CHAPTER 1

INTRODUCTION

Kallia: Definition of AMCs: Asymmetric Multi-core (AMC) systems architecture is an interesting case of heterogeneous multi-core architecture. This multi-core systems architecture features cores with different microarchitectures but with a single Instruction Set Architecture (ISA) and potentially shared memory resources. Figure 1.1 shows a classification of multi-core systems and how AMCs relate to the rest of the multi-cores. Multi-core systems can be either homogeneous or heterogeneous. A homogeneous multi-core consists of multiple identical processing cores that have the same micro-architecture and instruction set architecture. A heterogeneous multi-core can fall into two categories. The first category consists of multi-cores with different micro-architectures and ISA; systems with GPUs or other compute accelerators fall in this category. The second category consists of of multi-cores that have different micro-architecture but a single ISA; AMC systems fall in this category.

AMCs have been used in different areas of parallel computing. In recent years they made their appearance in the mobile market and are nowadays the most commonly used architecture for mobile processors. The most widely used AMC architecture for mobile processors is the Arm big.LITTLE architecture [44] which combines low-power simple in-order cores (*little*) with fast out-of-order cores (*big*) to achieve high performance while keeping power dissipation low. Mobile processors are also utilized in HPC platforms aiming to energy savings [59]. Another area where AMCs have been successful is the supercomputing market. The Sunway TaihuLight supercomputer topped the Top500[1] list in 2016 using AMCs. In this setup, big cores, that offer support for speculation to exploit Instruction-Level Parallelism (ILP), run the master tasks such as the OS and runtime system. Little cores are equipped

[1]We refer to the list published on November 2017

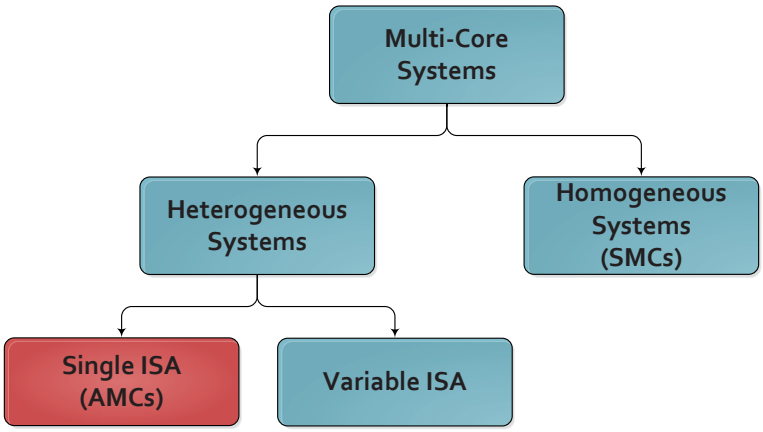

Figure 1.1 Classification of multi core system architectures

with wide Single Instruction Multiple Data (SIMD) units and lean pipeline structures for energy efficient execution of compute-intensive code.

Kallia: Challenges Using AMCs: AMCs are generally easy to program as the programmer does not need to express the functionality of an application for many different ISAs. However, there are still challenges that need to be addressed that are mainly related to the performance of the AMC. Like in other heterogeneous systems, load balancing and scheduling are fundamental challenges that must be addressed to effectively exploit all the resources in AMC platforms [42, 44, 51, 52, 76, 83]. To preserve load balance, the system has to make sure that the cores of the system remain busy and are fairly loaded depending on their type. For example running a barrier-based multi-threaded application where each thread runs on one core on an AMC might result in performance degradation as there is a high risk of starvation of the fast cores [66].

Additionally, choosing which task to execute in each type of core of an AMC is not as straightforward. Due to the different benefits offered by each core type, such decisions require runtime information, like task dependencies or whether a task is memory or computationally bound. An interesting approach against the above challenges is to take advantage of the heterogeneity of an AMC by providing novel asymmetry-aware schedulers. The state-of-the-art asymmetry-aware scheduling approaches perform thread scheduling within the Linux kernel. The most commonly used example in this category is the Global Task Scheduler (GTS) [31] that is implemented on top of the Linux Completely Fair Scheduler (CFS) and is aware of the asymmetry of the system. However, such approaches lack flexibility due to the high thread-migration overheads [66].

Kallia: Solutions – mention to Task-based models and process scheduling The above challenges can be tackled by using task-based parallel programming models. These parallel

```
1  #pragma omp task in(a) out(b)
2  void task1(int *a, int *b) {
3  *b = *a/2 + 3 * (*a);
4  return;
5  }
6  #pragma omp task in(c) inout(d)
7  void task2(int *c, int *d) {
8  *d = 3 * (*c) + (*d) / 2;
9  return;
10 }
11 int main() {
12 int x = 10, y = 5;
13 for(int i = 0; i <3; i++) {
14 task1(&x, &y);
15 task2(&y, &x);
16 }
17 return 0;
18 }
```

Listing 1.1 Example code using the OmpSs task-based programming model.

programming models have been widely used during the last decade in the development of parallel applications. They form an appealing solution as they significantly ease the parallel programming by providing a higher level of abstraction to the programmer through a directive-based or language-based interface. With these models challenges such as load balancing, scheduling, or thread migration costs are partially solved by using them out-of-the box. They allow the programmer to split the application in multiple work units called *tasks* (a function can be a task) that can potentially execute simultaneously[2]. As an example, Listing 1.1 shows a simple usage of a directive-based task-based programming model (the OmpSs programming model [9]). In this code, the *pragma omp task* directives define the functions that act as tasks, namely `task1` and `task2`. The input and output dependencies between these tasks are derived at execution time from the *in*, *out* and *inout* directionality clauses. When these task functions are being called by the main function, the runtime system creates tasks that can run in parallel if there are no task dependencies between them. The runtime system of the task-based programming model is responsible of maintaining software threads and distribute the tasks to the appropriate thread for execution. Typically the number of software threads that the runtime system maintains is equal to the number of available cores in the system and one thread is bound to each core. We further explain how the task-based runtime system functions in Section 2.3.

[2]They can execute simultaneously given there are no task dependencies.

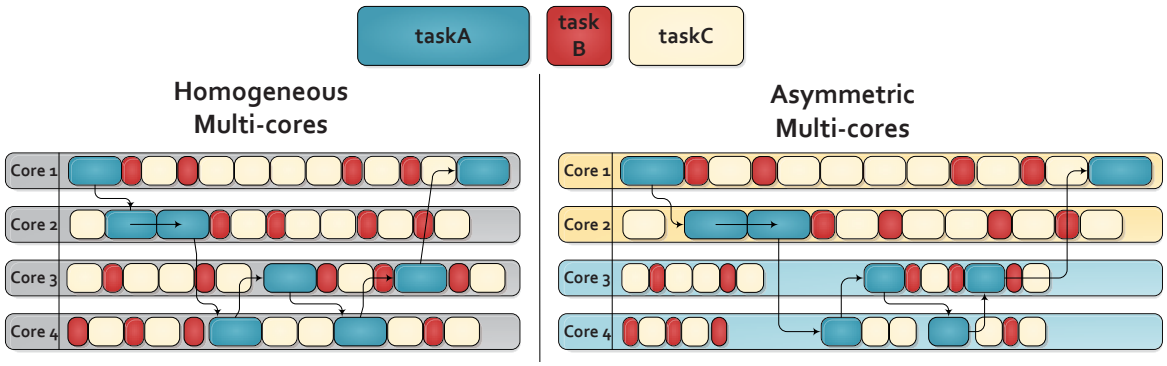

Figure 1.2 Increased idle time when moving from homogeneous multi-core to asymmetric multi-core. The directions of the arrows indicate the data dependencies.

1.1 Thesis Motivation

As was illustrated by the example of Listing 1.1 task-based programming models offer a very convenient abstraction layer to the programmer for parallelizing applications. However, these parallel programming languages exhibit some challenges when moving to asymmetric systems as their runtime system is platform agnostic. The current scheduling implementation in task-based programming models assumes that all tasks can be evenly distributed among the cores of the system and treats all tasks and cores as equal. In some cases, a bad scheduling decision on an asymmetric system can lead to significant performance degradation.

Figure 1.2 shows the execution representation of a simple example using a task-based programming model. This example consists of three types of tasks that have different execution times. In this example the tasks of type taskA have dependencies between them as the arrows indicate which means that the taskA tasks cannot execute simultaneously. The leftmost representation shows how a task-based programming model would execute this set of tasks on a homogeneous multi-core (where all cores are the same) while the representation on the right shows how the same example would execute on an asymmetric multi-core where the cores number three and four are faster than the cores number one and two. As we can see the idle time of the cores three and four is increased when using this scheduling but this is something that can be addressed if we provide a better scheduling policy in our task-based runtime system. Figure 1.3 shows one solution to improve this scheduling while respecting the inter-task dependencies.

Another challenge introduced by the task-based programming models is the significant task generation overhead. The high task generation overheads occur on both homogeneous and asymmetric multi-core systems. Spending a lot of time on task generation can result in

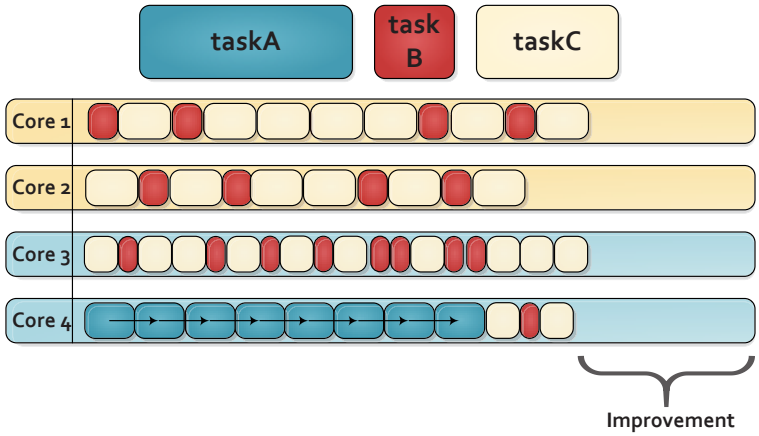

Figure 1.3 Changing the scheduling leads to improvement for AMC systems. The directions of the arrows indicate the data dependencies.

decreased performance as the scheduler is unable to detect the tasks and send them to the appropriate core. The leftmost representation of Figure 1.4 shows how the task generation affects the scheduling and increases the idle time. The black rectangles indicate the time spent to create one task and the number inside these boxes show the task that was just created. Looking at the yellow rectangles we can see that the execution of each task is delayed due to the delay in task generation. The rightmost part of Figure 1.4 shows how by reducing the task generation time scheduling of the tasks is improved and the CPU idle time is significantly reduced. This leads to inefficient schedules with increased idle time. Accelerating the task generation overheads increases performance of task-based programming models.

1.2 Thesis Contributions

Kallia: TODO: mention papers and summarize contributions

The performance of task-based programming models can be boosted by providing sophisticated scheduling policies that take into account the platform's asymmetry.

The contributions made in this thesis rely on the efficient utilization of asymmetric multicore systems. Figure 1.5 shows the research areas around performance of AMC systems. The areas shown in yellow color indicate the fields of our study. This thesis starts with a wide experimental evaluation of highly parallel applications on AMC systems. We then approach the two important challenges of using task-based programming models on AMC platforms. These challenges include first, the scheduling problem, and second the high task generation overheads introduced while using these parallel programming tools. Apart from scheduling within task-based programming models, this thesis includes the high-level

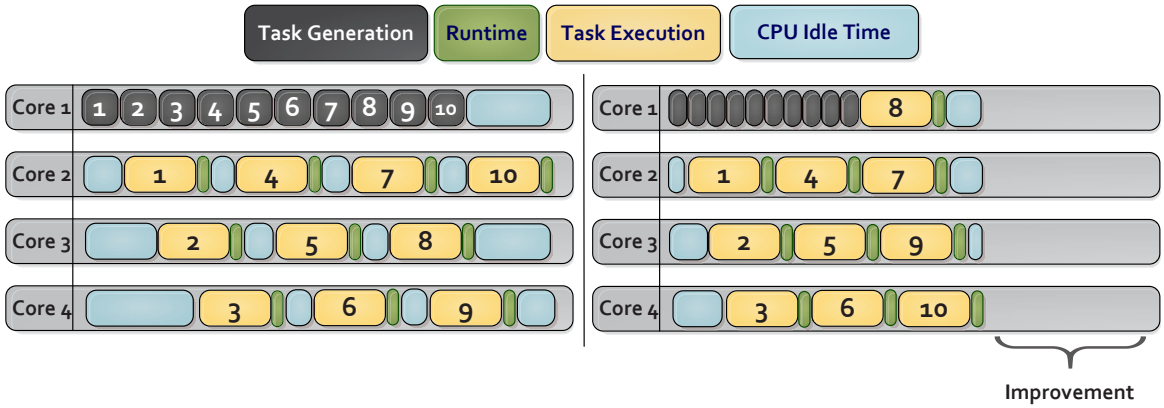

Figure 1.4 Task generation overheads leads to increased idle time. Numbers inside the boxes show the task ID that is either being created or executed

description of a thread scheduler for asymmetric systems targeting this time mobile devices. We tackle the above challenges and provide the following contributions:

1. A thorough study of the potential of the AMC systems when running out-of-the-box HPC applications. We compare scheduling on different levels of the software stack and we conclude that using a task-based programming model is indeed the most efficient solution as it allows the runtime system to maintain load balance even if the system is asymmetric. This study serves as a verification that for HPC, scheduling through task-based programming models is indeed more efficient compared to app-based scheduling, as shown on Figure 1.5.

2. The design, implementation and evaluation of three novel scheduling policies that are aware of the system's asymmetry. These schedulers have different criteria for rating the importance of the executing tasks of an application. According to their distinct criteria, each scheduler identifies the *critical tasks* of the application and executes them on the fast cores of the system. They then leave the *non-critical tasks* to be executed by the slower cores of the system. The research areas that were studied during at this part of the thesis are the scheduling and dependence analysis of the runtime system as shown on Figure 1.5.

3. The TaskGenX proposal, a hardware-software co-design scheme to reduce task generation overheads of task-based programming models. We implement the TaskGenX runtime system, that decouples the task generation from the other runtime activities and sends it for execution on the runtime optimized accelerator. Furthermore, we draw the requirements of the hardware accelerator in terms of performance with the hope to influence hardware designers for the implementation of such a component.

The research area that is related to this contribution is the resource allocation from Figure 1.5 of the runtime systems.

4. A high-level description of a thread scheduler for AMC systems that targets mobile devices. Our approach takes scheduling actions depending on the current temperature of the device and manages to increase the frames per second for three game applications while keeping the temperature stable. This study is briefly described due to NDA agreement as it was part of my internship in Samsung Electronics Research Institute, UK. This part of the thesis is within the Android framework as shown in Figure 1.5.

The publications that support this thesis are listed below:

1. **Kallia Chronaki**, Miquel Moreto, Marc Casas, Alejandro Rico, Rosa M. Badia, Eduard Ayguade, Mateo Valero: "On the Maturity of Parallel Applications for Asymmetric Multi-Core Processors", *under review, JPDC*.

2. [27] **Kallia Chronaki**, Miquel Moreto, Marc Casas, Alejandro Rico, Rosa M. Badia, Eduard Ayguade, Mateo Valero: "POSTER: Exploiting Asymmetric Multi-Core Processors with Flexible System Software". Poster presentation in International Conference on Parallel Architecture and Compilation Techniques (PACT) 2016, Haifa, Israel.

3. [30] **Kallia Chronaki**, Alejandro Rico, Rosa M. Badia, Eduard Ayguade, Jesus Labarta, Mateo Valero: "Criticality-Aware Dynamic Task Scheduling for Heterogeneous Architectures". Paper presentation in International Conference of Supercomputing (ICS) 2015, Newport Beach, California.

4. [28] **Kallia Chronaki**, Alejandro Rico, Marc Casas, Miquel Moreto, Rosa M. Badia, Eduard Ayguade, Jesus Labarta, Mateo Valero: "Task Scheduling Techniques for Asymmetric Multi-Core Systems". Published in IEEE Transactions on Parallel and Distributed Systems (TPDS) 2017 volume 28 number 7.

5. [29] **Kallia Chronaki**, Marc Casas, Miquel Moreto, Jaume Bosch, Rosa M. Badia "TaskGenX: A Hardware-Software Proposal for Accelerating Task Parallelism". Paper presentation in International Supercomputing Conference (ISC) 2018, Frankfurt, Germany.

The contributions of this paper have also been used as part of other relevant publication:

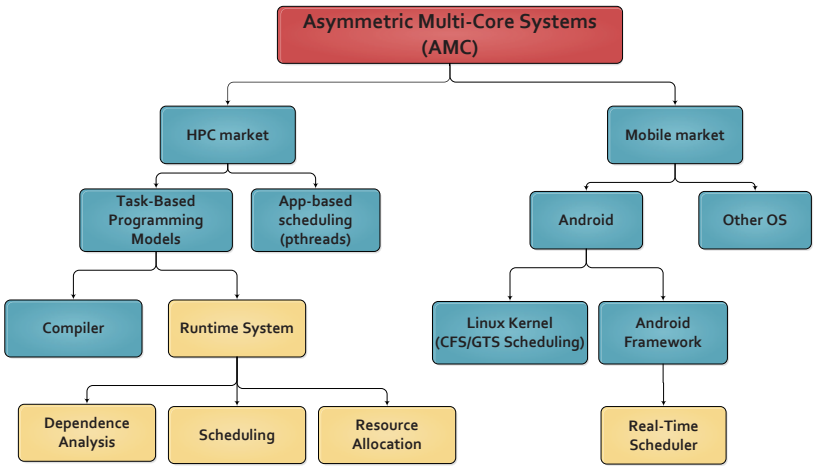

Figure 1.5 Research areas studied in this thesis

- [23] Emilio Castillo, Miquel Moreto, Marc Casas, Lluc Alvarez, Enrique Vallejo, **Kallia Chronaki**, Rosa M. Badia, Jose Luis Bosque, Ramon Beivide, Eduard Ayguade, Jesus Labarta, Mateo Valero: "CATA: Criticality Aware Task Acceleration for Multicore Processors". Paper presentation in IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2016.

1.3 Thesis Organization

The rest of this document is organized as follows: Chapter 2 presents the background of tis thesis. We first report the characteristics of AMC architectures, we then explain how task-based programming models are organized and how their runtime system operates, we explain a few things about the TaskSim simulator [79], that is our tool for evaluating the impact of our implementations on larger systems and finally we provide the list of applications used in our evaluation together with a short description for each one.

Chapter 3 provides our thorough and detailed study of highly parallel applications on asymmetric systems. This study is a proof that scheduling on the runtime system is the most efficient way of utilizing an AMC system. Chapter 4 describes and evaluates our three novel scheduling approaches (CATS, CPATH and HYBRID). We implement these approaches within the OmpSs programming model and evaluate them using real scientific applications. To see their impact on larger systems we use TaskSim simulator.

Chapter 5 presents our software-hardware co-design proposal, TaskGenX. We show the implementation done and we evaluate our proposal using TaskSim simulator and real applications. Chapter 6 includes the thread-based scheduling within the Android framework

together with its evaluation on a mobile device using three games with intensive graphics. Chapter 7 presents our related work and Chapter 8 concludes this thesis.