j. \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

5.3.3 Constraints

- **Application assignment constraints:** Each application must be assigned to a cluster in each use-case.

$$\sum_{j=1}^{J} a_{i,j} = 1, \forall i, a_{i,j} \in \{0, 1\} \tag{5.2}$$

- **Application timing constraints:** The timing constraints of active applications are guaranteed by cluster frequency configurations. In this work, the optimized frequency level of each cluster is estimated by the prepared MAFs of active application and can be expressed as follows.

$$f_j = \max\{MAF_{i,j}^{c_i} \times a_{i,j}\}, \forall i \tag{5.3}$$

where f_j is the optimized cluster frequency of $cluster_j$. $MAF_{i,j}^{c_i}$ is the minimum allowed frequency of app_i in $cluster_j$ with respect to its timing constraint. $MAF_{i,j}^{c_i}$ can be obtained through measurements in different clusters or can be estimated based on $MAF_{i,r}^{c_i}$ as in Eq.(5.4).

$$MAF_{i,j}^{c_i} = R_j^{perf} \times MAF_{i,r}^{c_i} \tag{5.4}$$

- **Cluster frequency constraints:** The selected frequency of each cluster should be within the frequency range of a cluster.

$$f_{j,1} <= f_j <= f_{j,max}, \forall j \tag{5.5}$$

- **Platform resource constraints:** In each cluster, the number of used cores (N_j^{used}) by the applications should not be larger than the number of cores in each cluster (N_j). Note that the N_j^{used} in each cluster depends on the applied local management strategy, which will be discussed in Section 5.4.3.

$$N_j >= N_j^{used} \tag{5.6}$$

5.3.4 Objective

The objective of the global management is to minimize the average power consumption of the system. This objective is expressed as Eq.(5.7) based on Eq.(3.14) (see Chapter 3), where P_{sys}^{avg} is related to the square of each cluster frequency (*i.e.*, f_j^2) and H denotes the number of tasks of each application. Since this work assumes that communication energy is much smaller than computation energy, communication energy is not taken into account. It is worth noticing that Eq.(5.7) will be used latter in the proposed global management strategies to estimate the average dynamic power of the overall system.

$$\begin{aligned}\min P_{sys}^{avg} &= \min \sum_{i=1}^{I} P_{app_i}^{avg} \\ &= \min_{\{f_j, a_{i,j}\}} \sum_{j=1}^{J} f_j^2 \sum_{i=1}^{I} a_{i,j} \times \frac{\sum_{h=1}^{H} (R_j^{power} \times \xi_{h,i,r} \times R_j^{perf} \times CompTime_{h,i,r}(f_0) \times f_0)}{Period_{app_i}}\end{aligned} \tag{5.7}$$

subject to Eq.(5.2), (5.3),(5.5) and (5.6).

5.3.5 Observations

Based on this 0-1 IP formulation, we have the two following observations.

- **Observation 1:** As each cluster frequency (f_j) is determined by the maximum $MAF_{i,r}^{c_i}$ of the applications in the same cluster, assigning applications with *close* $MAF_{i,r}^{c_i}$ in a cluster can lower the average power consumption ($P_{app_i}^{avg}$) of other applications (having lower $MAF_{i,r}^{c_i}$ in the same clusters). Thus the applications with *close* $MAF_{i,r}^{c_i}$ should be assigned to the same cluster to optimize the P_{avg}^{sys} in Eq.(5.7).

- **Observation 2:** For a use-case with only one application to an empty cluster-based platform (without any already executed application), the average dynamic power of the application can be written into Eq.(5.8) based on Eq.(5.7). In Eq.(5.8), let R_j^4 denote $R_j^{perf} \times R_j^{perf} \times R_j^{perf} \times R_j^{power}$. The minimum $P_{app_i}^{avg}$ (*i.e.*, P_{sys}^{avg} for one application) can be achieved in the cluster with the least R_j^4. Thus, the cluster with the least R_j^4 has the highest priority to be used. Note that R_j^4 only depends on core types of clusters. It means that homogeneous platforms have the same R_j^4 for all clusters, while heterogeneous

platforms have different R_j^4 for all clusters.

$$
\begin{aligned}
P_{app_i}^{avg} &= (MAF_{i,j}^{c_i})^2 \times \frac{\sum_{h=1}^{H}(R_j^{power} \times \xi_{h,i,r} \times R_j^{perf} \times CompTime_{h,i,r}(f_0) \times f_0)}{Period_{app_i}} \\
&= (MAF_{i,r}^{c_i})^2 \times (R_j^{perf})^2 \times \frac{\sum_{h=1}^{H}(R_j^{power} \times \xi_{h,i,r} \times R_j^{perf} \times CompTime_{h,i,r}(f_0) \times f_0)}{Period_{app_i}} \\
&= R_j^{power} \times (R_j^{perf})^3 \times (MAF_{i,r}^{c_i})^2 \times \frac{\sum_{h=1}^{H}(\xi_{h,i,r} \times CompTime_{h,i,r}(f_0) \times f_0)}{Period_{app_i}} \\
&= R_j^4 \times (MAF_{i,r}^{c_i})^2 \times \frac{\sum_{h=1}^{H}(\xi_{h,i,r} \times CompTime_{h,i,r}(f_0) \times f_0)}{Period_{app_i}}
\end{aligned}
\tag{5.8}
$$

5.4 Solution to the 0-1 IP optimization problem

This section presents two global management strategies to achieve the solution to the 0-1 IP optimization problem. As discussed in [4, 29], obtaining the optimal mapping is an NP-hard problem. To avoid time-consuming searches (*e.g.*, caused by exhaustive search), the two proposed global management strategies aim to achieve a near-optimal solution for the 0-1 IP problem. Our proposed global management strategies determine application-to-cluster assignments for the active applications in each *use-case*, and cluster frequencies are set under the guidance of MAFs to meet application time constraints (see Eq.(5.3)). Moreover, the global management strategies have to respect platform resource constraints (see Eq.(5.6)), while the resource usage of each cluster is dependent on the applied local management (task-to-core mapping) strategies (will be discussed in Section 5.4.3).

According to the previous observations (see Section 5.3.5), we propose two global management strategies. The first strategy *Neighboring Search Application-to-Cluster Assignment* (NSACA) considers assignments *holistically* for all active applications. The second strategy *Greedy Search Application-to-Cluster Assignment* (GSACA) considers assignments *individually* for each active application. Afterward, we discuss the impact of different local management strategies (due to different resource usages) on the global management decisions.

5.4.1 Neighboring Search Application-to-Cluster Assignment

The first global management strategy *Neighboring Search Application-to-Cluster Assignment* (NSACA) aims to achieve near-optimal energy-efficient application assignments on cluster-based systems. All active applications can be re-assigned in every use-case (*i.e.*, without considering the application assignments of the previous use-case). The NSACA strategy

considers application assignments *holistically*, where the platform resources are negotiated among all active applications.

The NSACA strategy starts by assigning applications with close $MAF_{i,r}^{c_i}$ in the same cluster initially. Then the application assignments are iteratively improved through the changes of application assignments to their neighboring clusters. Neighboring clusters refers to any two clusters whose cluster frequency configurations are relatively closer compared to other clusters. The NSACA strategy is presented in Algorithm 2. The algorithm takes design-time prepared data, active applications in the current use-case (u_m) and application timing constraints ($Period_{app_i}$) and platform resource constraints (*e.g.*, J clusters, N_j cores in a cluster, $f_{j,max}$) as inputs, and provides the energy-efficient application-to-cluster assignments and cluster frequency configurations. The algorithm consists of three basic stages.

Stage 1: Active Application Order

For a certain use-case, the first stage (line 1-3) of NSACA is to sort all the active applications according to the descending order of $MAF_{i,r}^{c_i}$. Then, the sorted applications are divided into several groups as averagely as possible based on the number of applications. The number of groups is set as the number of available clusters (*i.e.*, J) in this work. Each application group aims to be assigned to one cluster to avoid too many applications assigned to the same cluster. Otherwise, the frequency of one cluster can be very high and result in a high P_{sys}^{avg}. Note that assigning more applications in some clusters to shut down other idle clusters is beyond the scope of this work. Figure 5.3 gives an example of ordering 10 active applications in 4 groups.

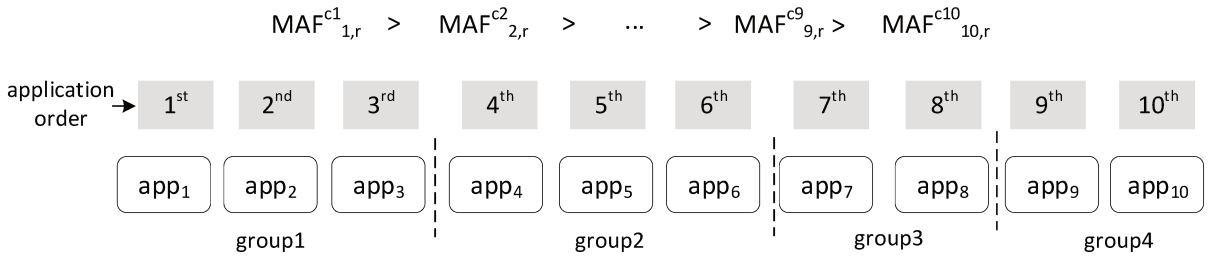

Figure 5.3: NSACA application order results for 10 active applications in 4 groups.

Stage 2: Initial Application Assignment

The second stage aims to assign each application group to a cluster under system constraints (*i.e.*, assignment, timing, frequency and resource constraints defined in Section 5.3.3). According to the *Observation 2*, the cluster with the least R_j^4 (on heterogeneous platforms) has the highest priority to be used. Thus, for each application group, we select an empty cluster with the lowest R_j^4 and satisfies the application timing constraints (line 6). Particularly for homogeneous platforms where all clusters have the same R_j^4, in such cases any empty cluster can be selected.

Algorithm 2: Neighboring Search Application-to-Cluster Assignment (NSACA) Strategy

Input: design-time prepared data, active applications in the current use-case, application timing constraints, cluster-based multi/many-core platform

Output: application-to-cluster assignments, cluster frequency configurations

```
1  //Stage 1: Active Application Order
2  Sort active applications in the descending order of MAF_{i,r}^{c_i};
3  Average I active applications into J groups;
4  //Stage 2: Application Assignment Initialization
5  for each application group do
6      Select an empty cluster with the lowest R_j^4 and f_{j,max} >= max{MAF_{i,j}^{c_i}}_{group};
7      for each app_i do
8          if N_s >= N_s^{used}, depending on Local Management Strategy then
9              Assign app_i to the selected cluster;
10         else
11             if the current group is not the last group then
12                 Move the current and remaining applications to the next group;
13             else
14                 Assign the current application to the used clusters by backtracking;
15                 Otherwise, application assignment fails;
16             end
17         end
18     end
19 end
20 //Stage 3: Application Assignment Improvement
21 Polish assignment by application switching in neighboring clusters;
22 for each used cluster_s do
23     Estimate P_{sys}^{avg} when app_i with the maximum MAF_{i,r}^{c_i} is moved to cluster_{s-1};
24     if meet system constraints && P_{sys}^{avg} is lower then
25         Move app_i to cluster_{s-1};
26     end
27     if fail to move app_i to cluster_{s-1} then
28         Estimate P_{sys}^{avg} when app_i with the minimum MAF_{i,r}^{c_i} is moved to cluster_{s+1};
29         if meet system constraints && P_{sys}^{avg} is lower then
30             Move app_i to cluster_{s+1};
31         end
32     end
33 end
```

Then, with the selected cluster, applications in the group are assigned *successively* according to the order obtained in Stage 1. Note that we do not assign all applications in a group *directly* to a cluster, in case the selected cluster cannot provide enough cores for the grouped applications. The number of applications in each group can be seen as the maximum number of applications that can be assigned to the same cluster. The NSACA assigns each application according to the resource availability of the cluster (line 7-19). The resource availability can be checked through estimations or feedback from the applied *local management strategy* (see Eq.(5.6)), which will be further discussed in Section 5.4.3. If the available cores of the selected cluster are not enough, the remaining applications in the same group are incorporated into the next group (line 11-12), and then it goes to the assignment of the next application group. On the other hand, if there is no more group left, each remaining application searches for an available used cluster by backtracking, from the latter selected clusters to the first selected one until a possible cluster is found (line 13- 14). This is because the remaining applications have close $MAF_{i,r}^{c_i}$ with the applications in the latter selected clusters. The application assignment fails when there is no available cluster for the remaining unassigned applications.

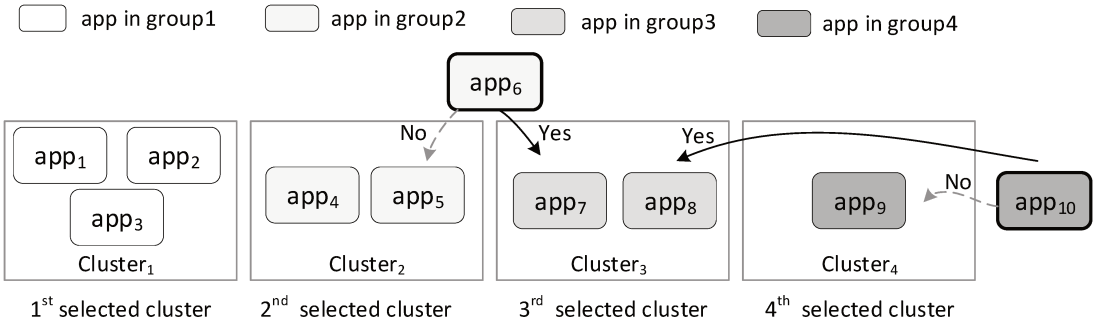

Figure 5.4: NSACA initial application assignment result for 10 active applications in 4 clusters.

Figure 5.4 gives one possible initial application-to-cluster assignment solution corresponding to Figure 5.3. Applications in $group1$ are assigned to the first selected cluster ($cluster_1$). Note that the heterogeneity of the platform is considered when selecting a cluster for each application group based on R_j^4. For the applications in $group2$, the second selected cluster ($cluster_2$) cannot provide enough available resources. Thus, app_6 is incorporated to the applications in $group3$ and then assigned to $cluster_3$. Applications in $group4$ are assigned to $cluster_4$, which cannot provide enough resources for app_{10}. As there is no more unassigned application group or available cluster, NSACA checks for available resources in the used clusters and finds a possible solution for app_{10} in $cluster_3$. Otherwise, the application assignments in this use-case would have failures.

Stage 3: Application Assignment Improvement

Based on the initial assignments, the third stage of NSACA tries to further reduce P_{sys}^{avg} by *application switching*[11] and *application moving*[12] between neighboring clusters.

The *application switching* (line 21) happens when platform resources are not enough for all active applications. In such cases, application orders (*i.e.*, according to $MAF_{i,r}^{c_i}$ criteria) cannot be maintained due to the *backtracking* processing in Stage 2. For the example in Figure 5.4, app_{10} is assigned to $cluster_3$, while its previous application app_9 is assigned to its next neighboring cluster $cluster_4$. Better solution might be to assign app_9 onto $cluster_3$ and app_{10} onto $cluster_4$, as it strictly keeps applications with close $MAF_{i,r}^{c_i}$ in the same cluster. For such cases, NSACA checks the assigned applications in two neighboring clusters. When the last-order application in a cluster has larger $MAF_{i,r}^{c_i}$ than the first-order application in the next neighboring cluster (*e.g.*, app_{10} in $cluster_3$ has larger $MAF_{i,r}^{c_i}$ than app_9 in $cluster_4$ in Figure 5.4), the NSACA estimates P_{sys}^{avg} after switching the application assignments. If P_{sys}^{avg} is reduced compared to that before application switching and the system constraints are still met, the assignment of the two applications are switched. Figure 5.5 shows the result of switching the assignments of app_9 and app_{10}.

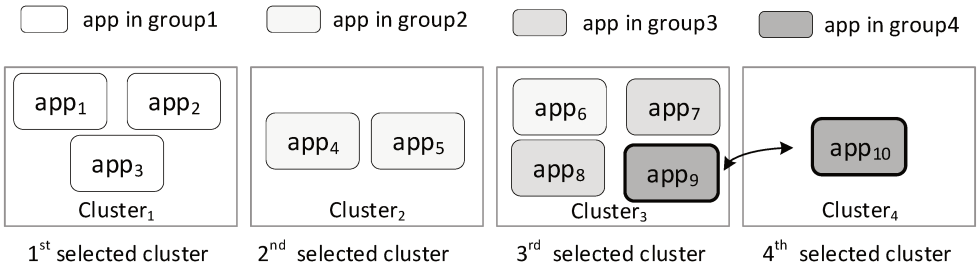

Figure 5.5: NSACA application switching result for 10 applications in 4 clusters.

Afterwards, the *application moving* is performed to move applications from a cluster to its neighboring clusters to further reduce P_{sys}^{avg} (line 22-33). For each used cluster, the application with the maximum $MAF_{i,r}^{c_i}$ is attempted to move to its previous neighboring cluster, if P_{sys}^{avg} can be reduced and the system constraints can be met. When one application is moved successfully, the next application with the maximum $MAF_{i,r}^{c_i}$ is considered until no reduction of average system power is observed. When there is no application moved to the previous cluster, the application with the minimum $MAF_{i,r}^{c_i}$ is attempted to move to its next neighboring cluster. The application moving finishes when all the used clusters are checked. Figure 5.6 gives an example of moving an application to its neighboring cluster. In this example, app_3 is first moved from $cluster_1$ to $cluster_2$, with respect to the moving conditions (line 24 or 29). Then, it goes to check the move of the application (app_2) with the next minimum $MAF_{i,r}^{c_i}$ in the same

[11] Application switching: exchange the assignments of two applications in two different clusters.
[12] Application moving: change the assignment of an application from one cluster to another.

cluster. Note that the order of applications based on $MAF_{i,r}^{c_i}$ is already known in Stage 1 (see Figure 5.3). If app_2 moving is not acceptable, it goes to check the application moving between $cluster_2$ and $cluster_3$ and so on.

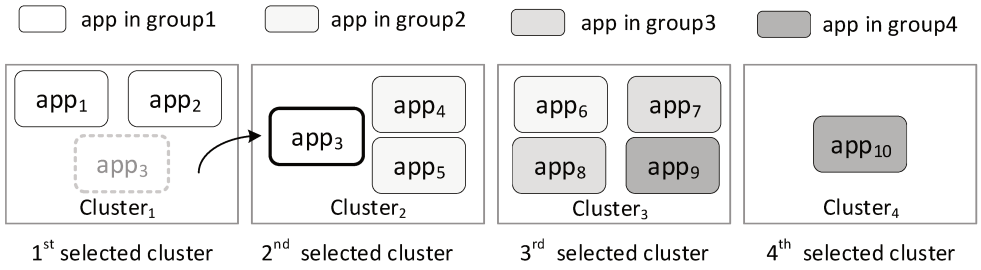

Figure 5.6: NSACA application moving result for 10 applications in 4 clusters.

NSACA assigns applications with close $MAF_{i,r}^{c_i}$ to the same cluster. In this work, *close* is a relative concept, meaning the applications in the same cluster have closer $MAF_{i,r}^{c_i}$ than the applications in two different clusters. NSACA performs further optimization of application assignment between two neighboring clusters. Since some applications might be active in several consecutive use-cases, re-assigning all the active applications in each use-case can cost high overhead.

5.4.2 Greedy Search Application-to-Cluster Assignment

To avoid excessive application migrations, the second global management strategy, *Greedy Application-to-Cluster Assignment* (GSACA), is proposed to assign only newly activated applications and migrate the old existing applications in a small extent. The number of migrations can be controlled by the user. Migration is a processing that migrate processing code and data [79] from a source to a destination. The migration process can take some time and energy, in order to obtain a better energy-efficient mapping. It is important to weigh the benefits and costs of migration to make a wise migration decision. For analysis purpose, our work focuses on migration benefits, assuming that migration overheads can be neglected compared to benefits. Besides, migration is often performed at predefined checkpoints at which is it safe to migrate the task [79]. We assume that migration is performed at the end of application periods to avoid migrating intermediate task states.

This GSACA strategy considers application assignments *individually*. When accomplishing the assignment for one application, it goes to consider a next application. GSACA assigns each newly active application to a cluster that can lead to the minimum estimated P_{sys}^{avg}. The GSACA strategy is presented in Algorithm 3, with the same inputs and outputs as NSACA. The GSACA strategy consists in three basic stages.

Algorithm 3: Greedy Search Application-to-Cluster Assignment (GSACA) Strategy

Input: design-time prepared data, active applications in the current use-case, application timing constraints, cluster-based multi/many-core platform

Output: application-to-cluster assignments, cluster frequency configurations

```
1  //Stage 1: New application order
2  Average I newly active applications into J groups, in decreasing order of MAF_{i,r}^{c_i};
3  Order newly active applications in a particular way;
4  //Stage 2: New application assignment
5  for each new app_i or released old app_i do
6      Select clusters that meet f_{j,max} >= MAF_{i,j}^{c_i} and N_j >= N_j^{used};
7      if several potentially selected empty clusters then
8          Only keep the one with the minimum R_j^4;
9      end
10     for each selected cluster cluster_s do
11         if the first cluster_s or P_{sys}^{avg} decreases then
12             if N_s >= N_s^{used}, depending on Local Management Strategy then
13                 Assign app_i to cluster_s;
14                 Update frequency level of cluster_s;
15             end
16         end
17     end
18     if No solution can be found for new app_i then
19         Release an already assigned new application that has lower MAF_{i,r}^{c_i} and can
             provide available cores for the currently considered new application;
20         Otherwise, application assignment fails;
21     end
22 end
23 //Stage 3: Old application migration
24 Initialize further allowed migration Nm = 0;
25 if Nm <= Allow_{migration} then
26     Assign the old unmigrated application with the minimum MAF_{i,r} (line 6-17);
27     Nm++;
28 end
```

Stage 1: New Application Order

In the first stage, newly active applications are ordered in a particular way (line 1-3). In our ordering approach, the newly active applications are divided into several groups according to descending order of $MAF_{i,r}^{c_i}$. Here, GSACA aims to divide the newly applications into J (*i.e.*, the number of clusters) groups averagely based on the number of newly active applications. Then, GSACA takes turns to index applications in different groups according to descending

order and ascending order of $MAF_{i,r}^{c_i}$ alternatively. Figure 5.7 gives one example of ordering 7 newly active applications in 4 groups.

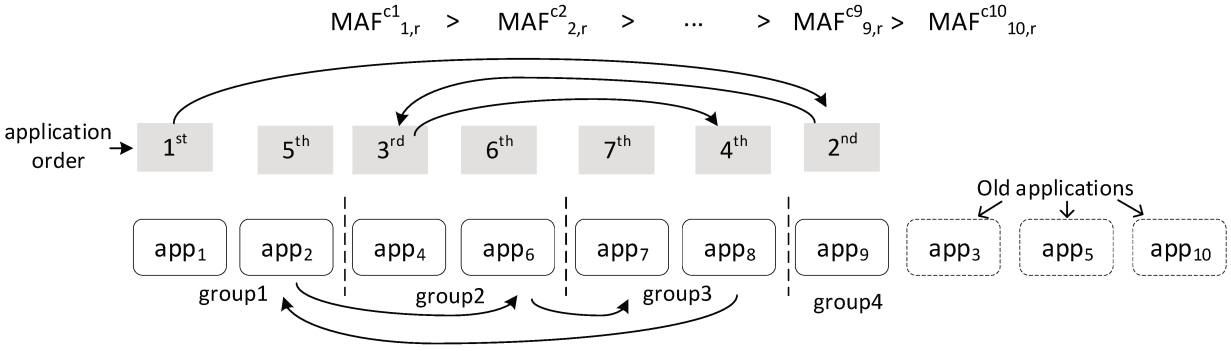

Figure 5.7: GSACA order result for 7 newly active applications in 4 groups.

In this example use-case, there are 10 active applications in total, and only the newly active 7 applications are ordered. The first indexed application is the one with the maximum $MAF_{i,r}^{c_i}$ in the first group ($group1$), the second is the one with the minimum $MAF_{i,r}^{c_i}$ in the last group ($group4$). The third is the one with the maximum $MAF_{i,r}^{c_i}$ in the next first group ($group2$), and the fourth is the one with the minimum $MAF_{i,r}^{c_i}$ in the next last group ($group3$). After all the groups have been considered once, it goes back to $group1$ and reorders the remaining applications.

It is worth noting that our particular application-ordering approach is to avoid successively assigning applications with close $MAF_{i,r}^{c_i}$. Otherwise, applications with close $MAF_{i,r}^{c_i}$ can be separated into different clusters, resulting in high P_{sys}^{avg}. Our application-ordering approach takes turns to assign applications with relatively far $MAF_{i,r}^{c_i}$ (in different groups) and achieves good results in our experimental evaluations (see Section 5.5). In the future, we can consider other ordering approaches (*e.g.*, random).

Stage 2: New Application Assignment

According to the obtained application orders, the second stage assigns each new application to an appropriate cluster (line 4-20). Each application aims to be assigned to the cluster that leads to the minimum P_{sys}^{avg} based on the current system state.

The potentially selected clusters include clusters (used or empty) that meet the application timing constraint (*i.e.*, $f_{j,max} >= MAF_{i,j}^{c_i}$) and meet the cluster resource constraint (*i.e.*, $N_j >= N_j^{used}$). The resource availability can be checked through estimations or feedback from the applied *local management strategy*, which will be further discussed in Section 5.4.3. When there are several possible selected empty clusters, only the one with the minimum R_j^4 is chosen for further evaluation. This is because clusters with lower R_j^4 have higher priority to be used (see Observation 2). Figure 5.8 gives an example of cluster selections for app_4. In the example, the

old existing applications are kept in the same cluster as its previous use-case. After the first two indexed applications (app_1 and app_3) are assigned, it searches for possible clusters for the third indexed application app_4. As $cluster_2$ cannot provide enough resources, the remaining used cluster ($cluster_1$) and the empty cluster ($cluster_3$) with the lowest R_3^4 are further evaluated.

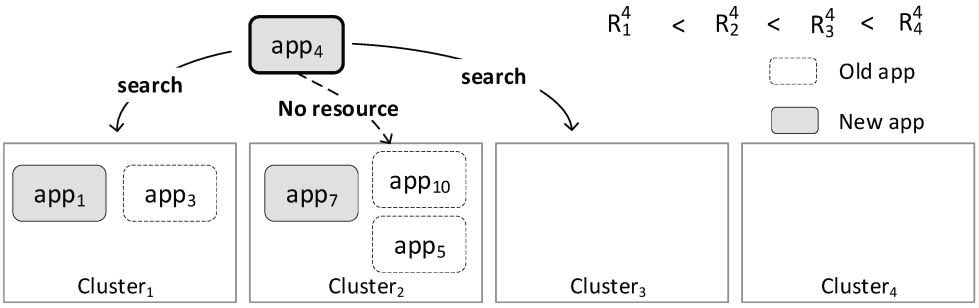

Figure 5.8: GSACA cluster selection for app_4 in 4 clusters.

With all the selected potential clusters for a new application, it evaluates different assignments and chooses the one that leads to the minimum P_{sys}^{avg} (line 10-17). However, it is possible that no cluster can be found for a new application since the clusters that meet the application timing constraint are occupied by other applications. In this case, GSACA checks already assigned new applications that have lower $MAF_{i,r}^{c_i}$ than that of the currently considered new application, in order to release some available cores. The released application will be reassigned to another cluster by greedy searching (line 18-21). If no new application can be released, the application assignment fails.

Stage 3: Old Existing Application Migration

After all new applications are assigned to clusters, the third stage is to further optimize P_{sys}^{avg} by migrating some old existing applications from one cluster to another (line 23-28). Some migrations can reduce P_{sys}^{avg} at the expense of increased migration overhead. The number of further allowed migrations ($Allow_{migration}$) can be set by users. For each migration, any old application can be migrated to another cluster by the greedy search. In this work, we choose to migrate the old application with the minimum $MAF_{i,r}^{c_i}$ first. In future study, we can give higher migration priority to old applications according to different characteristics (*e.g.*, application workload, the number lasting use-cases without changing its assignment).

5.4.3 The Impact of Local Management on Global Management

The global management determines the application-to-cluster assignment and the cluster frequency configuration. For the assigned applications in each cluster, the local management determines task-to-core allocation and scheduling. On one hand, the output of the global

management is the input of the local management. On the other hand, the global management decision depends on the local management in terms of core utilization of each cluster (*i.e.*, line 8 of NSACA algorithm, line 12 of GSACA algorithm).

In the local management, different task-to-core mapping strategies can be adopted. In this work, we consider FCFS and GAPVC application mapping combination strategies. As previously discussed in Chapter 4 (Section 4.4.3), FCFS and GAPVC perform task-to-core mapping without degrading application performance. Thus, cluster frequencies set by the global management can be respected.

In terms of strategy implementation, we use different methods for FCFS and GAPVC to coordinate with a global management strategy. Since FCFS directly applies the prepared mappings of active applications, the total number of used cores in a cluster is the sum of the used cores of the prepared mappings ($N_j^u = \sum_{i=1}^{I} c_i \times a_{i,j}, \forall i$, c_i is the number of used cores of the prepared mapping for app_i). Therefore, it is not necessary to execute FCFS to estimate core usage in a cluster. On the other hand, as GAPVC allows tasks of different applications to be allocated onto the same core, it is possible to use fewer cores than FCFS within a cluster (previously shown in Section 4.5.2). In this case, we cannot accurately estimate core usages without actually executing GAPVC in a cluster. In the work of this chapter, we execute GAPVC in each cluster when the cores used for the prepared mappings are not too much larger than the available cores (*i.e.*, $0 < \sum_{i=1}^{I} c_i \times a_{i,j} - N_j <= 3$). Here, 3 is set arbitrarily and this value can be changed by users according to the expected core savings of GAPVC (w.r.t. FCFS) in each cluster. In the cases when applications compete for limited platform resources, the less resource usage of GAPVC allows more applications to be assigned to more energy-efficient clusters (*i.e.*, due to heterogeneous cluster or low cluster frequency). As a result, lower P_{sys}^{avg} can be achieved in the overall system. However, more computations can be spent to integrate GAPVC in the hierarchical management. Future work can be considered to predict the used cores within a cluster without actually performing GAPVC.

5.5 Experimental Evaluations

In the experimental evaluations, our proposed management algorithms are coded in C++ on Visual Studio. To go beyond the exiting small-sized heterogeneous cluster-based platforms (*i.e.*, ARM big.LITTLE, Exynos 5422 [21]), we used 4 types of ARM-Cortex processors to compose platforms with different numbers of clusters and different numbers of cores within the cluster. Based on different resource constraints, we can evaluate our management strategies on heterogeneous and homogeneous multi/many-core platforms.

Table 5.1 depicts the physical characteristics of four commercial processor models (*e.g.*, Cortex-A9, A15, A7, A17) that are considered in our experiments. The data of performance ratio (R_j^{perf} defined in Eq.(3.11) and power ratio (R_j^{power} defined in Eq.(3.12)) normalized to Cortex-A9 are obtained from [25]. The frequency ranges of processors are set differently with

$0.1GHz$ increments in the evaluations. The four processor models are used to create different clusters. In our experimental evaluations, heterogeneous platforms consist of different clusters (from $cluster_1$ to $cluster_8$), while homogeneous platforms consist of several $cluster_1$.

Table 5.1: Platform settings for 8 considered clusters

Processor	R_j^{perf}	R_j^{power}	$f_{min}(GHz)$	$f_{max}(GHz)$	$f_{step}(GHz)$	$cluster_j$
Cortex-A9	1	1	0.4	2.0	0.1	$cluster_1, cluster_5$
Cortex-A15	0.625	2.25	0.4	2.0	0.1	$cluster_2, cluster_6$
Cortex-A7	1.25	0.55	0.4	1.4	0.1	$cluster_3, cluster_7$
Cortex-A17	0.645	1.3	0.4	1.8	0.1	$cluster_4, cluster_8$

The simulated applications (app_1 to app_{10}) are defined in Table 5.2. They are derived from reference (H263 encoder and H263 and JPEG decoders) applications with different input/output tokens, based on the descriptions provided in SDF3 [67]. Note that the applications are the same with the simulated applications in the previous chapter (Chapter 4, Table 4.1). The table gives the number of used cores MAF values (obtained in the reference cluster Cortex-A9) of the design-time prepared mapping of each application. In this work, we have evaluated optimized mappings having different trade-offs between core usages and application performance at design-time, but only the mapping having the best application performance is stored in the design-time database.

Table 5.3 summaries the strategies that are considered in the global management and in the local management. Firstly, we compare five global management strategies that consider the assignments of all active applications in each use-case, regardless of whether any applications already existed in the previous use-case. *Exhaustive* strategy evaluates all possible application-to-cluster assignments and apply the solution with the lowest P_{sys}^{avg} at scaled cluster frequencies satisfying the system constraints. *LPF* [49] and *LEF* [77] strategies assign all active applications to the clusters respectively with the lowest power consumption (*i.e.*, Cortex-A7 due to the minimum R_j^{power}) and the lowest energy consumption (*i.e.*, Cortex-A17 due to the minimum R_j^4) first. When the cluster with the lowest power/energy consumption is not possible to accept a new application, the cluster with the next lowest power/energy consumption is considered. As previously introduced, *NSACA* and *GSACA* are our proposed global management strategies. *GSACA** is a modified version of *GSACA*, which considers the assignments of all active applications in a use-case allowing all migrations. *GSACA** considers each application assignment *individually*, serving as the counterpart of *NSACA* (*holistically*). Secondly, we consider global management strategies that assign only the newly active applications (in each use-case) with limited migrations. *Exhaustive_M0* and *GSACA_M0* are two strategies that only assigns newly active applications in a use-case forbidding any migration. *GSACA_M1*, *GSACA_M2* and *GSACA_M3* refer to the proposed Greedy Search strategies that allows different

Table 5.2: Design-time prepared information of 10 considered applications

Application				**Prepared Mapping**	
Type	app_i	**Nb of tokens of each task**	**Period** (μs)	**Nb of used cores** c_i	**MAF** (GHz) **in Cortex-A9**
H26 decoder :4 tasks :3 edges	app_1	$\{1, 6, 6, 1\}$	60	2	1.0
	app_2	$\{1, 4, 4, 1\}$	180	2	0.4
	app_3	$\{1, 264, 264, 1\}$	360	2	0.8
H263 encoder :5 tasks :4 edges	app_4	$\{1, 5, 5, 5, 1\}$	540	2	1.2
	app_5	$\{1, 15, 15, 15, 1\}$	1080	2	0.7
	app_6	$\{1, 45, 45, 45, 1\}$	1080	2	1.1
JPEG decoder :6 tasks :5 edges	app_7	$\{1, 7, 7, 7, 7, 1\}$	180	4	1.0
	app_8	$\{1, 9, 9, 9, 9, 1\}$	360	4	0.6
	app_9	$\{1, 22, 22, 22, 22, 1\}$	1080	4	0.4
	app_{10}	$\{1, 12, 12, 12, 12, 1\}$	180	4	1.3

Table 5.3: Considered Management Strategies for Comparison

Strategies	**Abbreviation**	**Management in each use-case**
Global Management (allow all migrations)	Exhaustive	Exhaustive strategy that considers assignments for all applications
	LPF	Low-Power-First [49] strategy that considers assignments for all applications
	LEF	Low-Energy-First [77] strategy that considers assignments for all applications
	NSACA	Proposed NSACA that considers assignments for all applications *holistically*
	GSACA*	Modified GSACA that considers assignments for all applications *individually*
Global Management (allow limited migrations)	Exhaustive_M0	Exhaustive strategy assigns only new applications and allows 0 migration per use-case
	GSACA_M0	Proposed GSACA that assigns only new applications and allows 0 migration per use-case
	GSACA_M1	Proposed GSACA that assigns only new applications and allows 1 migration per use-case
	GSACA_M2	Proposed GSACA that assigns only new applications and allows 2 migration per use-case
	GSACA_M3	Proposed GSACA that assigns only new applications and allows 3 migration per use-case
Local Management	FCFS	First-Come-First-Served [12, 13] application mappings combination strategy
	GAPVC	Proposed application mappings combination strategy in Chapter 4

numbers of migrations (*i.e.*, $Allow_{migration} = 1$, or 2, or 3, see Algorithm 3) in a use-case. Here, we set $Allow_{migration}$ to specific numbers to evaluate the benefit of application migrations. It would also be possible to set $Allow_{migration}$ as the percentage of the active applications. Note that we do not consider LPF and LEF with limited migrations, as migrations do not change the limitations of these two strategies (due to giving the highest priority to one cluster during application assignment, which will be discussed in Section 5.5.1). Finally, as the global management decisions can be different due to different applied local management strategies in each cluster. We consider two task-to-core mapping strategies, FCFS [12, 13] and GAPVC (proposed in Chapter 4), to analyze the benefit of using fewer cores in a cluster. These two local management strategies do not degrade application performance, thus avoiding changing

the cluster frequency of global management settings.

5.5.1 Evaluations of Global Management Strategies

(1) Global Management Allowing All Migrations

In the first experimental evaluation, we compare global management strategies that allow all migrations, assuming FCFS is locally applied to achieve task-to-core mapping in each cluster. The evaluations are performed among all the possible (*i.e.*,1023) use-cases for the 10 considered applications, under different platform constraints. We consider homogeneous and heterogeneous platforms of different platform sizes (*i.e.*, meaning the number of clusters $\times$ the number of cores inside each cluster). In each simulated use-case, we obtain the P_{sys}^{avg} of different global management strategies. The P_{sys}^{avg} values are normalized to the result of the *Exhaustive* strategy under the constraint of *heterogeneous 8 clusters* $\times$ *8 cores*. The average normalized P_{sys}^{avg} of the 1023 use-cases are shown in Figure 5.9. Notice that P_{sys}^{avg} refers to the average system power after migration, while migration overhead is not taken into account in the experimental comparison.

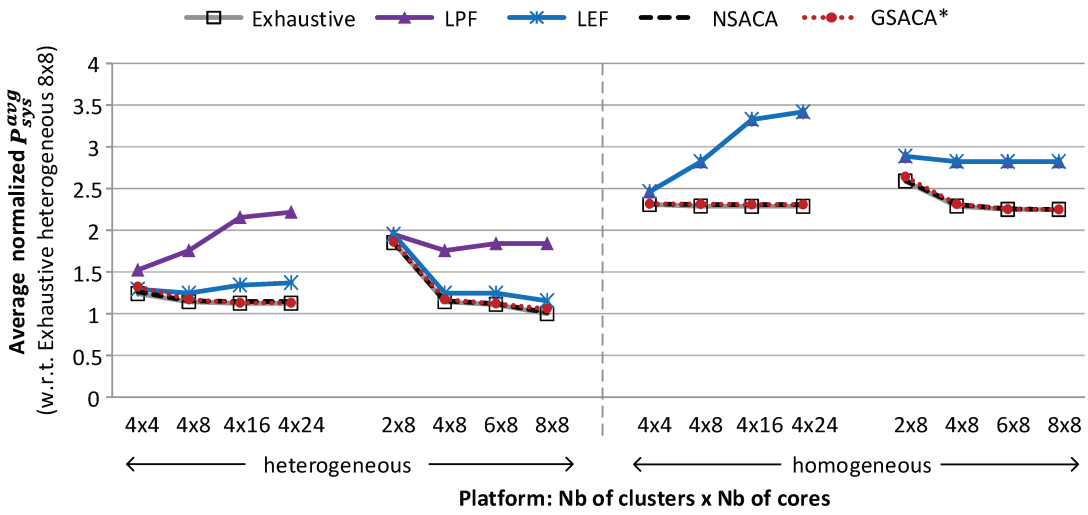

Figure 5.9: Average normalized P_{sys}^{avg} with respect to Exhaustive (constrained by the heterogeneous 8$\times$8 platform) among the 1023 use-cases. Results are given for different platform constraints.

From the *Exhaustive* results, we can observe that the optimal P_{sys}^{avg} highly depends on the platform constraints. The P_{sys}^{avg} values of *Exhaustive* are given in Table 5.4. With more cores in a cluster in heterogeneous platforms, reduced P_{sys}^{avg} can be achieved (*e.g.*, $P_{sys}^{avg} = 1.240$ on the 4$\times$4 heterogeneous platform, $P_{sys}^{avg} = 1.144$ on the 4$\times$8 heterogeneous platform) as more applications are allowed to be assigned in a more energy efficient cluster (due to heterogeneous

Table 5.4: Normalized P_{sys}^{avg} of Exhaustive compared to the four different strategies

Platform	Criteria	4x4	4x8	4x16	4x24	2x8	4x8	6x8	8x8
Heterogeneous	Exhaustive Value	1.240	1.144	1.124	1.124	1.850	1.144	1.110	1.000
	ΔLPF[13]	22.96%	53.57%	91.78%	97.28%	5.46%	53.57%	65.93%	84.17%
	ΔLEF	4.38%	8.96%	19.52%	21.96%	5.46%	8.96%	12.27%	15.49%
	ΔNSACA	1.87%	0.94%	1.93%	1.93%	0.02%	0.94%	1.01%	1.35%
	ΔGSACA*	6.84%	2.24%	0.51%	0.51%	0.67%	2.24%	1.10%	5.96%
Homogeneous	Exhaustive Value	2.309	2.287	2.286	2.286	2.588	2.287	2.250	2.248
	ΔLPF	6.64%	23.46%	45.58%	49.58%	11.59%	23.46%	25.53%	25.62%
	ΔLEF	6.64%	23.46%	45.58%	49.58%	11.59%	23.46%	25.53%	25.62%
	ΔNSACA	0.19%	1.03%	1.00%	1.00%	0.18%	1.03%	0.34%	0.02%
	ΔGSACA*	0.11%	1.21%	1.06%	1.06%	2.19%	1.21%	0.17%	0.01%

[13] ΔStrategy=$\frac{P_{sys}^{avg}(Strategy)-P_{sys}^{avg}(Exhaustive)}{P_{sys}^{avg}(Exhaustive)}$: the P_{sys}^{avg} difference between the considered strategy (*i.e.*, LPF, LEF, NSACA or GSACA*) and Exhaustive.

clusters or low frequency configurations). On the other hand, more clusters can also lead to reduced P_{sys}^{avg} in the heterogeneous platforms (*e.g.*, $P_{sys}^{avg} = 1.850$ on the 2×8 heterogeneous platform, $P_{sys}^{avg} = 1.144$ on the 4×8 heterogeneous platform). As applications with different $MAF_{i,r}^{c_i}$ can be separated to more different clusters, lower cluster frequencies can be achieved for less P_{sys}^{avg}. The trend of more platform resources (*e.g.*, more clusters and more cores within the cluster) leading to reduced P_{sys}^{avg} can also be observed for homogeneous platforms in Table 5.4. However, the results for homogeneous platforms is not as obvious as heterogeneous platforms, because heterogeneous platforms can be more energy efficient.

The results obtained for *LPF* and *LEF* are very different from the optimal ones (*Exhaustive*), as shown in Figure 5.9. *LPF* and *LEF* give different assignment priorities to clusters. They might achieve close results with the optimal ones in small platforms (*e.g.*, 4×4 and 2×8 platforms). For example, LEF has only 4.38% higher P_{sys}^{avg} than *Exhaustive* on the 4×4 heterogeneous platforms. However, with more cores in a cluster, more applications are assigned in one cluster while leaving other clusters empty without any assigned application. In such cases, the frequency of one cluster can be increased and consequently increases P_{sys}^{avg} of the system. On the 4 clusters × 24 cores heterogeneous platform, *LPF* and *LEF* respectively have 97.28% and 21.96% higher P_{sys}^{avg} than *Exhaustive*.

Furthermore, Figure 5.9 shows that *NSACA* achieves very close results to the optimal ones under different platform constraints. The maximum difference between *NSACA* and *Exhaustive* are is 1.93% on heterogeneous platforms and about 1.03% on homogeneous platforms (see Table 5.4). *NSACA* achieves near-optimal global management results. It assigns applications with close $MAF_{i,r}^{c_i}$ to the same cluster and gives higher priorities to clusters with lower $MAF_{i,r}^{c_i}$. *NSACA* considers assignments holistically for all active applications. For *GSACA** that considers application assignments individually, it also achieves near-optimal results. The

maximum difference between *GSACA** and *Exhaustive* is about 6.84% on heterogeneous platforms and about 2.19% on homogeneous platforms (see Table 5.4). This confirms that our greedy strategy based on a particular application assignment order is able to assign applications with close $MAF_{i,r}^{c_i}$ to the same cluster.

(2) Global Management Allowing Limited Migrations

The second experimental evaluation is to compare global management strategies that allow limited migrations. It is assumed that FCFS is locally applied in each cluster to combine application mappings. The evaluations are performed for 1023 use-cases that are generated in random sequences. The obtained P_{sys}^{avg} in each use-case is normalized to the *Exhaustive* strategy under different platform constraints. The average P_{sys}^{avg} for the 1023 use-cases are shown in Figure 5.10. The P_{sys}^{avg} values obtained by different strategies are normalized to the result of the Exhaustive strategy constrained by *each platform size*. The reason for doing this is to show details of the strategy differences, not the impact of platform size. On the other hand, the strategies compared in this evaluation have different numbers of migrations across all the 1023 use-cases. The average number of migrations per use-case of each strategy is shown in Figure 5.11.

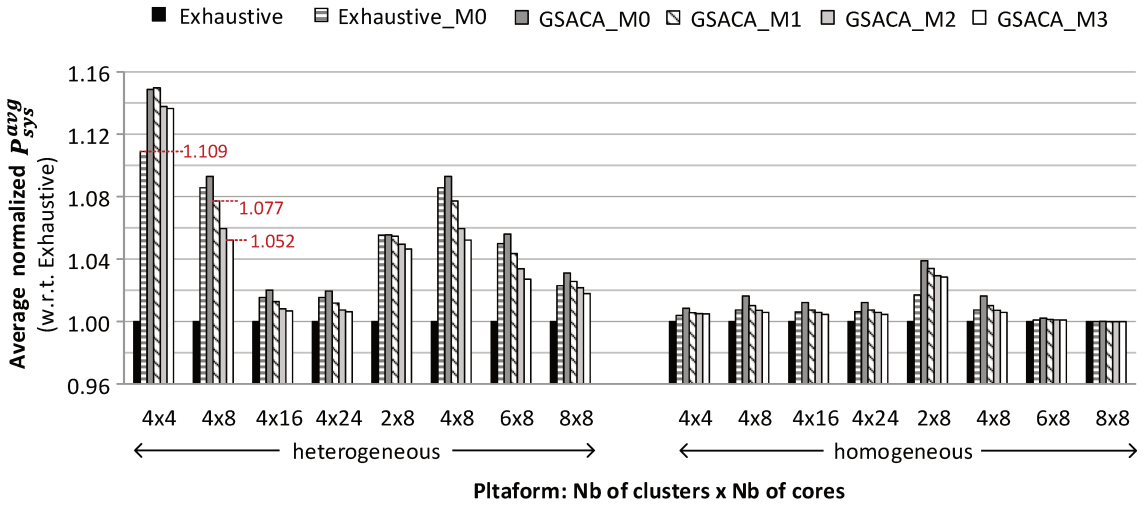

Figure 5.10: Average P_{sys}^{avg} normalized to Exhaustive (constrained by each platform size) for the 1023 use-cases.

Exhaustive achieves the optimal P_{sys}^{avg} in each use-case at the cost of large numbers of migrations (up to 1.82 in Figure 5.11). As *Exhaustive_M0* only assigns newly active applications without any migration, it results in larger P_{sys}^{avg} than *Exhaustive* (*e.g.*, 10.9% on the 4×4 heterogeneous platform in Figure 5.10) due to its fewer assignment possibilities. *GSACA_M0* also prohibits migrations and it achieves close results compared to *Exhaustive_M0*. Considering the proposed *GSACA* (*i.e.*, M1,M2,M3) that allows different numbers of migrations in each use-

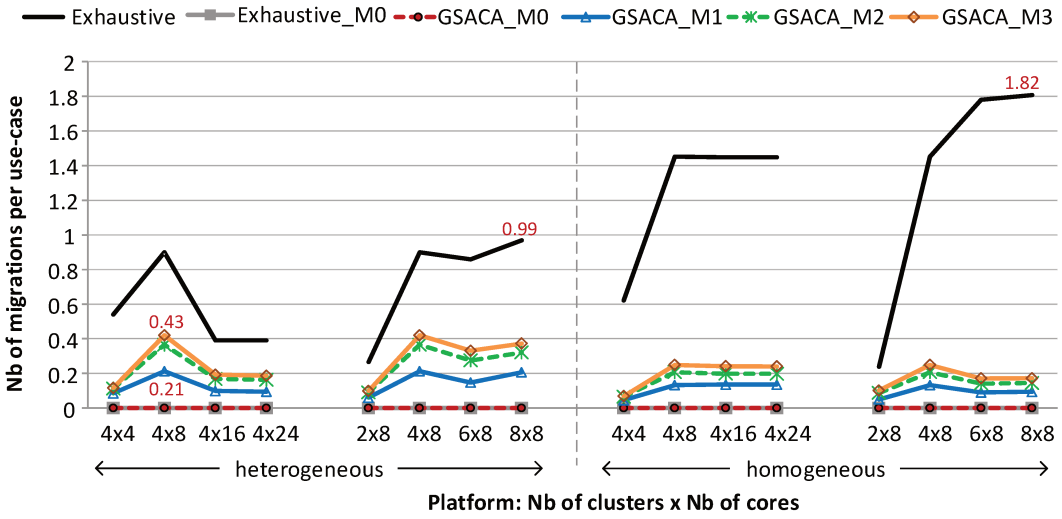

Figure 5.11: The average number of migrations in a use-case (among the 1023 use-cases) of the considered applications. Results are given for different platform constraints.

case, we can observe that more migrations can decrease P_{sys}^{avg}. As an example, we compare *GSACA_M1* and *GSACA_M3* on the 4×8 heterogeneous platform. It can be observed that P_{sys}^{avg} decreases from 1.077 to 1.052 (*i.e.*, 2.5%, see Figure 5.10), while migrations per use-case increase from 0.21 to 0.43 (*i.e.*, 0.22 more migration, see Figure 5.11).

In this work, P_{sys}^{avg} does not take migration overheads into account. Whether an application should be migrated depends not only on the saving P_{sys}^{avg} and the migration overheads, but also on the active duration of the current use-case. It is deserved to sacrifice some migration overheads to achieve a low P_{sys}^{avg} for a long time. As a consequence, the energy saving after migration (in a long time) can be larger than the migration costs. Future work should estimate migration costs and the active duration of the current use-case in order to provide a more accurate evaluation of the proposed strategies.

(3) Strategy Complexity of Global Management

The third experimental evaluation is to compare the complexity of global management strategies, assuming FCFS is locally applied in each cluster. For this purpose, we measure the simulation time of each use-case (among the 1023 use-cases). The average simulation time used per use-case is shown in Figure 5.12. The simulation is performed for different global management strategies in Visual Studio running on a laptop (Intel Core i5 processor, 16.0GB).

Firstly, we compare the first five global management strategies (*i.e.*, Exhaustive, LPF, LEF, NSACA and GSACA*) that allow all migrations. It can be observed that LPF and LEF has the minimum simulation time (less than $0.011ms$ in 4-cluster platforms). However, as previously discussed, these two strategies are limited in energy efficiency on platforms with more clusters

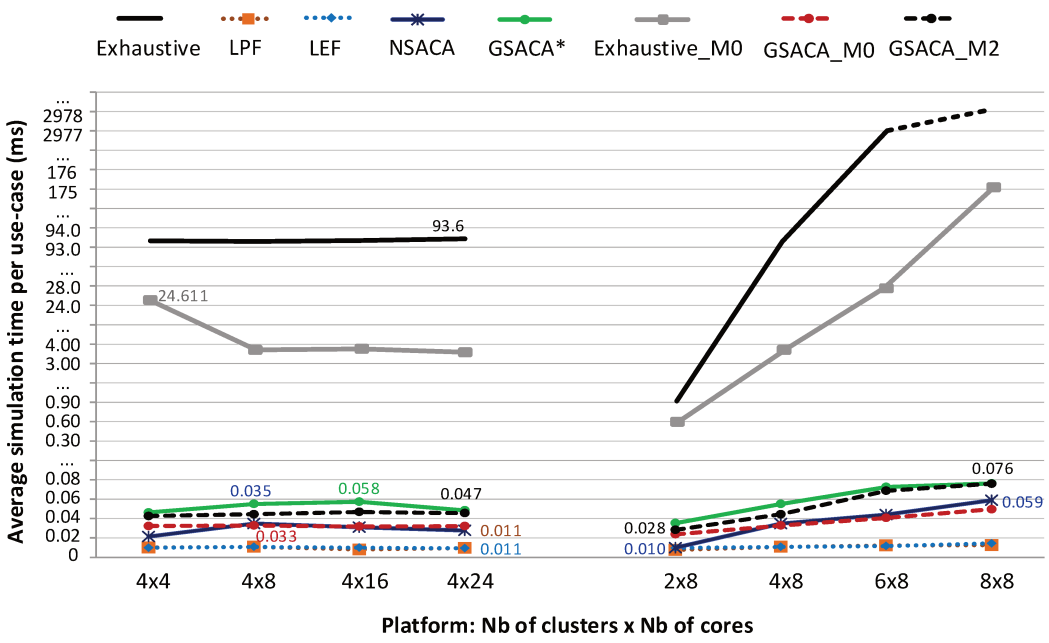

Figure 5.12: The average simulation time (ms) used per use-case (among the 1023 use-case). Results are given for heterogeneous platform constrained by different platform sizes.

and more cores in each cluster. Compared to Exhaustive, the proposed NSACA and GSACA* use much less simulation time. For the 4-cluster platforms, Exhaustive uses about $93.6ms$ for simulation, while NSACA and GSACA* use less than $0.035ms$ and less than $0.058ms$ respectively. That means NSACA and GSACA* respectively run 2674 times and 1614 times faster than Exhaustive. Whereas, as shown in Table 5.4, NSACA and GSACA* only have 4.38% and 6.84% higher P_{sys}^{avg} than Exhaustive.

Then, we consider the last three strategies (Exhaustive_M0, GSACA_M0 and GSACA_M2) that allow limited migrations. For clarity, we do not present the simulation time of GSACA_M1 and GSACA_M3. It can be observed that the proposed GSACA_M0 and GSACA_M2 (up to $0.033ms$ and $0.047ms$ respectively on 4-cluster platforms) use less simulation than Exhaustive_M0 (up to $24.611ms$ on 4-cluster platforms). That means GSACA_M0 and GSACA_M2 can be 746 times and 524 times faster than Exhaustive_M0. Particularly, GSACA_M2 (allows 2 migrations per use-case) takes more time for simulation than GSACA_M0 (allows 0 migration per use-case). It indicates that more time is spent exploring an energy-efficient assignment for each migrated application.

Finally, we can also observe that simulation time does not change much with the number of cores, but increases with the number of clusters. For instance, from 2×8 to 8×8 platforms, the simulation time for NSACA increases from $0.010ms$ to $0.059ms$, while the simulation time for GSACA_M2 increases from $0.028ms$ to $0.076ms$.

5.5.2 Evaluation the Influences of Local Management Strategies

In the last experiment, we compare FCFS and GAPVC to evaluate their influences on the global management decisions. As previously discussed in Chapter 4 (Section 4.4.3), GAPVC can use fewer cores than FCFS to map application tasks onto cores within a cluster. The comparison results for four different global management strategies (*i.e.*, Exhaustive, NSACA, Exhaustive_M0, GSACA_M2) based on the two local strategies are shown in Table 5.5. Note that this evaluation does not take LPF and LEF into account due to their limitation in energy efficiency on platforms with more clusters and more cores in each cluster.

Table 5.5 summarizes the improvement of GAPVC compared to FCFS among the 1023 use-cases. First, we compare the failed use-cases (see column (1)) when using the two different local management strategies. Some use-cases fail to achieve a possible mapping solution due to insufficient platform resources for all the active applications. The column (1.1) lists the number of failed use-cases when using FCFS under the constraints of different platform sizes. For example, *Exhaustive+FCFS* has 273 failures on both 4x4 and 2x8 heterogeneous platforms. Because GAPVC uses fewer cores to achieve a combined mapping then FCFS, GAPVC has more potential to handle use-cases with more active applications and consequently results in fewer failures. The column (1.2) shows the number of reduced failed use-cases achieved by GAPVC (w.r.t. FCFS). Compared to *Exhaustive+FCFS*, *Exhaustive+GAPVC* has 66 (*i.e.*, $\frac{66}{273} = 24.18\%$) and 71 (*i.e.*, $\frac{71}{273} = 26.01\%$) fewer failures on 4x4 and 2x8 heterogeneous platforms respectively. Note that when platform resources are sufficient (*e.g.*, 4x8, 6x8 platforms), no failed use-case can be observed.

Then, we compare FCFS and GAPVC in their common feasible use-cases (see columns (2)), where both local management strategies can achieve a feasible solution. Among all the common feasible use-cases (see column (2.1)), we list the number of use-cases where GAPVC can achieve lower P_{sys}^{avg} in the column (2.2). The maximum P_{sys}^{avg} reduction of GAPVC is shown in the column (2.3). It can be observed that GAPVC can achieve lower P_{sys}^{avg} than FCFS in some use-cases. For the 4x4 heterogeneous platform, *Exhaustive+GAPVC* achieves lower P_{sys}^{avg} than *Exhaustive+FCFS* in 58 use-cases (over 750 common feasible use-cases). Among the 58 use-cases, the maximum reduction of P_{sys}^{avg} is 5.73%. This is because the less core usage allows GAPVC assigns more applications to more efficient clusters (*e.g.*, due to heterogeneous cluster or low cluster frequency). Particularly, more significant P_{sys}^{avg} reduction can be observed for *Exhaustive_M0+GAPVC* (up to 57.65% w.r.t.*Exhaustive_M0+FCFS*) and *GSACA_M2+GAPVC* (up to 41.87% w.r.t.*GSACA_M2+FCFS*). The two strategies only considers newly active applications and allows limited migrations (*i.e.*, 0 and 2 respectively) per use-cases. Since GAPVC uses fewer cores for the mapping of the existing old applications, it allows more assignment possibilities for newly active applications and consequently achieves lower P_{sys}^{avg}. Note that when platform resources are quite enough, each global management strategy can achieve the same P_{sys}^{avg} by using FCFS or GAPVC in the local management. Such observations can be seen for the Exhaustive and NSACA global management strategies on 4x8 and 6x8

Table 5.5: Comparison of FCFS and GAPVC local strategies in hierarchical management among the 1023 use-cases

	Global strategy	Local strategy / platform	(1) Among Failed u_m		(2) Among common feasible u_m [15]			
			(1.1) Nb of failed u_m of FCFS	(1.2) Reduced failed u_m of GAPVC w.r.t. FCFS [14]	(2.1) Nb of common feasible u_m	Improvement of GAPVC w.r.t. FCFS		
						(2.2) Nb of u_m with less P_{avg}^{sys}	(2.3) Max P_{sys}^{avg} reduction	(2.4) Total reduced cores
Heterogeneous	**Exhaustive**	**4x4**	273	66	750	58	5.73%	117
		2x8	273	71	750	15	3.65%	253
		4x8	0	0	1023	0	0	143
		6x8	0	0	1023	0	0	104
	NSACA	**4x4**	322	24	701	6	9.50%	12
		2x8	318	21	705	8	1.53%	217
		4x8	0	0	1023	0	0	147
		6x8	0	0	1023	0	0	148
	Exhaustive_M0	**4x4**	300	55	723	68	57.65%	145
		2x8	279	63	744	37	32.91%	315
		4x8	0	0	1023	12	21.85%	147
		6x8	0	0	1023	11	21.14%	121
		4x24	0	0	1023	0	0	287
	GSACA_M2	**4x4**	310	47	713	61	41.87%	106
		2x8	290	47	733	39	33.74%	292
		4x8	0	0	1023	16	25.62%	174
		6x8	0	0	1023	11	29.79%	148
		4x24	0	0	1023	0	0	278
Homogeneous	**Exhaustive**	**4x4**	273	66	750	24	2.82%	58
		2x8	273	71	750	19	2.24%	288
		4x8	0	0	1023	0	0	102
		6x8	0	0	1023	0	0	19
	NSACA	**4x4**	322	24	701	12	2.25%	24
		2x8	318	21	705	16	6.64%	235
		4x8	0	0	1023	0	0	60
		6x8	0	0	1023	0	0	0
	Exhaustive_M0	**4x4**	295	53	728	38	12.16%	80
		2x8	280	61	743	40	25.97%	301
		4x8	0	0	1023	0	0	124
		6x8	0	0	1023	0	0	26
	GSACA_M2	**4x4**	323	27	700	29	8.36%	106
		2x8	286	47	737	48	25.97%	323
		4x8	0	0	1023	1	6.71%	121
		6x8	0	0	1023	0	0	15

[14] Reduced failed u_m of GAPVC w.r.t. FCFS = Nb of failed u_m of FCFS - Nb of failed u_m of GAPVC.
[15] Common feasible use-cases: the use-cases where both FCFS and GAPVC achieve a mapping result.

platforms, and also for the Exhaustive_M0 and GSACA_M2 global management strategies on 4x24 heterogeneous platforms. These observations suggest that with sufficient platform resources, our considered global management strategies can have the same application-to-cluster assignments by using FCFS and GAPVC in the local management.

We also compare the total number of reduced cores in the common feasible use-cases (see column (2.4)) for the two local management strategies. For instance, on the 4x4 heterogeneous platform, *Exhaustive+GAPVC* uses 117 fewer cores than *Exhaustive+FCFS* over the 750 common feasible use-cases. On the heterogeneous 4x8 and 6x8 platforms, *Exhaustive+GAPVC* still uses fewer cores (up to 143) than *Exhaustive+FCFS* even though there is no difference of P_{sys}^{avg} between the two strategies. This indicates that for the same application-to-cluster assignments (due to the same P_{sys}^{avg} values), GAPVC can use fewer cores than FCFS to achieve a combined mapping in each cluster.

To sum up, GAPVC can lead to less resource usage within a cluster than FCFS. In particular, when platform resources are insufficient, GAPVC can result in fewer failed use-cases and lead to lower P_{sys}^{avg}. On the other hand, when platform resources are sufficient, our considered global management strategies can achieve the same P_{sys}^{avg} results (*i.e.*, indicating the same application-to-cluster assignments). In such cases, using GAPVC in the local management can still reduce the number of used cores in each cluster. These observations can be seen for different global management strategies when managing both heterogeneous and homogeneous platforms.

However, using GAPVC in the local management can lead to more computation than FCFS. Because GAPVC has to be actually executed in order to accurately estimate the core usages within a cluster (see Section 5.4.3). Consequently, It requires more calculations within a cluster to provide feedback (*e.g.*, the number of cores used) for global management. Here, we measure the simulation time of each use-case (among the 1023 use-cases) when using FCFS and GAPVC in local management. The average simulation time used per use-case is shown in Table 5.6.

Table 5.6: Comparison of average time (ms) used to simulate a use-case for FCFS and GAPVC in hierarchical management

Strategy	**Platform**	**Heterogeneous**			**Homogeneous**		
		FCFS (ms)	**GAPVC** (ms)	**Time difference**[16]	**FCFS** (ms)	**GAPVC** (ms)	**Time difference**
NSACA	4x4	0.021	0.373	16.384	0.019	0.265	12.591
	2x8	0.010	0.275	26.698	0.011	0.268	24.103
	4x8	0.035	0.061	0.754	0.029	0.032	0.102
	6x8	0.044	0.109	1.463	0.041	0.044	0.065
GSACA_M2	4x4	0.043	0.557	12.040	0.041	0.637	14.478
	2x8	0.028	0.178	5.278	0.291	0.176	5.056
	4x8	0.044	0.175	2.926	0.048	0.084	0.730
	6x8	0.069	0.161	1.336	0.069	0.076	0.113

[16] Time difference: $\Delta t = \frac{t_{GAPVC} - t_{FCFS}}{t_{FCFS}}$

Due to the scalability problem of Exhaustive and Exhaustive_M0, we just consider NSACA and GSACA_M2 global management strategies in the comparison. The table indicates that more simulation time is used per use-case for the global management strategies combining GAPVC (*i.e.*, NSACA+GAPVC, GSACA_M2+GAPVC). When using GAPVC in the local management, we can observe the mapping time of the hierarchical management has increased by 26.7 times (NSACA on the 2x8 heterogeneous platform). Moreover, compared to small-sized platforms (*i.e.*, 4x4, 2x8), the obtained time difference of big-sized platforms (*i.e.*, 4x8, 6x8) can be smaller. For NSACA, the time difference decreases to 0.754 on the 4x8 heterogeneous platform. As big-sized platforms reduce the situations where active applications compete for limited resources (*i.e.*, when $0 < \sum_{i=1}^{I} c_i \times a_{i,j} - N_j <= 3$), GAPVC is less executed.

The results in Table 5.6 suggest that when platform resources are competitive for all active applications, using GAPVC in the local management will cost more time to obtain management results. Spending more time on searching for management solutions also means more energy consumption. However, it is possible that no solution (or high P_{sys}^{avg} results) can be found without using GAPVC (or other local management strategies using fewer cores to get a mapping). The future work will further weigh the gains and losses of energy efficiency when using the GAPVC or other local management strategies.

5.6 Summary and Discussion

This chapter considers the global optimization of the average dynamic power consumption of cluster-based multi/many-core systems. For this purpose, hierarchical management is employed to deliver global optimization among clusters and local optimization inside each cluster.

For the global management, we presented an 0-1 IP formulation that integrates application-to-cluster assignments and cluster frequency configurations, under application timing, platform resource and frequency constraints. To achieve near-optimal solutions of the 0-1 IP formulation in fast speeds, we propose two global management strategies. The first global management strategy NSACA considers the assignments of all active applications (in a use-case) holistically, allowing all possible migrations. The NSACA strategy can achieve near-optimal results. It assigns applications with close $MAF_{i,r}^{c_i}$ to the same clusters and iteratively improves the assignments through moving applications between two neighboring clusters. Our experimental evaluation shows that the average power consumption achieved by NSACA is only 1.93% worse than the optimal solution (*i.e.*, by Exhaustive search), but the speed of NSACA is 2674 times faster. The second global management strategy GSACA considers assignments for applications individually. It greedily finds the most energy-efficient application-to-cluster assignment for each newly active application, and only migrates some old executing applications for further optimizations. The GSACA strategy allows users to control the number of migrations per use-case. When assuming migration overhead is 0, more migrations can lead to reduced average power consumption. Our experimental evaluation indicates that 0.22 more migration (*i.e.*, the

number of migrated applications) per use-case in GSACA can lead to 2.5% reduction of the average power consumption of the overall system.

In particular, our GSACA global strategy allows migrating some applications/tasks from one cluster to another to achieve lower P_{sys}^{avg}. The existing migration mechanisms include *process recreation* and *process replication* [79]. During task migration, process recreation kills a process (*e.g.*, task) in the source (*e.g.*, core or cluster) and then creates a new process in the destination. Process recreation has high migration overheads as it not only migrates some data/states (*e.g.*, stack, an contents of internal registers [80]), but also migrates the process code. To avoid migrating process code, process replication keeps process copies in different clusters/cores. When a process is migrated, it suspends the process in the source and then restarts the process in the destination. Compared with process recreation, process replication has less migration data, but it consumes more memory space to store process copies. The selection of migration mechanism depends on the characteristics of the platform resources (*e.g.*, available memory space, communication speed). To reduce migration costs, we can just migrate the applications without heavy migration code/data. Our future work should take into account migration overheads to provide a more accurate evaluation of the proposed strategies.

For the local management in each cluster, FCFS and GAPVC mapping combination strategies are applied to determine task-to-core mappings. Due to the less resource usage of GAPVC, it reduces the number of used core inside a cluster, making more applications assigned to more efficient clusters. Based on the same global management strategies, our experiments have shown that GAPVC can reduce system energy efficiency (*i.e.*, P_{sys}^{avg}) by up to 57.65% (the maximum P_{sys}^{avg} reduction in a use-case) compared to FCFS. However, GAPVC costs more computations to explore resource-efficient task mapping results. When platform resources are insufficient, using GAPVC in local management can take up to 26.7 times longer to obtain the management solution of a use-case. This is because we have to actually execute GAPVC to estimate the used cores within each cluster. The future work could consider predicting the used core of GAPVC without executing the strategy.

In this work, only one optimized mapping is prepared for each application at design-time. Since applications can be assigned to different clusters, each prepared mapping can be adjusted according to the impact of different core types. For multiple applications executed on a cluster-based multi/many-core platform, the reduction of cluster frequencies is mainly realized by assigning applications with close required frequencies in the same cluster. The average dynamic power of the overall system can be further reduced if more design-time mappings are prepared. In this case, resource usage of each cluster can be further reduced by appropriately selecting a prepared mapping for each application in the same cluster. Multiple prepared mappings for each application will be considered in further works.

Chapter 6

System-Level Evaluation Approach of Run-Time Management Strategies

Contents

6.1 Overview

Due to high-performance requirements and the increasing application dynamism on nowadays multi/many-core systems, run-time management strategies have been proposed to favor the achievement of non-functional requirements such as timing and power constraints of systems. To guarantee the non-functional requirements to be respected, extensive evaluation of run-time management strategies is imperative. In this chapter, we focus on introducing a system-level simulation approach to support run-time management strategy evaluations.

System-level modeling and simulation approaches favor early detection of potential issues and prevent costly design cycles. In existing system-level simulation-based approaches, a system model is formed by a combination of an application model and a platform model. Then these models can be simulated, as executable descriptions, under different situations to estimate system performance and optimize system design. However, in most of the existing frameworks, the mapping of applications on the platform resources is statically defined and cannot be modified during the simulation. If the application mapping has to be adapted to system dynamism, the simulation should be stopped and restarted each time when the mapping is modified. To allow more efficient evaluation of run-time management strategies, extending system-level simulation-based approaches is thus mandatory.

The main contributions of this chapter are as follows.

- We present a new system-level modeling and simulation approach to allow evaluations of run-time management strategies on multi/many-core systems. The proposed approach dynamically computes the instants when platform resources are used by the running applications. Using the computed simulation instants, the simulation model of the run-time manager controls both the order of task execution and the advancement of simulation time.

- We implement and validate the proposed approach using Intel Cofluent Studio modeling framework [39] and SystemC simulation language [81]. Through a case-study that considers seven applications (85 tasks in total) running on a heterogeneous cluster-based platform, the proposed approach demonstrates the abilities to evaluate different management strategies.

6.2 Comparison with Existing Trace-driven Simulation

As previously discussed in Chapter 2, the work of [10] presents an extension of the Sesame framework to facilitate system-level modeling and simulation of multiple applications dynamically executed on a multi-core platform. The work is based on the trace-driven simulation approach [62, 63]. In this chapter, a new simulation approach that applies a different principle to adapt to the application executions is presented. The differences between the trace-driven

approach and our proposed approach are compared in Figure 6.1. The most significant differences are highlighted in different colours (*i.e.*, red for the trace-driven approach, blue for the proposed approach).

In part (a) of the figure, a 3-task application changes its mapping from (a.1) to (a.2) (see the dashed lines) onto a heterogeneous system with two cores. Note that when the application mapping is dynamically adapted, the executions of application tasks are expected to be dynamically controlled with the advancement of simulation time. Part (b) presents the application execution results of the trace-driven simulation approach. The simulation is performed based on the execution order of *application events*, including task computation

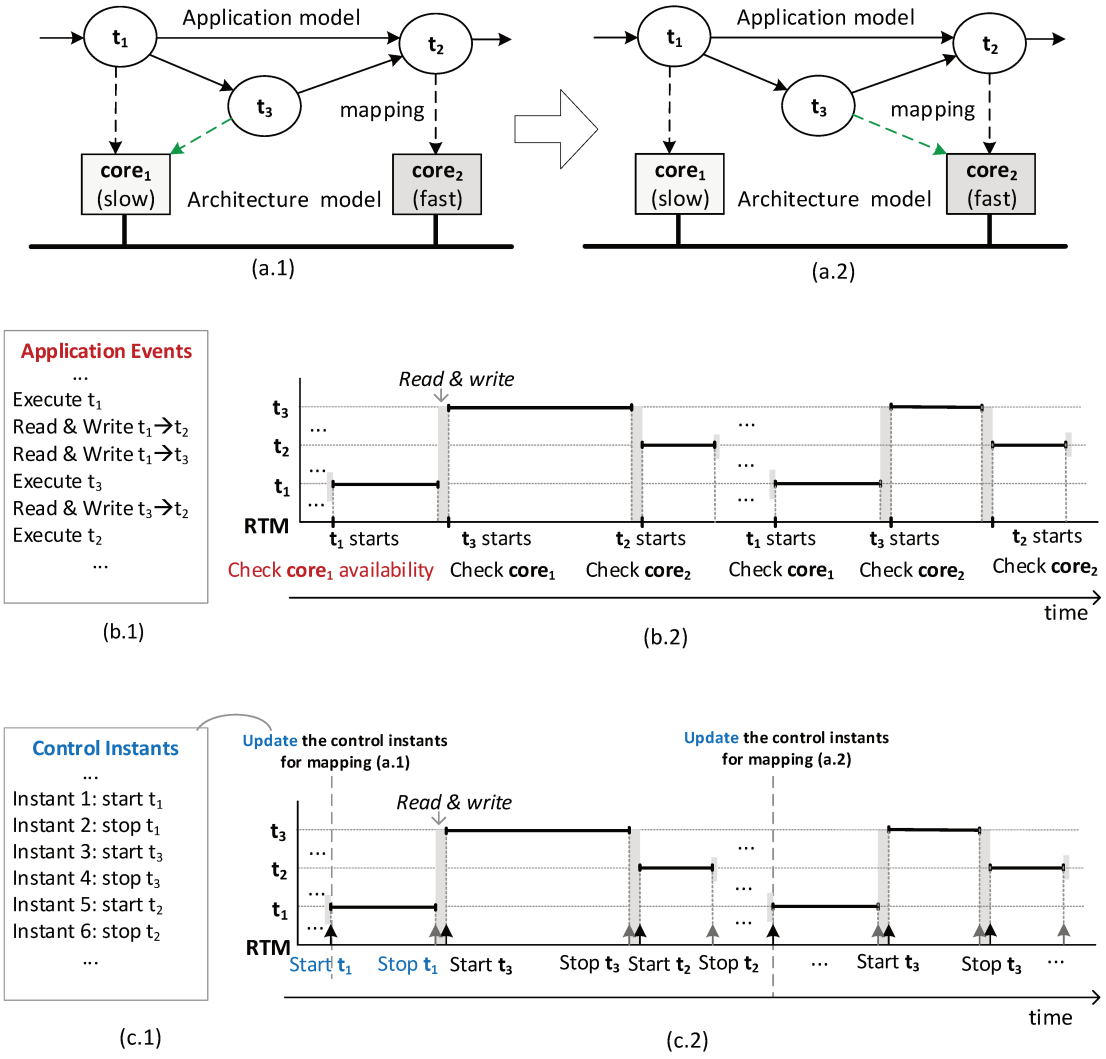

Figure 6.1: (a) Two mappings of 3-task application onto 2 cores; Dynamic task execution of (b) trace-driven simulation approach and (c) the proposed simulation approach.

events (*e.g.*, t_1 in part (b.1)) and communication events (*e.g.*, read and write in part (b.1)). When a new application event is identified during simulation, the Run-Time Manager (RTM) first checks the *availability* of the allocated hardware resource (*e.g.*, $core_1$) and then dispatches the event onto the targeted resource. It means that some synchronizations have to be done to ensure the availability of the allocated hardware resources during event dispatching. As the simulation time advances, different application events are dispatched and the performance characteristics of the application can be evaluated. When the application mapping changes, the application events are re-dispatched and the application performance characteristics are adapted accordingly.

On the other hand, figure (c) presents the simulated result of our proposed simulation approach. For a given mapping (*e.g.*, mapping (a.1)), our approach first *computes the instants* when platform resources are used by applications (*e.g.*, see (c.1)). Then the RTM *controls (starts or stops) the task execution* at the different computed instants. When the application adapts its mapping (*e.g.*, mapping (a.2)), our approach *updates the task instants* and then controls the task executions accordingly. Compared to the trace-driven simulation shown in Figure 6.1 (b), our proposed simulation approach does not dispatch application events and avoids model synchronization by computing the instants when platform resources are used. With the knowledge of the computed instants, our proposed approach controls when application tasks are run on platform resources.

6.3 Proposed Modeling and Simulation Approach

This section presents our simulation approach referring to the application and platform model presented in Chapter 3. Each application is characterized by its computation and communication behaviors through a SDF model. As we consider a dynamic execution scenario of multiple applications, the Use-case Definition module is introduced to define the set of successive use-cases (with different active applications) and their execution durations. The platform model is characterized by the hardware resources and their non-functional parameters (*i.e.*, computation time, communication time, power), in order to evaluate the performance consequences of each task/edge executed on different core types and at different frequencies. In addition, management components are introduced to support dynamic mappings of multiple applications onto platform resources in varying use-cases. The management components are used to execute and evaluate a certain run-time management strategy. Like the state-of-the-art simulation approaches [10, 11] for run-time management, our approach is also based on a design-time database.

As illustrated in Figure 6.2, we consider run-time management in three steps: **(1)** a design-time preparation, where one or several mappings for each application are prepared and stored in a database. **(2)** A run-time mapping process, performed when a new use-case is detected (*Use-case Detection*). In this step, a run-time mapping is established based on a particular

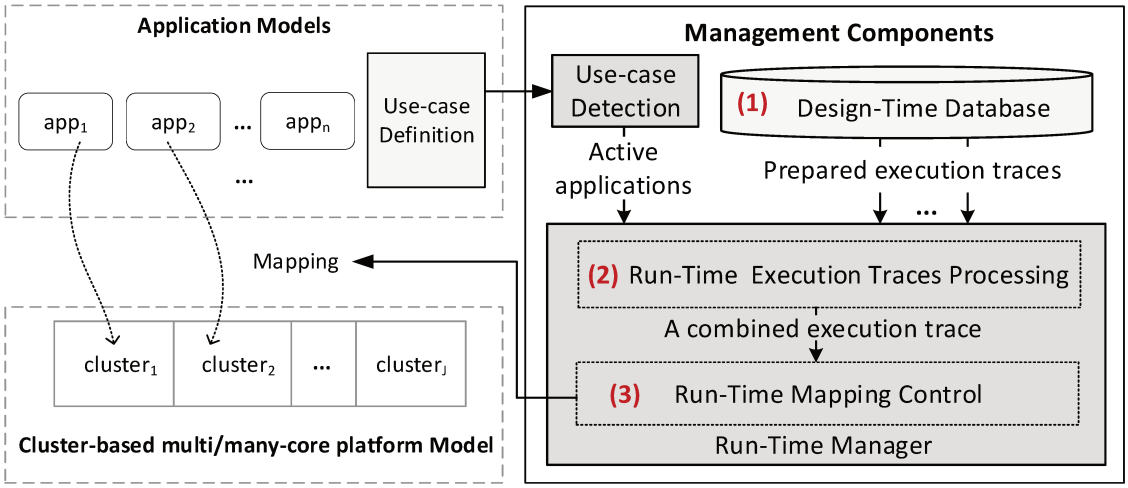

Figure 6.2: The modeled system with application, platform and management components descriptions.

algorithm (under evaluation) and, on the analysis of design-time execution traces of every active application in the use-case. Note that the proposed run-time management strategies in previous chapters (*i.e.*, Chapter 4 and 5) can be implemented in the **(1)** and **(2)** steps. Finally, **(3)** the run-time mapping control, which introduces *the new simulation approach* to control the execution of active tasks during system simulation based on the mapping established in the previous step.

6.3.1 Design-Time Database preparation

In the step of design-time preparation, a set of prepared mappings, one or several for each application is stored in a database. As previously discussed in Chapter 2, a mapping is characterized by its execution trace, *i.e*, a set of instants defining the start (x_s) and end (x_e) times of each task when executed on a specific platform configuration (processing element, $v/f, \cdots$). Only the instants within an application period ($Period_{app_i}$) are prepared for a design-time mapping.

Without loss of generality, lets consider that each application is prepared with one design-time execution trace, where each task is mapped onto one distinct core. Figure 6.3 shows the design-time execution traces X_{app_1} and X_{app_2} for app_1 and app_2 respectively. Note that the characterization of each design-time execution trace is the same as in the previous chapters. But the simulation approach presented in this chapter expresses the instants of each design-time execution trace in a particular way.

In our proposed simulation approach, instants x_s and x_e are expressed according to dependencies between tasks. For the example of X_{app_1} in Figure 6.3.(a), dependencies of task $t_{2,1}$ for instance, are expressed as follows: $x_{s\_t_{2,1}}(1) = x_{e\_t_{1,1}}(1) + CommTime_{e_{1,1}}(1)$,

$x_{e\_t_{2,1}}(1) = x_{s\_t_{2,1}}(1) + CompTime_{t_{2,1}}(1)$, $x_{s\_t_{2,1}}(2) = x_{e\_t_{2,1}}(1) + CommTime_{e_{2,1}}(1)$ and so on. It has to be noticed that the instants here are relative since $CompTime_{t_{h,i}}(k)$ and $CommTime_{e_{g,i}}(k)$ will depend on the real mapping determined at run-time in the next step.

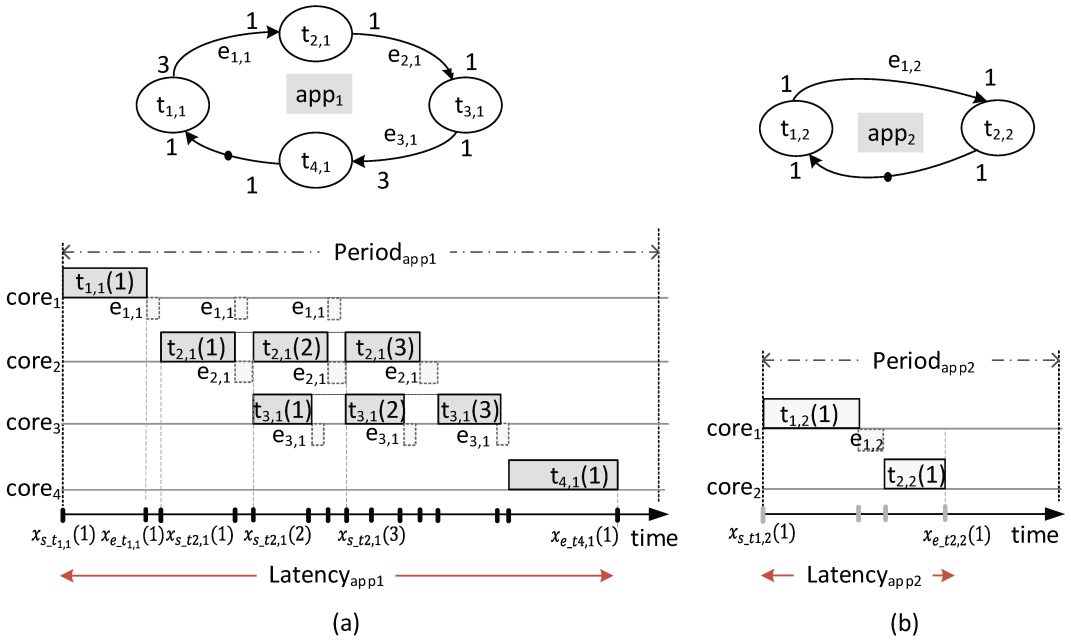

Figure 6.3: A design-time prepared execution trace for the mapping of app_1 (a) and app_2 (b).

6.3.2 Run-Time Execution Traces Processing

The second step concerns the processing of run-time execution traces. This step is performed each time a new use-case u_m is detected. The objective is to obtain at run-time a combined execution trace (e.g., $X'_{Apps}(u_m)$) of the I active applications in the use-case u_m.

For that purpose, the design-time prepared execution traces of each active application (X_{app_i}) are combined according to a given algorithm (different mapping combination strategies can be used). In the following, we denote the process of combining execution traces by $/processing$. Figure 6.4 gives an example of one possible combined execution trace of applications on homogeneous cores in a use-case $u_1 = \{app_1, ap_2\}$, based on the previously prepared mappings. In this example the *LASP* (Longest Available Slot Packing) strategy presented in [14] has been used. According to this algorithm, $X'_{Apps}(u_1) = /processing\{X_{app_1}, X_{app_2}\}$, includes all the execution instants, from x_s of the first task to x_e of the last task in one hyper-period (*i.e.*, the least common multiple of periods as defined in Section 3.4) of the active applications in u_1.

In LASP, the instances of a task are always mapped into the same core in each period (task

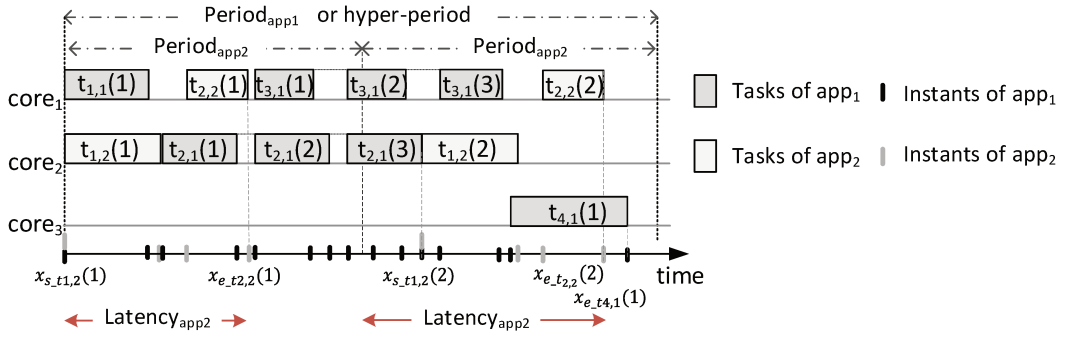

Figure 6.4: A run-time combined execution trace $X'_{Apps}(u_1)$ using the LASP strategy [14].

instances $t_{1,2}(1)$ and $t_{1,2}(2)$, allocated on $core_2$, are an example). Once the execution traces are combined, the start time of $t_{1,2}(2)$ (i.e. $x_{s\_t_{1,2}}(2)$) is adjusted and delayed in order to start after the previous task allocated into $core_2$ (*i.e.*, starting instant dependency on $x_{e\_t_{2,1}}(3)$). The adjusted instants then increase $Latency_{app_2}$ of the second period. In $X'_{Apps}(u_1)$ the instants are now absolute and computed according to the applied management strategy. Note that different instants can be obtained depending on the run-time management strategies that we want to evaluate.

6.3.3 Run-Time Mapping Control

In our approach, the run-time mapping simulation, handled by the *Run-Time Manager* (RTM), aims to control the execution of tasks according to the information provided by the run-time combined execution trace X'_{Apps}. The proposed simulation approach is depicted in Figure 6.5 for $u_1 = \{app_1, app_2\}$ and $u_3 = \{app_2\}$.

As shown in Figure 6.5, when u_1 is detected, the $/processing$ step is performed to determine $X'_{Apps}(u_1)$. The execution of the $/processing$ action corresponds to a certain evaluated run-time mapping strategy, and it is done in zero simulation time with no call to the simulation kernel. Corresponding to the obtained $X'_{Apps}(u_1)$ shown in Figure 6.4, the computed instants are further sorted in ascending order. The RTM then controls the states of each task accordingly.

In Figure 6.5, at the simulated instant $x_{s\_t_{1,1}}(1)$, $t_{1,1}(1)$ and $t_{1,2}(1)$ are started. The RTM inserts simulation delays through the action $/wait(wt)$ to wait for the next instant. Simulation time $SimTime$ moves forward $SimTime = SimTime + wt$. As the next instant is $x_{e\_t_{1,1}}(1)$, the waiting duration wt is expressed as $wt = x_{e\_t_{1,1}}(1) - x_{s\_t_{1,1}}(1)$. After this time, $t_{1,1}(1)$ is stopped. As for task instance , refering to Figure 6.4, $x_{s\_t_{2,1}}(1)$, $x_{s\_t_{2,1}}(2)$ and $x_{s\_t_{2,1}}(3)$, the RTM detects when several instances of the same task execute successively on the same processing core. In this case, only the first instance ($x_{s\_t_{2,1}}(1)$) is started and the last instance $x_{s\_t_{2,1}}(3)$ is stopped. This further reduces the activity of the simulation effort (*i.e.*, simulated kernel calls). This process is repeated for all the task instances in every hyper-period. When a

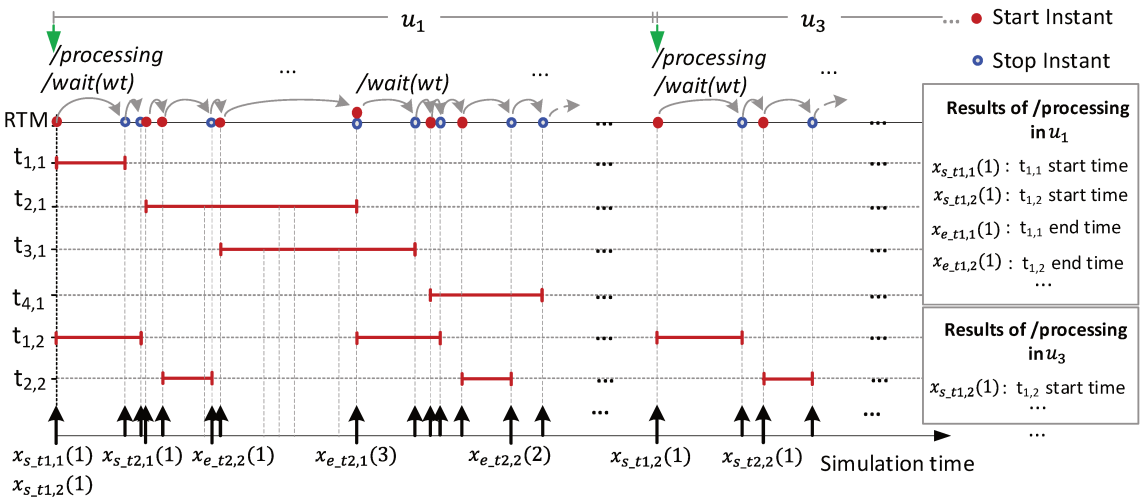

Figure 6.5: System-level approach for the simulation of run-time mapping strategies through the dynamic control of the execution of tasks for different use-cases.

new use-case is detected, the RTM performs the dynamic control of the tasks in the new use-case (*e.g.* $X'_{Apps}(u_3)$ where only app_2 is active). In this work, use-case detection is performed by periodical check, while the checkpoint interval can be set by the user.

6.3.4 Platform Heterogeneity Consideration

It is worth noting that this approach allows as well the evaluation and simulation of run-time mapping strategies for heterogeneous platforms (different processing cores, different v/f levels, $\cdots$). For this purpose, the varying values of computation time ($CompTime_{t_{h,i}}$) and communication time ($CommTime_{e_{g,i}}$) based on heterogeneous resource configurations should be taken into account in the run-time execution trace processing (step **(2)**). For a detected use-case, the design-time execution trace of each active application is first computed to obtain the absolute values of instants, according to the prepared task dependencies, the used core types and the configured v/f. The models (Eq.(3.8), (3.9), (3.11), (3.12)) presented in Chapter 3 can be used to update the values of the performance (*e.g.*, $CompTime_{t_{h,i}}$) and power values of tasks according to different core types and v/f configurations. Then, after applying a certain run-time mapping strategy, the instants of the combined execution trace are also dependent on heterogeneous resource configurations. Finally, the run-time mapping control (step **(3)**) is performed at different instants with adapted waiting time wt. Additionally, the power parameter of tasks can be updated at each instant for power/energy analysis.

6.4 Evaluation of the modeling and simulation approach

6.4.1 Simulation Environment

We validated the proposed simulation approach by using the industrial modeling and simulation framework Intel CoFluent Studio [39] without any framework modification. The built environment is given in Figure 6.6. The added modules of our proposed approach are highlighted by dotted box.

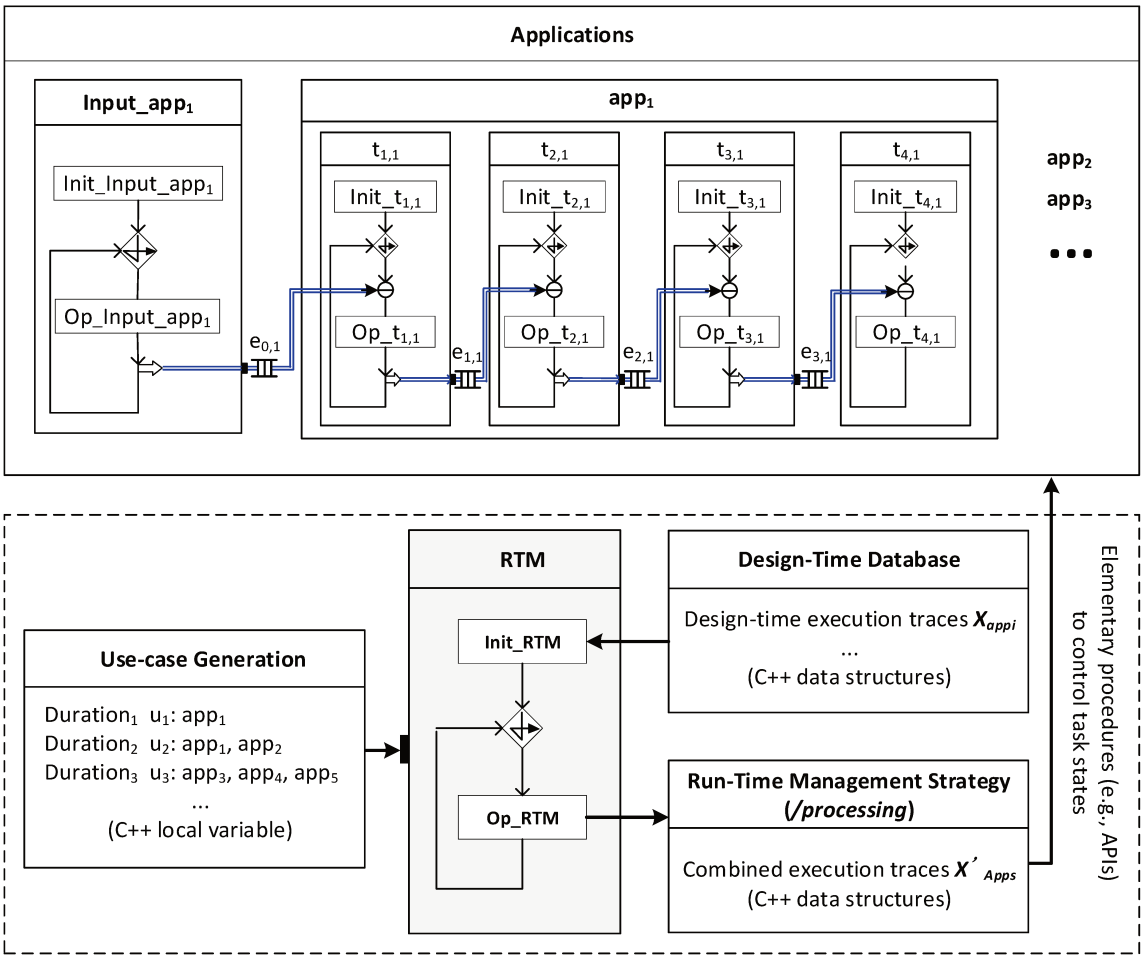

Figure 6.6: The simulation environment built in Intel CoFluent Studio framework. The added modules of our approach are highlighted by dotted box.

In the CoFluent framework, each application is modeled graphically with several functions (*i.e.*, tasks) and communications (*i.e.*, edges). For each function and communication, the computation/communication time and power consumption can be set by considering the influence of the platform. The built system model is then generated as a SystemC description

for further execution analysis.

In our implementation, the run-time manager model is captured graphically and can be considered as a specific function of the system. The RTM is activated by some simulated use-cases that are expressed by local variables. The implementation of a certain run-time management strategy ($/processing$) corresponds to the call to a C++ code developed to manipulate the previously defined data structures X_{app_i} and X'_{Apps}.

During the simulation, the run-time manager controls the states of each function and the advancement of simulation time according to the combined execution trace. Elementary procedures available in the used framework (`start`, `stop`, `resume`, `wait`) are used by the run-time manager to control the state of the functions. In the following, we evaluate the influence of this run-time manager model on the effort required for the system simulation.

6.4.2 Simulation Setup

In the case study, we aim to illustrate how the proposed modeling and simulation approach is applied to a heterogeneous architecture. The organization of the evaluated system is presented in Figure 6.7.

Applications	Nb of tasks	Nb of edges	Nb of token for each task	Period (µs)	Latency[17] of X_{appi} (µs)
app$_1$: H263-decoder.S	4	3	{1,11,11,1}	60	44
app$_2$: JPEG-decoder.S	6	5	{1,6,6,6,6,1}	180	118
app$_3$: H263-decoder.L	4	3	{1,66,66,1}	180	77
app$_4$: JPEG-decoder.L	6	5	{1,12,12,12,12,1}	270	164
app$_5$: H263-encoder	5	4	{1,5,5,1,1}	540	451
app$_6$: 5x**A$_1$**	20	19	5x{1,11,11,1}	270	220
app$_7$: 10x**A$_1$**	40	39	10x{1,11,11,1}	540	441

Use-case	Active app$_i$	Use-case	Active app$_i$
u_1	i=1	u_8	i=6,7
u_2	i=3	u_9	i=5,6,7
u_3	i=1,2	u_{10}	i=4,5,6,7
u_4	i=1,2,3	u_{11}	i=3,4,5,6,7
u_5	i=1,2,3,4	u_{12}	i=2,3,4,5, 6,7
u_6	i=1,2,3,4,5	u_{13}	i=1,2,3,4, 5,6,7
u_7	i=7		

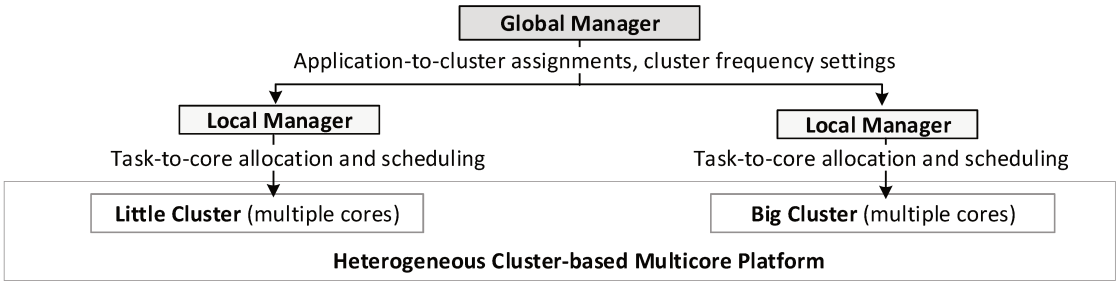

Figure 6.7: Evaluated hierarchical run-time management of multiple applications executed on a heterogeneous cluster-based platform.

We considered multimedia applications, such as H263 decoder, JPEG decoder and H263 encoder. Each application has been captured as a SDF model, based on the descriptions

[17]The latency of X_{app_i} is assumed to be obtained in the little cluster at $1.4GHz$.

provided in SDF3 [67]. As shown in Figure 6.7, app_1 and app_3 (respectively app_2 and app_4) are set to consume different token sizes for processing at different data exchanging speed. The first five applications are representatives and require different computation time and power. To evaluate the scalability of the proposed simulation approach, app_6 and app_7 are arbitrarily created to significantly increase the number of tasks. They are created by duplicating app_1 5 and 10 times respectively, while the iterations execute successively in one period. Each application is constrained by a predefined period. For further evaluation, in the following, 13 possible use-cases are defined with different active applications (seen on top right part of Figure 6.7). The duration of each use-case is not depicted in the figure for sake of clarity, but it can easily be set by the user.

We choose the Samsung Exynos 5422 [21] platform as the hardware target. As summarized in [25], the computation time of a task presents a ratio (*i.e.*, R_j^{perf} defined in Eq.(3.11), where j denotes the core type) of $1 : 0.5$ when executed on the little (Cortex-A7) or the big (Cortex-A15) cluster. Besides, the ratio of power consumption (*i.e.*, R_j^{power} defined in Eq.(3.12)) of a task executed on the little cluster and the big cluster is set to $1 : 4$. This platform allows frequency scaling of each cluster, while the operating voltage is adapted to the frequency setting. We used the model of Exynos 5422 in [8] to model how computation time and dynamic power consumption of tasks change with frequency. In the case-studies, we assume that the processing/communication resources in each cluster are sufficient for the active applications of each use-case. It means that resource constraints are not taken into account in the case-studies for the reason of simplification. Note that whether considering resource constraints only affects task execution instants (due to the potential resource competition and communication congestion), and does not change the *run-time mapping control* processing (the main contribution) of the proposed simulation approach.

The hierarchical managers are built to implement some run-time management strategies of the system. The two local managers are individually used for each cluster to optimize task-to-core allocation and scheduling. In order to coordinate the local managers, the global manager determines application-to-cluster allocations and sets cluster frequencies. The management strategies are based on design-time prepared execution traces. For each application, the established X_{app_i} maps one task onto one core.

6.4.3 Validation of the Simulation Approach on Latency Criteria

In this part, the proposed simulation approach is applied for the evaluation of a local management strategy, which considers task-to-core mapping inside a cluster (*i.e.*, on a homogeneous architecture). Here, we apply FCFS [12, 13] and LASP [14] in the local manager to get a combined mapping of active applications in each use-case. Then the latency of each application can be obtained for the two different strategies.

Figure 6.8 shows the latency evolution of app_1 in four different simulated use-cases. The simulations are performed in the little cluster at a fixed cluster frequency of $f = 1.4\ GHz$.

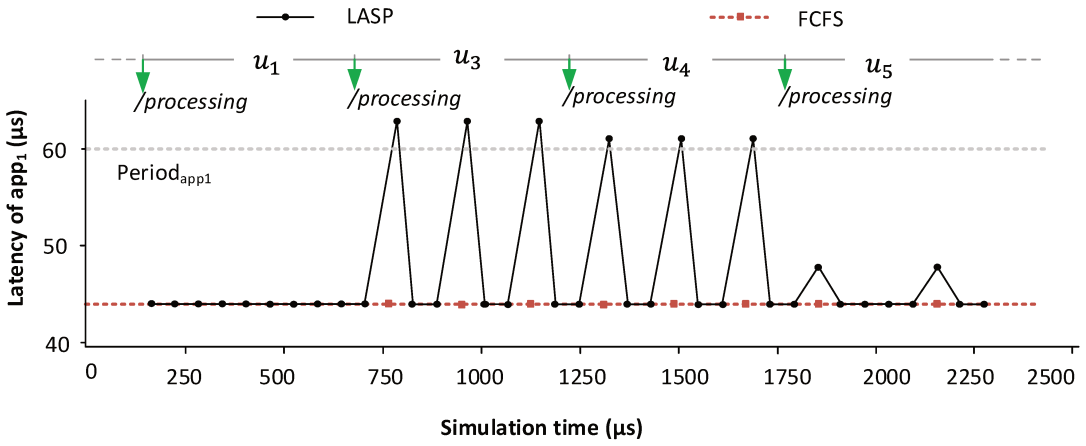

Figure 6.8: Evolution of simulated app_1 latency, captured for four different use-cases. Results are given for FCFS [12, 13] and LASP [14].

In this figure, the green arrows indicate the instants when a new combined execution trace is computed by $/processing$ (*i.e.*, LASP or FCFS) in each use-case. For a clear illustration, the latency of app_1 is captured nine times for each use-case. It can be observed that FCFS achieves the same $Latency_{app_1}$ (*i.e.*, equals to the latency of its design-time prepared mapping) in all use-cases. This is because FCFS performs task-to-core mapping without degrading application performance, as previously discussed in Chapter 4. On the other hand, LASP achieves the same $Latency_{app_1}$ as FCFS in u_1 where only app_1 is active. However, $Latency_{app_1}$ of LASP can be larger in u_3, u_4 and u_5. In particular, the maximum $Latency_{app_1}$ in u_3 and u_4 even violate the timing constraint $Period_{app_1}$. As previously discussed, the increase of latency comes from the possible delay of tasks re-allocation using the LASP combination strategy.

From this evaluation, we can see that our simulation approach is able to capture the latency behavior of an application in different use-cases, depending on the applied run-time mapping strategy.

6.4.4 Validation of the Simulation Approach on Power Criteria

We then applied the proposed simulation approach to the global manager, which determines the application-to-cluster assignment and set the cluster frequencies on a heterogeneous architecture. In the simulation model, different platform configurations result in different computation time and different dynamic power consumption of a task, as can be explained in Figure 6.9.

Figure 6.9 shows the dynamic power consumption of app_1 (in u_1) under the control of the global manager. The green arrows indicate the instants when an execution trace is adapted according to different platform configurations. In the first configuration (*i.e.*, $f = 1.4\ GHz$ in the little cluster), index (1) corresponds to the active state of $t_{1,1}$. Index (2) indicates the

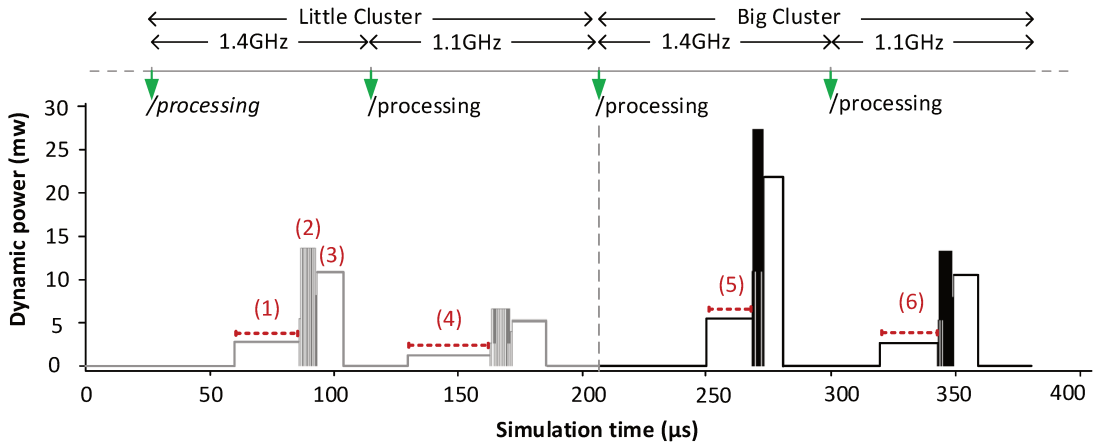

Figure 6.9: Simulated dynamic power of app_1 captured with the advancement of simulation time. Results are given for u_1 according to different platform configurations.

activities of $t_{1,2}$ and $t_{1,3}$ that are active in parallel. Index (3) shows the activity of task $t_{1,4}$. For task $t_{1,1}$, the power consumption with different platform configurations (*i.e.*, different frequency configurations in the little and big clusters) are represented in indexes (1), (4), (5) and (6), while the task computation time is reflected by the length of the red dotted lines. From (1) and (4), when the operating frequency decreases from $f = 1.4\,GHz$ to $f = 1.1\,GHz$ in the little cluster, the dynamic power consumption of $t_{1,1}$ decreases and the task computation time increases. In the case of (1) and (5), $t_{1,1}$ is executed at $f = 1.4\,GHz$ on the little cluster and big cluster respectively. The power consumption of the task observed on the big cluster is higher, while the computation time is smaller. Note that the dynamic power consumption and task computation time, which depend on the used core types and v/f configurations, are estimated through the models (*i.e.*, Eq.(3.8), (3.9), (3.11) and (3.12)) presented in Chapter 3.

This evaluation shows that our proposed approach is able to capture the behavior of an application under different platform configurations (*i.e.*, different application-to-cluster assignment, various v/f settings), depending on the used run-time management strategy.

6.4.5 Evaluation of the Simulation Approach

Comparison of run-time mapping strategies:

The proposed simulation approach allows for evaluating different run-time management strategies. We compared two Local Management Strategies, namely FCFS [12, 13], where only the tasks from one application are mapped on the same core and LASP [14], which allows the task of different applications to be mapped on the same core. The simulations are performed for the little cluster with $f = 1.4\,GHz$. In Table 6.1, we summarize the estimated application latency for different use-cases. As previously observed, LASP leads to application latency

increase in some use-cases.

Table 6.1: Evaluation of run-time management strategies based on latency and power

Compared Critera	Strategy	u_1	u_2	u_3	u_4	u_5	u_6
Latency[18]	FCFS	1	1	1	1	1	1
	LASP	1	1	1.43	1.39	1.18	1.64
System Power[19]	Exhaustive	1	1	1	1	1	1
	HPF	1	1	1	1.12	1.15	1.07
	LPF	1.68	2	1.68	1.89	1.94	2.14

[18] Depicts the latency of the application that has the highest variation in a use-case. Each value is normalized by the latency obtained by FCFS.
[19] Represents the average dynamic power of the system at the scaled frequency. Each value is normalized by the system power obtained by Exhaustive.

Three Global Management Strategies are also compared. They differ in how they allocate applications to the clusters. *Exhaustive* refers to the strategy that assigns applications to the two clusters by exhaustively searching the best power efficiency. The High-Performance-First (HPF) strategy and the Low-Power-First (LPF) strategy assign all the active applications to the big cluster or to the little cluster respectively. Once the application allocation is done, cluster frequency is decreased as much as possible under timing constraints. Then FCFS is used in each local manager to determine task-to-core mappings. From Table 6.1, we can observe the poor power efficiency of using only one cluster (*i.e.*, HPF and LPF).

Evaluation of simulation efficiency:

We also analyzed the scalability of the proposed simulation method by comparing it with the CoFluent default simulation method. The proposed approach simulates the execution of applications under the control of the Run-Time Manager (RTM), and different mappings can be provided for each application in different use-cases. On the other hand, without the RTM model, the default simulation approach only provides one static mapping of the applications in every use-case. In this work, we characterize *simulation effort* by the average time needed to complete one simulation run. Figure 6.10 shows the differences of the simulation effort [20] between the two approaches. The results include the execution traces processing and mapping control overheads.

We define an increasing number of successively simulated use-cases (see *the first use-case* $\rightarrow$ *the last use-case* in Figure 6.10) within a fixed duration of simulation time, allowing each application to execute 100 to 240 periods. When the number of simulated use-cases increases from 1 to 7, the number of considered tasks increases from 40 to 85, while the difference of the

[20] Differences of simulation effort $= \frac{\text{Simulation effort of the proposed approach - Simulation effort of the CoFluent default approach}}{\text{Simulation effort of the CoFluent default approach}}$

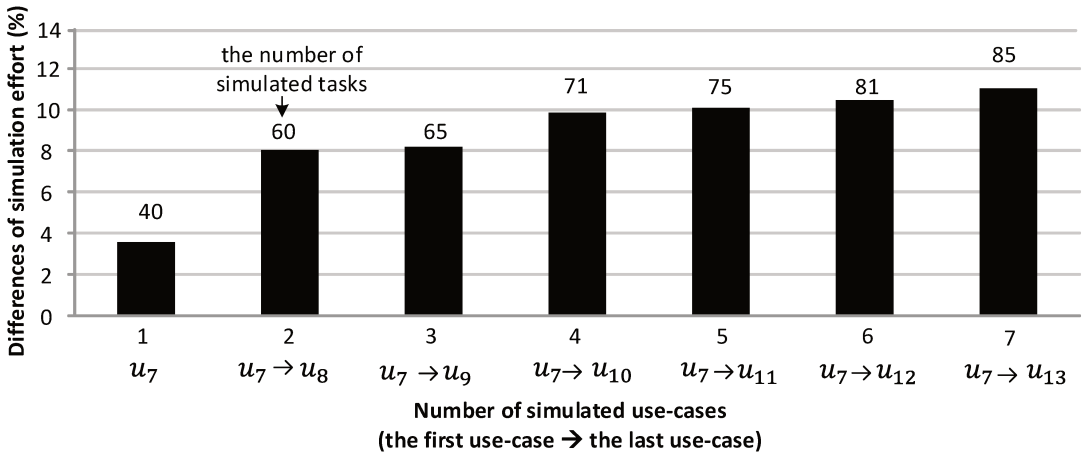

Figure 6.10: The differences of simulation effort between the proposed approach and the default approach. Results are given for an increasing number of simulated use-cases and running tasks.

simulation effort increases only from 3.8% to 10.8%. Since the proposed approach dynamically starts and stops the execution of tasks during simulation, it is reasonable to use more time to finish a simulation than the default approach (which considers only one mapping). This overhead is also due to the fact that our approach takes into account the run-time manager in simulation while the default approach considers a static mapping (requiring eventually more corner-cases study).

6.5 Summary and Discussion

In this chapter, we presented an approach to allow the system-level simulation of run-time management strategies in multi/many-core systems. This approach could be used to consider different numbers of applications executed on heterogeneous platforms at varied v/f configurations. In contrast to the trace-based simulation approach used by the *Sesame* framework, the proposed approach does not dynamically dispatch application events to platform resources. Instead, our approach dynamically controls the execution states of application tasks and advances the simulation time based on the computed mapping (execution traces) for active applications.

The proposed approach is implemented in the Intel CoFluent Studio framework. In the experimental case-study, seven applications executed dynamically on a heterogeneous cluster-based multicore platform are considered and different run-time management strategies are evaluated in terms of latency and power. The scalability of the simulation approach is also analyzed and it has been observed that the influence of the proposed approach on the simulation effort is reasonable (less than 10.8% compared to the default Cofluent framework for 85 running tasks).

Our simulation approach can be conveniently implemented without any modification of the used framework, making the approach portable to other simulation environments. The main difficulty is to correctly compute the instants of each application task executed on the platform resources under the control of a run-time management strategy.

Like the existing system-level modeling and simulation approaches with run-time management strategy extensions, the proposed approach explores and simulates run-time mapping based on a design-time database. This approach can support both hybrid and fully on-the-fly decision-making strategies. In hybrid strategy simulation, the database provides some optimized design-time mappings to establish a run-time mapping. In on-the-fly strategy simulation, the database provides the computation/communication time of task/edge on different platform resources and then task execution instants are computed at run-time according to the task execution order, the platform current state, and the applied strategy.

Chapter 7

Conclusion

Contents

In this chapter, we first summarize the work of this dissertation. Then, we discuss the possible improvements and future works.

7.1 Dissertation Summary

This dissertation work has focused on run-time management of multiple applications (*i.e.*, each application has dependent tasks) executed dynamically in cluster-based multi/many-core systems. Dynamic task mapping and DVFS aim to be simultaneously applied to achieve energy efficiency while satisfying system constraints (*i.e.*, application timing, platform resource, and platform frequency constraints). In this work, energy efficiency is characterized by the average dynamic power of active applications. To reduce run-time computation burdens and guarantee mapping feasibility, our work applies hybrid mapping strategies that fulfill dynamic mapping based on one or several design-time prepared mappings for each application.

Towards the run-time management purpose, two main research problems are studied. The first research problem is *how to coordinate dynamic task mapping and DVFS*. Most existing strategies separate dynamic task mapping and DVFS into two independent steps without considering their mutual influence. In our work, to estimate the impact of a prepared application mapping on the cluster frequency configurations, we have introduced a new parameter, the Minimum Allowed Frequency (MAF). MAF defines the minimum required frequency for a given prepared mapping to meet the application timing constraint. We have applied MAF to coordinate hybrid mapping and per-cluster DVFS according to two dimensions: local optimization within a cluster and global optimization of the overall system.

To achieve local optimization within a cluster, Chapter 4 has presented a novel management strategy to determine task-to-core mapping and cluster frequency configuration to achieve near-optimal energy efficiency of the cluster. At design-time, multiple mappings that have different trade-offs between application performance and core usage are prepared. At run-time, one prepared mapping is selected for each active application. However, many exploration attempts might be required to select application mappings that allow a low cluster frequency configuration without violating system constraints. To offer a good trade-off between management efficiency and complexity, a run-time selection strategy is proposed to select a prepared mapping for each active application and a low cluster frequency under the guidance of MAFs. Besides, a mapping combination strategy, GAPVC, is proposed to heuristically combine the selected application mappings with less resource usage without degrading the application performance. To achieve a near-optimal solution, the selection strategy and the mapping combination strategy are applied iteratively. Thanks to the guidance of MAFs, our run-time management strategy can exclude some unnecessary iterations to explore system configuration. Moreover, our experiments have demonstrated that the involvement of our selection strategy and mapping combination strategy can reduce average power consumption by up to 206% when compared to the literature. The proposed management strategy can be applied to each cluster to enable distributed management in the overall system.

For the global optimization of the overall system, we have proposed a hierarchical management strategy in which the global management determines application-to-cluster assignments and sets cluster frequencies, while the local management optimizes task-to-core mappings in each cluster. However, energy optimization in a global system has not been explicitly studied for task-dependent applications in platforms with more clusters (*e.g.*, more than 2) in state-of-the-art strategies. In Chapter 5, we presented our management solution for homogeneous/heterogeneous systems with different numbers of clusters or with different numbers of cores inside each cluster. In our strategy, we assume that one mapping is prepared for each application at design-time and one application is assigned to one cluster at any given time. Under these assumptions, we formulate the global management problem into a 0-1 IP model. The 0-1 IP model takes use of MAF to estimate the optimized cluster frequencies based on different application-to-cluster assignments. To achieve the solution to the 0-1 IP optimization problem, two different global management strategies have been proposed. Firstly, a neighboring search strategy NSACA is proposed. It considers the assignments of all active applications (*i.e.*, in a use-case) holistically, allowing all migrations of applications. The NSACA strategy reveals the application assignment principle for near-optimal results of the 0-1 IP formulation. That is to assign applications with close MAF to the same cluster. Our experimental evaluation shows that the average power consumption achieved by NSACA is only 1.93% worse than the optimal solution (*i.e.*, by Exhaustive search), but the speed of NSACA is 2674 times faster. Moreover, a greedy search strategy GSACA that considers the assignments of active application individually is presented. Motivated by reducing the number

of migration, the GSACA strategy only assigns newly active applications in each use-case and allows limited migrations for further energy optimization. The number of allowed migrations can be controlled according to user requirements. Our experimental evaluation indicates that 0.22 more migration (*i.e.*, the number of migrated applications) per use-case in GSACA can lead to 2.5% reduction of the average power consumption of the overall system. This evaluation was performed under the assumption that migration overhead is 0. Lastly, mapping combination strategies FCFS and GAPVC have been compared in the local management of each cluster. The comparison reveals that less core usage within a cluster achieved by GAPVC enables further energy optimization in the overall system. This is because more applications can be assigned to more efficient clusters (*i.e.*, due to heterogeneous cluster or low cluster frequency). Based on the same global management strategies, our experiments have shown that GAPVC can reduce the average power consumption of system by up to 57.65% compared to FCFS (First-Come-First-Served application mapping combination strategy) in a use-case.

The second research problem addressed in this work is: *how to evaluate run-time management strategies*. Regarding this problem, most of the existing system-level simulation-based frameworks consider static application mapping and do not support run-time management effects. To enable run-time management property, we have proposed a new system-level modeling and simulation approach that allows flexible evaluation of different run-time management strategies, different numbers of active applications, and different platform configurations (*e.g.*, heterogeneous processing elements, v/f configurations). In our simulation approach, the task execution instants of dynamic application mappings are first computed based on design-time prepared data. Then, according to the computed instants, the execution states of application tasks are controlled dynamically as simulation time advances. The proposed simulation approach has been validated in the Intel CoFluent framework. We have evaluated the scalability of the simulation approach. It has been observed that the influence of the proposed approach on the simulation effort is reasonable. Compared to the default Cofluent framework (for 85 running tasks), the simulation workload increased by less than 10.8%. But the default Cofluent framework simulates only one mapping at run-time.

In summary, this dissertation has presented our contributions on run-time management of multiple task-dependent periodic applications on cluster-based multi/many-core systems for energy optimization. Our proposed run-time management strategies can be partially applied to *sporadic applications or tasks*. Sporadic tasks refer to non-periodic tasks with minimum arrival time to ensure their schedulability [27]. On one hand, our *GAPVC mapping combination strategy* (in the local management) assumes that application arrival time is predictable and known, but this is not the case for sporadic applications. Therefore, GAPVC is not feasible for sporadic applications. We can still use FCFS mapping combination strategy to set the scheduling of sporadic tasks. On the other hand, our *MAF-based selection strategy* (in the local management) and *application-to-cluster assignment strategy* (in the global management) can still be feasible for sporadic applications. The two strategies depend on *design-time prepared*

MAFs. We can estimate the MAF for each prepared mapping of sporadic applications through scheduling analysis (*i.e.*, demand bound function [82]) or measurement.

Our current work has some limitations. In terms of application models, our work considers computation-intensive applications, assuming communication congestion and communication energy can be neglected. But this is not the case for communication-intensive or memory-intensive workloads. Further evaluations and improvements should be performed to adjust our proposed strategies to communication-intensive workloads. In terms of power and performance models, we built the models for each independent application. The models are fixed and are established according to design-time information. However, due to the influence of communication contention or memory demands, the models may be inaccurate when multiple applications are executing simultaneously at run-time. Moreover, our work assumes that all applications have the same power/performance ratios from one cluster to another. This is an ideal assumption in our global management strategies. Nevertheless, computation-intensive and communication-intensive can have different power/performance ratios in different platform configurations (*i.e.*, resource heterogeneity and frequency configurations). More experimental measurements should be performed to validate the power and performance models.

7.2 Future Works

Future work can address the limitations of our work, that is, we can adjust our proposed strategies to various applications (*e.g.*, computation-intensive, communication-intensive workloads) and conduct more measurements to verify our power/performance models. Additionally, we can also improve our work in other possible aspects.

Further improvement of management strategies

One of our future work will consider further optimization of run-time management strategies. Further improvements can be considered for both local optimization and global optimization.

- **Optimizing mapping combination strategy in local management**

For the local optimization within a cluster, we have proposed a heuristic strategy (*i.e.*, GAPVC) to determine task-to-core mappings. However, our local management strategy has assumed that the communication cost between core is much smaller than computation energy, and the communication energy can be neglected. The possible communication congestion has been also neglected when performing application mapping combination at run-time. Communication congestion occurs due to the competition for platform resources from simultaneous communication activities. Figure 7.1 (a) gives one example of combined mapping (of app_1 and app_2) that neglects the possible communication congestion (highlighted in shadow). This Communication congestion can delay some task execution and degrade

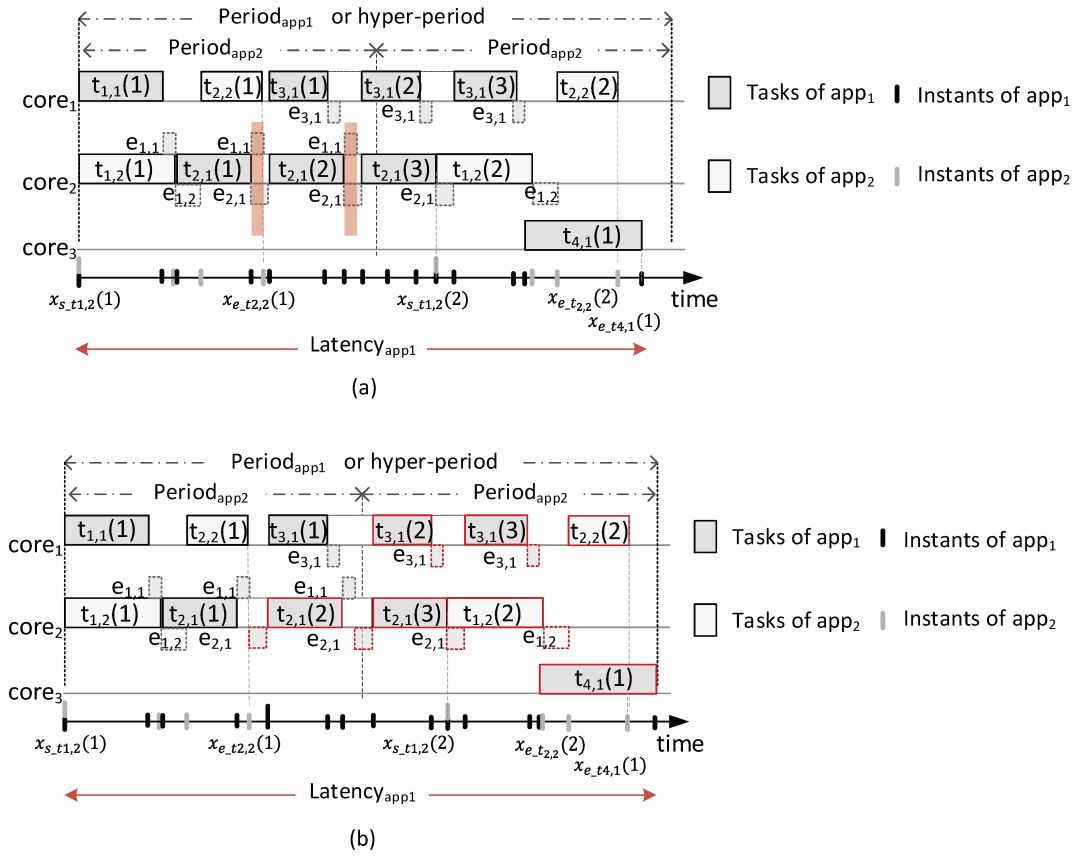

Figure 7.1: The combined mappings that (a) neglect and (b) consider communication congestion

application performance. In our future work, we can take the influence of communication congestion into account. For this purpose, a *correction* operation should be done to delay some task execution of the combined mapping. Figure 7.1 (b) shows the combined mapping after *correction* operation. The tasks and edges with correction are highlighted in red in the figure. The communication congestion can be avoided but application latency can increase (see $Latency_{app_1}$). In the future, we will consider the *correction* operation based on specific communication resources (*e.g.*, bus or NoC).

- **Optimizing greedy strategy (GSACA) in global management**

For the global optimization of the overall system, we have presented a greedy search strategy to determine application-to-cluster assignment and cluster frequency configurations for active applications. In this strategy, the number of allowed migrations can be set according to user requirements. The migration decisions can be further addressed in future work. For example, we can only migrate applications with acceptable migration costs (*i.e.*, time and power/energy spent migrating applications from one cluster to another). Suppose that the migration cost

of each application (app_i) is known through design-time measurement, the migration energy is indexed by $E^m_{app_i}$. In contrast, migration benefits refer to energy savings due to migration. Let's assume that app_i (*i.e.*, in a use-case) will be active for T_{app_i} units of time (*i.e.*, known by design-time information or run-time prediction). The system average power before and after the migration of app_i are indexed by P^{sys}_{avg} and $P^{sys\prime}_{avg}$, respectively (*i.e.*, known through the estimation of the 0-1 IP formulation). Thus the migration benefit can be expressed as $(P^{sys\prime}_{avg} - P^{sys}_{avg}) \times T_{app_i}$. When the migration benefit outweighing its cost ($(P^{sys\prime}_{avg} - P^{sys}_{avg}) \times T_{app_i} > E^m_{app_i}$), the migration of app_i is acceptable. The difficulties would be on how to measure the migration cost of each application ($E^m_{app_i}$) and how to correctly predict the duration of the current use-case (*i.e.*, T_{app_i}).

- **Optimizing global management strategies in design-time preparation**

In this work, our global management strategies have considered that one mapping is prepared for each application at design-time. This is the first step in developing our hierarchical management strategies. In the future, we will consider the situation where multiple design-time mappings can be prepared for each application. The prepared mappings of each application

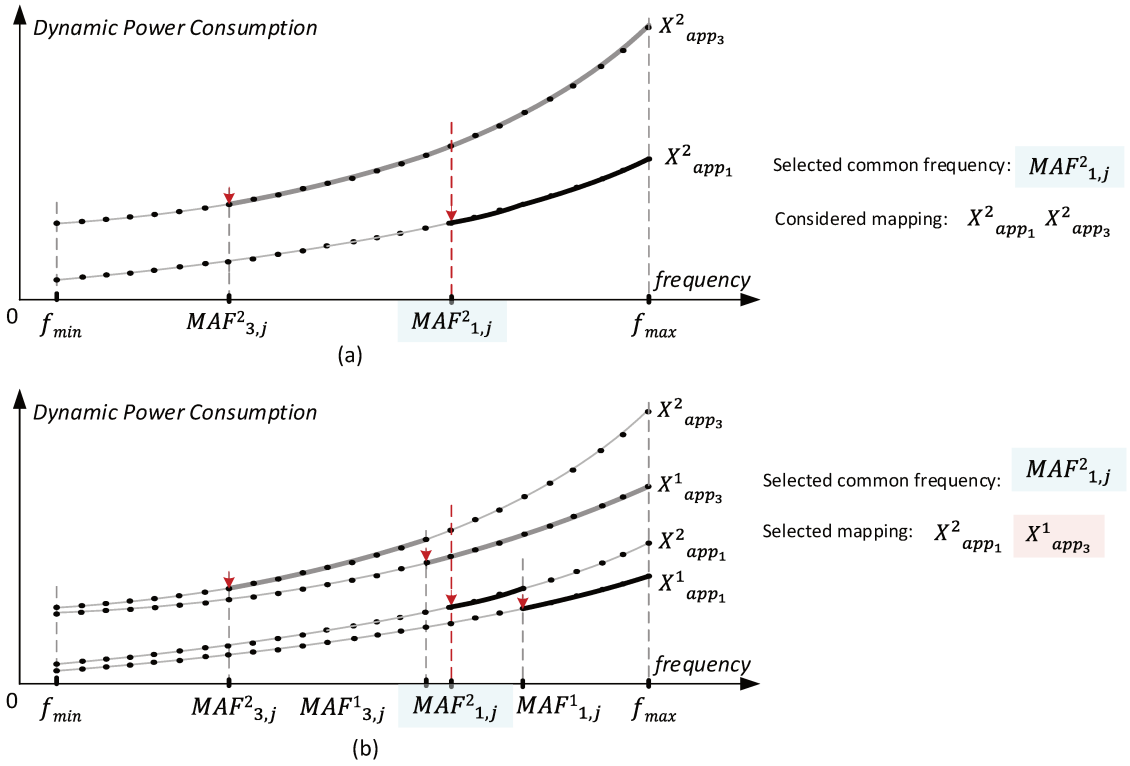

Figure 7.2: MAF-based mapping selection for active applications in situations of (a) one prepared mapping for each application and (b) multiple prepared mapping mappings for each application

have different trade-offs between application performance and used cores, providing more opportunities for resource optimization. As previously discussed in Section 5.4.3, fewer used cores within a cluster allows more applications executed in more energy-efficient clusters (*i.e.*, due to heterogeneous cluster, low cluster frequency), and consequently improves the energy efficiency of the overall system.

In this case, we can firstly perform global management (application-to-cluster assignment and cluster frequency configurations) based on *one* prepared mapping (*e.g.*, the mapping with the minimum MAF) of each application. Then, *multiple* prepared mappings can be considered in the local management (task-to-core allocation and scheduling) of each cluster to further optimize core usage. To this end, the NSACA, GSACA strategies based on one prepared mappings of each application can be directly applied in global management. The local management strategy based on multiple prepared mappings of each application should be slightly modified. In our modification, the local management is required to perform mapping selection, according to the cluster frequency determined by global management. Figure 7.2 gives an example of the modification of the mapping selection in local management. The definition of the symbols used in the figure can be found in Chapter 5 (see Section 5.3).

In the example of Figure 7.2, app_1 and app_3 are assigned to the same cluster and the cluster frequency is determined by the application with the maximum MAF ($MAF^2_{1,j}$ in part (a) of the figure). In the case of figure (a), the only one prepared mapping (*i.e.*, $X^2_{app_3}$ is the prepared mapping for app_3 using 2 cores) for each application is selected directly. In the case of figure (b), multiple mappings can be prepared for an application. As previously discussed in Section 4.4.2, at different frequencies, the design-time mapping that could be selected for each application can be different. If a low frequency level can be supported by multiple application mapping, the mapping using fewer cores should be selected due to its less power consumption (*i.e.*, less communication between cores). The part (b) of Figure 7.2 highlights the designated design-time mapping (bold line) selection for each application at different frequencies. In the example of figure (b), the cluster frequency is still $MAF^2_{1,j}$. The selected mapping for app_3 changes to $X^1_{app_3}$, reducing 1 core of the prepared mappings. For more active applications with more prepared mappings, the total number of reduced cores can be more significant. Note that after the mapping selection, the selected mappings can be combined through different mapping combination strategies, such as FCFS and GAPVC.

Real implementation of management strategies

Our future work will consider real implementation of our proposed management strategies in real cluster-based multi/many-core systems, such as Odroid XU3 [21], Kalray MPPA [20]. The implementation is feasible due to the support of some open-source tools. For example, PREESM [83] can simulate signal processing applications and generate application code for heterogeneous multi/many-core embedded systems. The Synchronous Parameterized and Interfaced Dataflow Embedded Runtime (SPIDER) [84, 85] allows the implementation of

run-time mapping strategies to adapt the allocation and scheduling of application tasks onto platform resources. The real implementation helps to make a practical evaluation of our management strategies. We can evaluate whether the strategy complexity, migration overheads, and communication congestion that can jeopardize application timing constraints.

Apply Dynamic Power Management (DPM) to optimize static energy

In the work of this dissertation, the proposed management strategies have focused on the optimization of average dynamic power consumption (*e.g.*, or dynamic energy consumption) through dynamic task mapping and DVFS. Since static power/energy is unneglectable in future multi/many-core systems [86], future work can consider Dynamic Power Management (DPM) to optimize system static power. DPM dynamically shuts-down system components (*e.g.*, cores) that are idle or underutilized, and wake up them when necessary. We can apply DPM with dynamic task mapping and DVFS to optimize both dynamic energy and static energy of systems. In this case, we can shut-down the idle cluster/cores after determining the application task map and cluster frequency configuration through our proposed hierarchical management (see Chapter 5). However, shutting-down and waking up clusters/cores will incur some penalties in terms of system response time and extra energy. The difficulty would be how to avoid DPM penalties outweighing its benefits.

Consider several other optimization objectives

In the future, we may consider several other optimization objectives that are not covered in this dissertation. For instance, energy efficiency is closely related to system thermal problems due to hot-spots in modern chips. Moreover, the overheating problem can worsen life-time reliability of systems. Finally, we can also address security issues of run-time managers. Since some possible attacks to a manager can mislead the manager to make unappropriated configurations of systems, and thus causes application performance violations or other serious problems (*e.g.*, in terms of energy, thermal and reliability).

Bibliography

[1] 42 years of microprocessor trend data. Available:https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data.

[2] Amit Kumar Singh, Piotr Dziurzanski, Hashan Roshantha Mendis, and Leandro Soares Indrusiak. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. *ACM Computing Surveys (CSUR)*, 50(2):24, 2017.

[3] Peng Yang, Paul Marchal, Chun Wong, Stefaan Himpe, Francky Catthoor, Patrick David, Johan Vounckx, and Rudy Lauwereins. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *Proceedings of the 15th international symposium on System Synthesis*, pages 112–119. ACM, 2002.

[4] Santiago Pagani, Anuj Pathania, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. Energy efficiency for clustered heterogeneous multicores. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1315–1330, 2016.

[5] Young Geun Kim, Minyong Kim, and Sung Woo Chung. Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors. *IEEE Transactions on Computers*, 66(11):1878–1889, 2017.

[6] Anil Kanduri, Antonio Miele, Amir M Rahmani, Pasi Liljeberg, Cristiana Bolchini, and Nikil Dutt. Approximation-aware coordinated power/performance management for heterogeneous multi-cores. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[7] Heba Khdr, Santiago Pagani, Ericles Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. Power density-aware resource management for heterogeneous tiled multicores. *IEEE Transactions on Computers*, 66(3):488–501, 2016.

[8] Houssam-Eddine Zahaf, Abou El Hassen Benyamina, Richard Olejnik, and Giuseppe Lipari. Energy-efficient scheduling for moldable real-time tasks on heterogeneous computing platforms. *Journal of Systems Architecture*, 74:46–60, 2017.

[9] Bart Kienhuis, Ed F Deprettere, Pieter Van der Wolf, and Kees Vissers. A methodology to design programmable embedded systems. In *International Workshop on Embedded Computer Systems*, pages 18–37. Springer, 2001.

[10] Wei Quan and Andy D Pimentel. A hybrid task mapping algorithm for heterogeneous mpsocs. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(1):14, 2015.

[11] J. Lemaitre and R. Le Moigne. Dynamic migration and performance optimization of deterministic applications across platform components using intel cofluent studio. In *DAC Workshop on System-to-Silicon Performance Modeling and Analysis*, June 2015.

[12] Heba Khdr, Santiago Pagani, Ericles Sousa, Vahid Lari, Anuj Pathania, Frank Hannig, Muhammad Shafique, Jürgen Teich, and Jörg Henkel. Power density-aware resource management for heterogeneous tiled multicores. *IEEE Transactions on Computers*, 66(3):488–501, 2017.

[13] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. A hybrid strategy for mapping multiple throughput-constrained applications on mpsocs. In *Proceedings of the 14th international conference on Compilers, architectures and synthesis for embedded systems*, pages 175–184. ACM, 2011.

[14] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Resource and throughput aware execution trace analysis for efficient run-time mapping on mpsocs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1):72–85, 2016.

[15] Gordon E Moore et al. Progress in digital integrated electronics. In *Electron Devices Meeting*, volume 21, pages 11–13, 1975.

[16] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[17] Thomas M Conte and Paolo A Gargini. On the foundation of the new computing industry beyond2020. *Preliminary IEEE RC-ITRS Report*, 2015.

[18] Andreas Olofsson. Epiphany-v: A 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832*, 2016.

[19] Jason Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, David Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *2010 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 108–109. IEEE, 2010.

[20] Mppa 2-256. Available:https://www.kalrayinc.com/technology/.

[21] Exynos 5 octa (5422). Available:http://www.samsung.com/exynos.

[22] Shubham Kamdar and Neha Kamdar. big. little architecture: Heterogeneous multicore processing. *International Journal of Computer Applications*, 119(1), 2015.

[23] Helio-x30. Available:https://www.mediatek.com/products/smartphones/mediatek-helio-x30.

[24] Lei Yang, Weichen Liu, Nan Guan, Mengquan Li, Peng Chen, and HM Edwin. Dark silicon-aware hardware-software collaborated design for heterogeneous many-core systems. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 494–499. IEEE, 2017.

[25] Anastasiia Butko, Florent Bruguier, David Novo, Abdoulaye Gamatié, and Gilles Sassatelli. Exploration of performance and energy trade-offs for heterogeneous multicore architectures. *arXiv preprint arXiv:1902.02343*, 2019.

[26] Young Choon Lee and Albert Y Zomaya. Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel and Distributed Systems*, 22(8):1374–1381, 2010.

[27] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, 1989.

[28] Nathaniel Pinckney, Korey Sewell, Ronald G Dreslinski, David Fick, Trevor Mudge, Dennis Sylvester, and David Blaauw. Assessing the performance limits of parallelized near-threshold computing. In *DAC Design Automation Conference 2012*, pages 1143–1148. IEEE, 2012.

[29] Abdullah Elewi, Mohamed Shalan, Medhat Awadalla, and Elsayed M Saad. Energy-efficient task allocation techniques for asymmetric multiprocessor embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(2s):71, 2014.

[30] Luca Benini, Davide Bertozzi, and Michela Milano. Resource management policy handling multiple use-cases in mpsoc platforms using constraint programming. In *International Conference on Logic Programming*, pages 470–484. Springer, 2008.

[31] Lionel Torres, Pascal Benoit, Gilles Sassatelli, Michel Robert, Fabien Clermidy, and Diego Puschini. An introduction to multi-core system on chip–trends and challenges. In *Multiprocessor System-on-Chip*, pages 1–21. Springer, 2011.

[32] Chung-Hsing Hsu, Ulrich Kremer, and Michael Hsiao. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. In *ISLPED'01: Proceedings of the 2001 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 01TH8581)*, pages 275–278. IEEE, 2001.

[33] Marco ET Gerards, Johann L Hurink, and Jan Kuper. On the interplay between global dvfs and scheduling tasks with precedence constraints. *IEEE Transactions on Computers*, 64(6):1742–1754, 2014.

[34] Basireddy Karunakar Reddy, Amit Kumar Singh, Dwaipayan Biswas, Geoff V Merrett, and Bashir M Al-Hashimi. Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores. *IEEE Transactions on Multi-Scale Computing Systems*, 4(3):369–382, 2017.

[35] Gang Chen, Kai Huang, and Alois Knoll. Energy optimization for real-time multiprocessor system-on-chip with optimal dvfs and dpm combination. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(3s):111, 2014.

[36] Jonathan A Winter, David H Albonesi, and Christine A Shoemaker. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *2010 19th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 29–39. IEEE, 2010.

[37] Shahid H. Bokhari. On the mapping problem. *IEEE Transactions on Computers*, (3):207–214, 1981.

[38] Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 9–pp. IEEE, 2003.

[39] Intel cofluent studio. Available:http://www.intel.com/.

[40] S Wayne Bollinger and Scott F Midkiff. Processor and link assignment in multicomputers using simulated annealing. In *ICPP (1)*, pages 1–7, 1988.

[41] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[42] Ch Ykman-Couvreur, Vincent Nollet, Fr Catthoor, and Henk Corporaal. Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management. In *2006 International Symposium on System-on-Chip*, pages 1–4. IEEE, 2006.

[43] Ewerson Luiz de Souza Carvalho, Ney Laert Vilar Calazans, and Fernando Gehm Moraes. Dynamic task mapping for mpsocs. *IEEE Design & Test of Computers*, 27(5):26–35, 2010.

[44] Vincent Nollet, Prabhat Avasare, Hendrik Eeckhaut, Diederik Verkest, and Henk Corporaal. Run-time management of a mpsoc containing fpga fabric tiles. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(1):24–33, 2007.

[45] Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar, and Wu Jigang. Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms. *Journal of Systems Architecture*, 56(7):242–255, 2010.

[46] Andreas Schranzhofer, Jian-Jian Chen, and Lothar Thiele. Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms. *IEEE Transactions on Industrial Informatics*, 6(4):692–707, 2010.

[47] Yu-Kwong Kwok, Anthony A Maciejewski, Howard Jay Siegel, Ishfaq Ahmad, and Arif Ghafoor. A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 66(1):77–98, 2006.

[48] Amit Kumar Singh, Akash Kumar, and Thambipillai Srikanthan. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 18(1):9, 2013.

[49] Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proceedings of the 50th Annual Design Automation Conference*, page 174. ACM, 2013.

[50] Odroid xu3. Available:https://wiki.odroid.com/old_product/odroid-xu3/hardware/xu3_hardware.

[51] Pi-Cheng Hsiu, Po-Hsien Tseng, Wei-Ming Chen, Chin-Chiang Pan, and Tei-Wei Kuo. User-centric scheduling and governing on mobile devices with big. little processors. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):17, 2016.

[52] Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. Distrm: distributed resource management for on-chip many-core systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 119–128. ACM, 2011.

[53] Peter Zipf, Gilles Sassatelli, Nurten Utlu, Nicolas Saint-Jean, Pascal Benoit, and Manfred Glesner. A decentralised task mapping approach for homogeneous multiprocessor network-on-chips. *International Journal of Reconfigurable Computing*, 2009:3, 2009.

[54] Ahsan Shabbir, Akash Kumar, Bart Mesman, and Henk Corporaal. Distributed resource management for concurrent execution of multimedia applications on mpsoc platforms. In *2011 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 132–139. IEEE, 2011.

[55] Guilherme Castilhos, Marcelo Mandelli, Guilherme Madalozzo, and Fernando Moraes. Distributed resource management in noc-based mpsocs with dynamic cluster sizes. In *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 153–158. IEEE, 2013.

[56] Al Faruque, Mohammad Abdullah, Rudolf Krist, and Jörg Henkel. Adam: run-time agent-based distributed application mapping for on-chip communication. In *Proceedings of the 45th annual Design Automation Conference*, pages 760–765. ACM, 2008.

[57] Maximilian Götzinger, Amir M Rahmani, Martin Pongratz, Pasi Liljeberg, Axel Jantsch, and Hannu Tenhunen. The role of self-awareness and hierarchical agents in resource management for many-core systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 53–60. IEEE, 2016.

[58] Wei Quan and Andy D Pimentel. A hierarchical run-time adaptive resource allocation framework for large-scale mpsoc systems. *Design Automation for Embedded Systems*, 20(4):311–339, 2016.

[59] André Luís del Mestre Martins, Alzemiro Henrique Lucas da Silva, Amir M Rahmani, Nikil Dutt, and Fernando Gehm Moraes. Hierarchical adaptive multi-objective resource management for many-core systems. *Journal of Systems Architecture*, 97:416–427, 2019.

[60] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*, pages 71–80. ACM, 2012.

[61] Andy D Pimentel, Cagkan Erbas, and Simon Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE transactions on computers*, 55(2):99–112, 2006.

[62] Paul Lieverse, Pieter Van Der Wolf, Kees Vissers, and Ed Deprettere. A methodology for architecture exploration of heterogeneous signal processing systems. *Journal of VLSI signal processing systems for signal, image and video technology*, 29(3):197–207, 2001.

[63] A. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transactions on Computers*, 55(2):99–112, 2006.

[64] Wei Quan and Andy D Pimentel. Towards self-adaptive mpsoc systems with adaptivity throttling. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, pages 157–164. IEEE, 2015.

[65] Alan L Davis and Robert M Keller. Data flow program graphs. 1982.

[66] Edward A Lee and David G Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[67] Sdf3. Available:http://www.es.ele.tue.nl/sdf3.

[68] Anup Das, Akash Kumar, and Bharadwaj Veeravalli. Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 689–694. IEEE, 2013.

[69] Maher Fakih, Kim Grüttner, Martin Fränzle, and Achim Rettberg. Towards performance analysis of sdfgs mapped to shared-bus architectures using model-checking. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1167–1172. EDA Consortium, 2013.

[70] Torres Lionel, Benoit Pascal, Sassatelli Giles, and Robert Michel. An introduction to multicore system on chip. trends and challenges. multiprocessor system-on-chip: Hardware design and tool integration. pag. 1-18, 2010.

[71] Anantha P Chandrakasan, Samuel Sheng, and Robert W Brodersen. Low-power cmos digital design. *IEICE Transactions on Electronics*, 75(4):371–382, 1992.

[72] Jing Mei, Kenli Li, Jingtong Hu, Shu Yin, and Edwin H-M Sha. Energy-aware preemptive scheduling algorithm for sporadic tasks on dvs platform. *Microprocessors and Microsystems*, 37(1):99–112, 2013.

[73] Steven M Martin, Krisztian Flautner, Trevor Mudge, and David Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 721–725. ACM, 2002.

[74] Ina231 sensors. Available:http://www.ti.com/lit/ds/symlink/ina231.pdf.

[75] Simon Holmbacka, Erwan Nogues, Maxime Pelcat, Sébastien Lafond, Daniel Menard, and Johan Lilius. Energy-awareness and performance management with parallel dataflow applications. *Journal of Signal Processing Systems*, 87(1):33–48, 2017.

[76] Ravindra Jejurikar, Cristiano Pereira, and Rajesh Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st annual Design Automation Conference*, pages 275–280. ACM, 2004.

[77] Santiago Pagani, Anuj Pathania, Muhammad Shafique, Jian-Jia Chen, and Jörg Henkel. Energy efficiency for clustered heterogeneous multicores. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1315–1330, 2017.

[78] Vinicius Petrucci, Orlando Loques, Daniel Mossé, Rami Melhem, Neven Abou Gazala, and Sameh Gobriel. Energy-efficient thread assignment optimization for heterogeneous multicore systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(1):15, 2015.

[79] Simon Holmbacka, Mohammad Fattah, Wictor Lund, Amir-Mohammad Rahmani, Sébastien Lafond, and Johan Lilius. A task migration mechanism for distributed many-core operating systems. *The Journal of Supercomputing*, 68(3):1141–1162, 2014.

[80] Eduardo Wenzel Brião, Daniel Barcelos, Fabio Wronski, and Flávio Rech Wagner. Impact of task migration in noc-based mpsocs for soft real-time applications. In *2007 IFIP International Conference on Very Large Scale Integration*, pages 296–299. IEEE, 2007.

[81] IEEE computer society. IEEE standard SystemC language reference manual. IEEE Std. 1666–2011, 9 2011.

[82] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *[1990] Proceedings 11th Real-Time Systems Symposium*, pages 182–190. IEEE, 1990.

[83] Preesm. Available:https://preesm.github.io/about/.

[84] Spider. Available:https://github.com/preesm/spider/releases/tag/v1.4.0.

[85] Hugo Miomandre, Julien Hascoët, Karol Desnos, Kevin Martin, Benoît Dupont de Dinechin, and Jean-François Nezan. Demonstrating the spider runtime for reconfigurable dataflow graphs execution onto a dma-based manycore processor. 2017.

[86] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, Vijaykrishnan Narayanan, et al. Leakage current: Moore. *computer*, (12):68–75, 2003.

Appendix A.

Personal Publications

International Conferences:

- **Yang, S.**, Le Nours, S., mendez Real, M., Pillement, S. (2019, July). System-Level Modeling and Simulation of MPSoC Run-Time Management using Execution Traces Analysis. In International Conference on Embedded Computer Systems (pp. 281-293). Springer, Cham.
- **Yang, S.**, Le Nours, S., mendez Real, M., Pillement, S. (2019, October). Mapping and Frequency Joint Optimization for Energy Efficient Execution of Multiple Applications on Multicore Systems. In 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP) (pp. 29-34). IEEE.

Journal:

- Hierarchical run-time management for energy efficiency of heterogeneous cluster-based multi/many-core systems (under modification)

Other Scientific Communications:

- **Presentation** - The Collaborative Workshop on Model-based Design of Signal and Information Processing Systems (COWOMO 2018, June), Rennes, FRANCE.
- **Poster** - Ecole d'hiver Francophone sur les Technologies de Conception des Systemes embarques Heterogenes (FETCH 2018, January), Saint-Malo, FRANCE

Appendix B.

Notations

- app_i represents a possible running application.
- $T_{app_i} = \{t_{1,i}, t_{2,i}, \cdots, t_{H,i}\}$ denotes a set of computation tasks for app_i, where a task is indexed by $t_{h,i}$ and H is the total number of tasks in the application.
- $E_{app_i} = \{e_{1,i}, e_{2,i}, \cdots, e_{G,i}\}$ denotes a set of communication edges for app_i, where an edge is indexed by $e_{g,i}$ and G is the total number of edges in the application.
- $Period_{app_i}$ represents the period of app_i.
- $u_m = \{app_1, app_2 \cdots, app_I\}$ denotes a set of active applications in use-case u_m, indexed by app_i.
- I represents the number of active applications in a use-case.
- $cluster_j$ represents a cluster of the cluster-based multi/many-core platform.
- J represents the total number of clusters on the considered cluster-based platform.
- N_j represents the total number of cores in $cluster_j$.
- $cluster_r$ represents a reference cluster of the cluster-based platform.
- R_j^{power} represents the power ratio of $cluster_j$ compared to $cluster_r$.
- R_j^{perf} represents the performance ratio of $cluster_j$ compared to $cluster_r$.
- $\{f_{j,1}, f_{j,2}, \ldots, f_{j,F_{max}}\}$ denotes the set of supported frequency levels of $cluster_j$.
- f_j represents the optimized frequency level of $cluster_j$.
- $CompTime_{h,i}(cluster_j, f_j)$ represents the computation time of $t_{h,i}$ executed on $cluster_j$ at f_j.

- $CommTime_{g,i}(cluster_j, f_j)$ represents the communication time of $e_{g,i}$ executed on $cluster_j$ at f_j.
- $Latency_{app_i}$ represents the latency of app_i.
- $MAF^c_{i,r}$ represents the Minimum Allowed Frequency of app_i executed on the reference cluster $cluster_r$ using c numbers of cores.
- $MAF^c_{i,j}$ represents the Minimum Allowed Frequency of app_i executed on $cluster_j$ using c numbers of cores.
- $\mathbf{A} = [a_{i,j}]_{I \times J}$ denotes the matrix of application-to-cluster assignment.
- $X^c_{app_i} = \{x_{s\_t_{h,i}}(1), x_{e\_t_{h,i}}(1), \cdots x_{s\_t_{h,i}}(k), x_{e\_t_{h,i}}(k)\}$ represents the execution trace of the design-time prepared mapping for app_i mapped on c cores. $x_{s\_t_{h,i}}$ and $x_{e\_t_{h,i}}$ respectively denote the start time and end time of $t_{h,i}$, and k refers to the k^{th} instance of a given task. Note that when only one mapping is prepared (in Chapter 5), X_{app_i} is used to denote the prepared mapping for simplify reason.
- $X'_{Apps}(u_m)$ represents the execution trace of the combined mapping for active applications in u_m.
- f_0 represents the reference frequency that used to get some design-time information (*e.g.*, computation time, power).
- $\xi_{h,i,j}$ represents the power coefficient of $t_{h,i}$ executed on $cluster_j$.
- $P_{h,i}(cluster_j, f_j)$ represents the dynamic power of $t_{h,i}$ executed on $cluster_j$ at f_j.
- $E_{g,i}(cluster_j, f_j)$ represents the dynamic energy of $e_{g,i}$ executed on $cluster_j$ at f_j.
- $P^{avg}_{app_i}(cluster_j, f_j)$ represents the average dynamic power of app_i executed on $cluster_j$ at f_j.
- P^{avg}_{sys} represents the average dynamic power of all the active application on systems.
- $hyper - period$ represents the hyper-period of active applications, which is the Least Common Multiple (LCM) of the periods of active applications.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

Titre : Gestion en ligne pour l'efficacité énergétique des systèmes multi- cœurs/multi- coeurs

Mots clés : Gestion en ligne, efficacité énergétique, allocation des tâches, DVFS par cluster

Résumé : Les plates-formes multi/multi-cœurs organisées en clusters représentent des solutions prometteuses pour fournir des performances de calcul élevées et une efficacité énergétique optimisée dans les systèmes embarqués modernes. Ces plates-formes prennent souvent en charge la gestion dynamique tension/fréquence (DVFS) par cluster, ce qui permet à différents clusters de modifier leurs propres niveaux tension/ fréquence indépendamment. La complexité et le dynamisme croissants des applications sur ces plates-formes rendent nécessaire la gestion en ligne des ressources. Ce mémoire se concentre sur les méthodes de gestion en ligne des applications sur des systèmes multi/multi-cœurs en cluster pour améliorer l'efficacité énergétique.

Cette thèse présente différentes stratégies de gestion qui permettent d'estimer l'influence mutuelle entre l'allocation des applications et la configuration en tension/fréquence des clusters afin d'obtenir respectivement une optimisation locale au sein d'un cluster et une optimisation globale dans l'ensemble du système. Les stratégies proposées permettent d'obtenir des solutions de gestion optimisées avec moins de complexité que les stratégies existantes. De plus, cette thèse présente une nouvelle approche de modélisation et de simulation qui permet d'évaluer les stratégies de gestion en ligne dans les systèmes multi/multi-cœurs pour garantir que les contraintes du système sont pleinement respectées. L'approche de simulation proposée est validée à l'aide d'un cadre de modélisation et de simulation industrielle.

Title : Run-Time Management for Energy Efficiency of Cluster-based Multi/Many-Core Systems

Keywords : Run-time management, energy efficiency, task mapping, per-cluster DVFS

Abstract : Cluster-based multi/many-core platforms represent promising solutions to deliver high computing performance and energy efficiency in modern embedded systems. These platforms often support per-cluster Dynamic Voltage/Frequency Scaling (DVFS), allowing different clusters to change their own v/f levels independently. The increasing application complexity and application dynamism on such platforms arise the need for run-time management. This dissertation focuses on the run-time management of applications on cluster-based multi/many-core systems to improve energy efficiency.

Towards the run-time management purpose, this dissertation presents different management strategies that estimate the mutual influence between application mapping and cluster v/f configurations to respectively achieve local optimization within a cluster and global optimization in the overall system. The proposed management strategies can achieve near-optimal management solutions with less strategy complexity compared to state-of-the-art strategies. In addition, this dissertation presents a new modelling and simulation approach that allows the evaluation of run-time management strategies in multi/many-core systems to guarantee that system constraints are fully met. The proposed simulation approach is validated using an industrial modelling and simulation framework.