# 1

# Introduction

The availability of highly parallel computing platforms based on multi and manycore processors enables a wide variety of technical solutions for systems across the embedded and high-performance computing domains. However, large scale manycore systems are notoriously hard to design and manage, and choices regarding resource allocation alone can account for wide variability in timeliness and energy dissipation, up to several orders of magnitude. For example, the allocation of many computation-centric jobs to the same processing core, or communication-intensive jobs to cores linked by a low bandwidth interconnect, can significantly impair system performance specially in applications with many dependencies between jobs.

Techniques to allocate computation and communication workloads onto processor platforms have been studied since the early days of computing. However, this problem has become significantly harder because of **scale** and **dynamicity**: compute platforms now integrate hundreds to thousands of processing cores, running complex and dynamic applications that make it difficult foresee the amount of load they can impose to those platforms.

Elementary combinatorics provides us with evidence of the problem of **scale**. For a simple formulation of the problem of allocating jobs to processors (one-to-one allocation), one can see that the number of allocations grows with the factorial of the number of jobs and processors. For example, a system with 4 jobs and 4 processing cores can have $P(4,4) = 24$ possible allocations, but simply by doubling the number of jobs and cores the number of allocations becomes $P(8,8) = 40320$ (where $P(n,k)$ denotes the k-permutations of n). The empirical evidence points in the same direction, as it can be seen in [110] that for realistic manycore embedded systems (40–60 jobs, 15–30 processing cores) a well-tuned search algorithm had to statically evaluate hundreds of thousands of distinct allocations before it finds one that meets the systems performance requirements.

To cope with **dynamicity**, a dynamic approach to resource management is the most obvious choice, aiming to dynamically learn and react to changes to the load characteristics and to the underlying compute platform. The baseline, which is a static allocation decided before deployment based on the (nearly)

complete knowledge about the load and the platform, is no longer viable. For example, static resource allocation in high-performance computing (HPC) has often been referred as a significant cause of low utilization of servers, which results in cost increases on hardware and energy [17]. Static allocation is also commonly used by aerospace and automotive industries to provide worst case performance guarantees that are required by certification authorities. However, it is well known that such an approach usually leads to under-utilised computing and communication resources at run-time [113].

The problems of scale and dynamicity are also made harder with the increasing density of computing and communication resources. The definition of density used here is not necessarily spatial, but rather on connectivity (i.e., dense graph). In densely connected systems, a resource allocation algorithm may have to make decisions very often due to the system dynamics, and may have to consider dozens or hundreds of potential allocation possibilities at each decision point (i.e., which processor should execute each job, which communication links should be used when those jobs exchange data). Furthermore, such algorithms have to work in a distributed way due to the difficulty to obtain the up-to-date state of the whole system. And despite such levels of complexity, the algorithms themselves are also subject to tight constraints in performance and energy. It is then evident that optimal resource allocation algorithms cannot cope with this type of problem, and that lightweight heuristic solutions are needed.

This book is therefore concerned with the kinds of resource allocation heuristics that can cover different levels of dynamicity, while coping with the scale and complexity of high-density manycore platforms.

## 1.1 Application Domains

The level of dynamicity of a system denotes how often it changes its characteristics. In this book, we are concerned with resource allocation, so dynamicity means how much variation can be found on the system workload (e.g., arrival patterns, computation and communication requirements, value to the end-user) and on the underlying compute platform (e.g., degradation or lost of performance due to faults, increase in capacity due to upgrades). Different application domains can be characterised by their typical levels of dynamicity.

For example, deeply embedded systems such as those in automotive, aerospace and medical domains have low dynamicity, and often their entire functionality and behaviour is known at design time, prior to deployment. The low dynamicity makes the performance of such systems easier to predict, and therefore guarantees regarding timeliness can be made (e.g., ECG signal of a

complete cardiac cycle will be processed in less than 10 ms). Such guarantees are often enforced by means of resource reservation and isolation, which can lead to very low levels of resource utilisation: a processing core can be exclusively allocated to a given job for the sake of performance predictability, but that job only needs the core to its full capacity for a limited period of its lifetime, leaving it subutilised for the rest of the time.

On the other hand, HPC and cloud computing have high dynamicity due to the wide variety of workloads they have to handle. That makes it harder to make performance guarantees, because one never knows what comes next. And due to the cost of deploying and maintaining such platforms, they are often only viable if operated at saturation point, with nearly 100% utilisation, which undermines performance guarantees even further by making nearly impossible to rely on resource reservation or isolation.

Figure 1.1 below shows both domains, embedded and HPC/cloud over the dimensions of dynamicity, typical resource utilisation and the ability to sustain performance guarantees. State-of-the-art resource allocation in the embedded domain is static, relying on the low dynamicity of those systems and producing allocations that can be derived at design time and used for the whole lifetime of the system, while ensuring the performance requirements are met even in worst case scenario. For HPC and cloud, the resource allocation is completely dynamic and often based on instantaneous metrics such as order of arrival of jobs and current utilisation of cores, which can certainly keep the platforms running at saturation point but cannot offer any performance guarantees.

Recently, the dichotomy described above became less visible. Embedded systems are becoming increasingly complex, having to cope with dynamic workloads, and using less predictable platforms (i.e., multi-level caches, speculative execution), while still having to fulfil strict performance requisites. HPC and cloud computing, in turn, critically need to address fundamental problems in energy efficiency and performance predictability, as they become more widespread and critical to our daily lives. This points to the importance

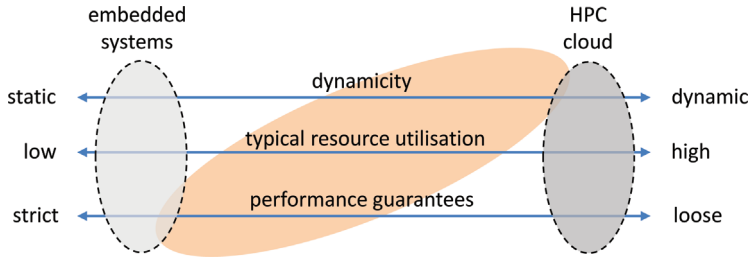


**Figure 1.1** Application domains and their characteristics with regard to dynamicity, resource utilisation and performance predictability.

of the areas in the central part of Figure 1.1, which represents increasingly dynamic embedded systems and predictable HPC and cloud systems.

The goal of this book is to identify and present resource allocation heuristics that can be used to achieve different levels of performance guarantees, and that can cope with different levels of dynamicity of the application workload.

## 1.2 Related Work

The problem of allocating tasks to platform elements is a classic problem in multiprocessor and distributed systems. Most formulations of this problem cannot be solved in polynomial time, and many of them are equivalent to well known NP problems such as graph isomorphism [18] and the generalised assignment problem [58].

This problem was first addressed from the cluster/grid point of view, but more recently the fine-grained allocation of tasks within manycore processors has also received significant attention due to its critical impact on performance and energy dissipation. In the following subsections, we consider allocation mechanisms at both grid and manycore CPU level, and review the most significant trends and achievements in terms of guaranteed performance and energy efficiency.

### 1.2.1 Allocation Techniques for Guaranteed Performance

There are numerous multiprocessor scheduling and allocation techniques that are able to meet real-time constraints, each of them under a different set of assumptions. A very comprehensive survey is given by [41], covering techniques that can be applied both at the grid or many-core level, but all of them assume that the platform is homogeneous and tasks are independent (i.e., do not explicitly consider communication costs). Many of them also assume that the allocation is done statically, or do not take into account the overheads of dynamically allocating and migrating tasks (i.e., context saving and transferring). In [96], heterogeneous platforms are considered but communication costs and overheads are still not taken into account.

Significant research on resource reservation has been done, aiming to increase time-predictability of workflow execution over HPC platforms [90]. Many approaches use a priori workflow profiling and use estimation of task execution times and communication volumes to plan ahead which resources will be needed when tasks become ready to execute. Just like in static allocation, resource reservation policies significantly reduce the utilisation of HPC platforms. A reduction of 20–40% in the utilisation is not unusual [150].

Allocation and scheduling heuristics based on feedback control have been used in HPC systems [44–83], aiming to improve platform utilisation without sacrificing performance constraints. Most cases concentrate on controlling the admission and allocation of tasks over the platform based on a closed-loop approach that monitors utilisation of the platform as well as performance metrics such as task response times [54].

Many cloud-based and grid-based HPC systems use allocation and scheduling heuristics that take into account not only the timing constraints of the tasks but also their value (economic or otherwise). This problem been well-studied under the model of Deadline and Budget Constraints (DBC) [27], where each task or taskflow has a fixed deadline and a fixed budget. State-of-the-art allocation and scheduling techniques target objectives such as maximising the number of tasks completed within deadline and/or budget [139], maximising profit for platform provider [76] or minimising cost to users [130] while still ensuring deadlines. Several approaches to the DBC problem use market-inspired techniques to balance the rewards between platform providers and users [154]. A comprehensive survey given in [157] reviews several market-based allocation techniques supporting homogeneous or heterogeneous platforms, some of them supporting applications with dependent tasks modeled as DAGs.

At the many-core level, there are a few allocation techniques that take into account both the computation and communication performance guarantees. Such techniques are tailored for specific platforms e.g., many-cores based on Network-on-Chip (NoC). To guarantee timeliness, all state-of-the-art approaches rely on a static allocation of tasks and communication flows. In [6], a multi-criteria genetic algorithm is used to evolve task allocation templates over a NoC-based many-core aiming to reduce their average communication latency. The approach in [110] also used a genetic algorithm that could find an allocation that can meet hard real-time guarantees on end-to-end latency of sporadic tasks and communication flows over many-cores that use priority-preemptive arbitration. Stuijk [136] proposed a constructive heuristic to do static allocation of synchronous dataflow (SDF) application models [133], which constraint all tasks to read and write the same number of data tokens every time they execute. The allocation guarantees the timeliness of the application if the platform provides fixed-latency point-to-point connection between processing units. In [161], the same author relaxes some of the assumptions of SDF applications (i.e., allows for changes on token production and consumption rates during runtime) and proposes analytical methods to evaluate worst-case throughput and to find upper bounds for buffering for a given static allocation.

### 1.2.2 Allocation Techniques for Energy-efficiency

Most allocation techniques addressing energy efficiency operate at the many-core processor level, mainly because of the difficulties of dealing with energy-related metrics at larger system granularities.

Hu et al. [60] and Marcon et al. [88] estimate the energy consumption according to the volume of data exchanged by different application tasks over the interconnection network. Such approaches lack in accuracy as they do not take into account runtime effects such as network congestion or time-varying workloads. Thus, research approaches on energy-aware dynamic allocation techniques have been proposed.

In [129], an iterative hierarchical dynamic mapping approach aims to reduce energy consumption of the system while providing the required QoS. In such strategy, tasks are firstly grouped by assigning them to a system resource type (e.g., FPGA, DSP, ARM), according to performance constraints. Then, each task within a group is mapped, minimising the distance among them and reducing communication cost. Finally, the resulting mapping is checked, and if it does not meet the application requirements, a new iteration is required.

Chou and Marculescu [37] introduce an incremental dynamic mapping process approach, where processors connected to the NoC have multiple voltage levels, while the network has its own voltage and frequency domain. A global manager (OS-controlled mechanism) is responsible for finding a contiguous area to map an application, and for defining the position of the tasks within this area, as well. According to the authors, the strategy avoids the fragmentation of the system and aims to minimize communication energy consumption, which is calculated according to Ye et al. [155]. This work was extended in [36, 38], incorporating the user behaviour information in the mapping process. The user behaviour corresponds to the application profile data, including the application periodicity in the system and data volume transferred among tasks. For real applications considering the user behaviour information, the approach achieved around 60% energy savings compared to a random allocation scenario.

Holzenspies et al. [58] investigate a run-time spatial mapping technique with real-time requirements, considering streaming applications mapped onto heterogeneous MPSoCs. In the proposed work, the application remapping is determined according to information that is collected at design time (i.e., latency/throughput), aiming to satisfy the QoS requirements, as well as to optimize the resources usage and to minimise the energy consumption. A similar approach is proposed in Schranzhofer et al. [120], merging pre-computed template mappings (defined at design time) and online decisions that define newly arriving tasks to the processors at run-time. Compared to the static-mapping approaches, obtained results reveal that it is possible to

achieve an average reduction on power dissipation of 40–45%, while keeping the introduced overhead to store the template mappings as low as 1 KB.

Another energy-aware approach is presented in Wilderman et al. [151]. This approach employs a heuristic that includes a Neighborhood metric inspired by rules from Cellular Automata, which allows decreasing the communication overhead and, consequently, the energy consumption imposed by dynamic applications. Lu et al. [85] propose a dynamic mapping algorithm, called Rotating Mapping Algorithm (RMA), which aims to reduce the overall traffic congestion (take in account the buffer space) and communication energy consumption of applications (reduction of transmission hops between tasks).

In turn, Mandelli et al. [87] propose a power-aware task mapping heuristic, which is validated using a NoC-based MPSoC described at a cycle-accurate level. The mapping heuristic is performed in a given processor of the system that executes a preemptive operating system. Due to the use of a low level description, accurate performance evaluation of several heuristics (execution time, latency, energy consumption) is supported. However, the scope of the work is limited to small systems configurations due to the long simulation time. In the previous works, only one task is assigned to each processing core. A multi-task dynamic mapping approach was proposed in [128]. Singh et al. [128] extends the work described in [32], which evaluates the power dissipation as the product of number of bits to be transferred and distance between source-destination pair.

Research in energy-efficient allocation for HPC and cloud systems is still incipient, with existing works addressing only the time and space fragmentation of resource utilisation at a very large granularity (server level), aiming to minimise energy by rearranging the load and freeing servers that are then turned off [12, 101].

## 1.3 Challenges

While the approaches mentioned in the previous section have presented sophisticated resource allocation approaches that can provide performance guarantees and/or improve energy efficiency, there are still challenges that require more advanced resource allocation approaches. The following subsections briefly describe some of those challenges, which are precisely the ones addressed in this book.

### 1.3.1 Load Representation

Load models are internal representations used by allocation algorithms to evaluate different allocation alternatives. Such models may use information that is available a priori about the load (such as job dependencies,

communication volumes, worst case execution times), but can be also extended with information obtained during runtime (e.g., actual execution and communication times). In dynamic resource allocation, it is very challenging to define a load model that includes sufficient information about static and dynamic characteristics of the load, and that is lightweight enough to be used by allocation heuristics to quickly evaluate and compare alternative allocation possibilities during runtime.

Chapter 2 addresses this challenge and presents a load model based on an interval algebra, aiming to allow quickly compose the load of multiple computation and communication jobs (represented as series of time intervals), enabling the evaluation of the impact of resource allocation (and thus resource sharing) on system performance and timeliness.

### 1.3.2 Monitoring and Feedback

In large-scale systems, obtaining updated information about the load during runtime is not trivial. Often, such information only makes sense when coupled with information about the underlying computation and communication platform. Furthermore, the costs of monitoring and transferring all such data to the resource allocation mechanism is already prohibitive. The major challenge in such scenarios is then to define a sufficiently meaningful set of metrics to monitor, and to design algorithms that can make meaningful resource allocation decisions based on the changes on those metrics over time.

Feedback control algorithms have been used for decades to make decisions based on time-series data, so in Chapters 3 and 4 we describe possible uses of such closed-loop algorithms to support resource allocation. In Chapter 3, we show that they can be used to increase throughput and energy-efficiency in HPC and cloud workloads. In Chapter 4, on the other hand, we show that it can be used to efficiently perform admission control tasks, aiming to maximise system utilisation without jeopardising predictability in performance-sensitive HPC applications.

### 1.3.3 Allocation of Modal Applications

Allocation heuristics may have to guarantee hard real-time constraints to critical jobs. This is possible for applications that have been profiled a priori so their execution and communication patterns can be accurately represented by an accurate load model. Such applications will not be highly dynamic, and will exhibit modal behaviour, so that distinct modes of operation can be analysed at design time, so the dynamic allocation can be based on pre-defined alternatives (thus the number of allocation decisions during runtime is minimal).

To address such scenario, modal allocation heuristics can guarantee hard real-time constraints by allowing different different allocations for each

operation mode while minimising the amount of remappings during mode transitions. Chapter 5 describes search-based heuristics that identify allocations that are optimised for specific operation modes, but also for coping with dynamic mode changes. It uses automotive applications and Network-on-Chip platforms as case studies, and shows that it is possible to guarantee hard real-time constraints during each of the system's modes as well as during transitions.

### 1.3.4 Distributed Allocation

In closed-loop systems, a centralised resource manager continuously receive feedback from the system so that it can have an up-to-date representation of its state. This usually comes with a significant communication overhead, specially in large-scale systems. Fully distributed approaches, on the other hand, offer higher scalability by relying on decision-making done by individual system components using only locally-available information. However, due to the lack of global knowledge, it is harder to achieve a reasonable level of performance predictability.

Chapter 6 presents a bioinspired approach based on the notion of swarm intelligence, aiming to support a fully distributed approach to load remapping. It can be used on its own or in conjunction with centralised approaches, aiming to fine-tune allocation decisions based on up-to-date local data. A case study based on multi-stream video processing over Network-on-Chip platforms shows the strengths and weaknesses of such approach.

### 1.3.5 Value-based Allocation

Many of the quality metrics associated to resource allocation in HPC and cloud computing are platform-specific. For instance, metrics that are often used to formulate optimisation objectives (such as job execution times, communication volumes and throughput) are not comparable across different computational platforms. There are other metrics, however, that are completely independent of the computational platform and relate instead to the requirements of the end-user. One of such metrics is the value of the completion of a job. This can be seen as a simple value, perhaps associated to a particular currency. More commonly, such value will be a function of time: the result of a job is very likely to lose value over time, and can even become worthless if it takes too long to be obtained.

Chapter 7 addresses resource allocation heuristics that are designed to optimise such time-varying notion of value. It presents approaches that can be configured to rely more or less on load models obtained in advance, and shows how much can be gained in value if these models are available.