TÉCNICO LISBOA

Real-Time Processing and Control on Multicore Mobile Devices

Francisco Gaspar

Relatório da disciplina de Introdução à Investigação e ao Projecto em Engenharia Electrotécnica e de Computadores

Júri

Presidente:	Doutor João Vaz
Orientador:	Professor Leonel Sousa
Co-orientador:	Doutor Pedro Tomás
Vogal:	Doutor Carlos Almeida

Janeiro de 2014

Abstract

With an ever growing demand for energy efficient systems, heterogeneity has found its way into processors through the combination of high performance and energy efficient cores. The scheduling of tasks on these systems presents a challenge since it is desirable to maintain performance while saving power. Therefore, this thesis aims to analyse current scheduling techniques applicable to heterogeneous systems, and to propose new methods and topologies for deciding where and when to schedule tasks with different characteristics (both Quality of Service (QoS) and non-QoS ones). The new scheduling method should combine application dictated metrics (for QoS) and hardware based ones (i.e. performance counters, for non-QoS).

This particular report focuses on the state of the art of such techniques and on the planning of the work, which will target the ARM big.LITTLE architecture for embedded devices.

Keywords

Heterogeneous multi processor; transparent scheduling; embedded systems; quality of service; big.LITTLE

Resumo

Com uma crescente procura por sistemas eficientes energicamente, os processadores começam a apresentar soluções heterogéneas combinando núcleos de alta performance com núcleos eficientes de baixo consumo de potência. O agendamento de tarefas nestes sistemas constitui um desafio para manter o desempenho dos processadores com reduzido consumo energético. Assim, esta tese pretende analisar técnicas actuais de agendamento de tarefas aplicáveis a sistemas heterogéneos e propor um novo método e topologia capazes de decidir onde e quando agendar tarefas com diferentes características, tanto as que pretendem manter um certo nível de qualidade de serviço (QoS) como as que não apresentam tal restrição. A nova topologia deve portanto combinar métricas ditadas pela aplicação (para tarefas com níveis de qualidade a respeitar) com métricas de hardware (como contadores de desempenho para as que não têm tais exigências).

Este relatório foca-se no estado da arte de técnicas de agendamento e no planeamento do trabalho que irá ter como objecto de estudo a arquitectura big.LITTLE para sistemas embebidos heterogéneos desenvolvida pela ARM.

Palavras Chave

Processadores heterogeneos; agendamento transparente; sistemas embebidos; qualidade de serviço; big.LITTLE

Contents

List of Figures

List of Tables

List of Acronyms

DVFS	Dynamic Voltage and Frequency Scaling
GPU	Graphics Processing Unit
QoS	Quality of Service
CPU	Central Processing Unit
SoC	System on a Chip
RISC	Reduced Instruction Set Computer
OS	Operating System
HMP	Heterogeneous Multi Processing
API	Application Programming Interface
ASTPI	Average Stall Time per Instruction
WED	Weighted Euclidean Distances
EDP	Energy Delay Product
PID	Proportional-Integral-Derivative
TDP	Thermal Design Power
FPGA	Field-Programmable Gate Array
CFS	Completely Fair Scheduler
IC	Integrated Circuit
ISA	Instruction Set Architecture
RAM	Random Access Memory

1

Introduction

Nowadays demand for high performance but low power processing is growing in several industries. In the last decade, the computer (and processor) market has shifted from desktops, without strict power restrictions, to laptops and even more recently to tablets and smartphones. This tendency for mobility is creating an increasing demand for low-power and high performance devices. Power considerations are also becoming more relevant on desktops, in part due to the density of modern Integrated Circuits (ICs), which creates a power dissipation problem (currently identified as the ”power wall”). To overcome this issue, power reduction techniques are finding their way into every system.

Most smartphones and tablets manufacturers use ARM architectures combined with energy efficient Graphics Processing Units (GPUs) to power their devices. With the increasing development that these industries have witnessed on recent years, ARM architectures have been pushed to keep up, meaning that new devices not only need to provide a greater performance level, but also need to do so while using approximately the same (or ideally less) power, which means increasing their efficiency. These devices have developed to a point of power-efficiency and performance that server manufacturers can now begin to explore the possibility of creating server farms based on such processors [1].

This demand for performance on mobile applications (while keeping tight energy budgets) has forced companies to explore alternatives, combining energy efficient cores with more power hungry but higher performance cores. Previous research had also found promising results on power saving through heterogeneity ([2], [3]), thus validating this idea.

This work will focus on state of the art implementations of Heterogeneous Multi Processing (HMP) architectures, particularly the ARM big.LITTLE and will aim to develop a method for optimizing the scheduling of tasks for achieving a given compromise between power and performance.

While this problem has given rise to the attention of researchers, existing solutions focus mainly on task distribution on homogeneous cores, considering thermal constraints or operational points to assign

tasks ([4], [5]). Nevertheless, some interesting work has been done in the heterogeneous area ([6], [7], [8]), namely the use of thread phases to build performance tables that are used for task assignment [9], or asserting if a migration can provide performance improvements based on stall time [10]. The challenge in heterogeneous systems, compared to homogeneous ones, is the increase in the number of variables to consider when performing task migrations. In homogeneous systems some linear behaviour is expected according to the operational point, but once heterogeneity is introduced, the same operational point on different processors will yield both different performance and power consumption levels, leading to higher or lower thermal dissipation.

The field of task scheduling targeted at heterogeneous System on a Chips (SoCs) is thus quite at its early stages of development and very little research exists in this specific field. Nevertheless interesting examples can be found combining different metrics such as the work done by [11] which uses Quality of Service (QoS), power awareness and core differences to distribute tasks to the appropriate core.

1.1 Motivation

The development of heterogeneous processors paves the way for new task scheduling schemes, which should be aware of micro architecture characteristics. As stated before, most of the work previously done on this field does not take into account heterogeneity and, as such, not only the research on this subject is in high demand as it is also quite interesting to pioneer the application and development of techniques possibly leading to improved devices.

The goal of task scheduling in embedded devices is to optimize energy consumption while maintaining (or improving) the experience to the end user. Previous work as already shown advantages on taking into account heterogeneity, QoS and power awareness, confirming that scheduling techniques can be further improved considering the existing hardware platforms [11].

A new task migration and scheduling strategy, which takes into account both QoS and core differences, is expected to fill the gap that most methods leave, since they try to maximize the performance of a given task blindly. Doing so might not be strictly necessary and/or might imply the usage of an oversized core. For instance, if the objective of a task is to decode a movie for playback it only needs to respect a given frame rate and thus this is a QoS task. On the other hand, if the task aims to do computational intensive work, such as transcoding a video file, it would be desirable to get the best available performance and thus methods such as those using stall time for non-QoS will still be interesting (e.g. [10]). A method that would achieve a trade-off between QoS, performance, resources' management and energy saving is evidently in demand.

The implementation of a new scheduling technique may cause problems such as the need of program adaptation. While this is undesirable, Application Programming Interfaces (APIs) - such as Application Heartbeats [12] - have been develop allowing for the monitoring of any task's QoS, with the addition of only half a dozen lines of code. Since not all programs will use this (or equivalent) framework and, even on the ones that do, it might be insufficient, the scheduling of tasks must also consider other factors such as memory accesses and other data from internal performance counters.

1.2 Objectives

Based on the previous considerations, the following goals shall be set, to be achieved in order, building on top of each other:

1. Analyse the state of the art of current scheduling and load balancing methodologies;
2. To design a Dynamic Voltage and Frequency Scaling (DVFS) controller based on a given metric (QoS and/or performance counter data);
3. To develop metrics to evaluate the most appropriate core for a task considering the whole system;
4. To combine the gathered knowledge in order to achieve a task scheduling technique applicable to a variety of tasks on embedded heterogeneous systems;
5. To evaluate the performance obtained with the new scheduler by measuring throughput and power consumption.

The most adequate metrics to trigger migrations shall be determined during the execution of this work. Image and video processing programs are of particular interest due to their demand for a given QoS.

As a last stage, the migration to mobile GPUs capable of OpenCL may also be analysed as an alternative step to achieve better performance at the same (or less) energy cost.

1.3 Outline

The following chapters will introduce the architecture to be studied and state of the art scheduling techniques (Chapter 2). Target platforms and benchmarking tools will be analysed (Chapter 3). Finally a schedule of the work to be done (Chapter 4) will be devised followed by a brief conclusion (Chapter 5).

2

Embedded heterogeneous platforms

The increasing complexity of mobile devices' software and applications requires high performance processors with a reduction of power consumption. To tackle this problem, processor architects are turning to heterogeneous multi-core designs.

ARM develops not only the Instruction Set Architecture (ISA) used on most mobile devices (currently, ARMv7-A) but also proprietary cores (e.g., the ARM Cortex-A series). It comes with no surprise that most chip makers choose to implement ARM IP cores on their systems. ARM has developed the big.LITTLE heterogeneous platform [13], a technology that combines cores with different characteristics but a common instruction set.

Other manufacturers, such as Nvidia and Qualcomm, have developed their own heterogeneous SoCs. Nvidia has its Tegra family [14] (combining Cortex-A15 cores with a power efficient one) and Quallcomm has the new Snapdragon 800 series harvesting the power of heterogeneity [15].

With ARM being the reference for most of today's mobile systems, the focus of the research herein presented will be the ARM big.LITTLE embedded heterogeneous platforms, although the techniques developed could be applied to other technologies. Taking this into consideration, the remaining of this chapter is organized as follows: first a brief explanation of what is big.LITTLE will be given, followed by an overlook at the ARMv7-A ISA and the cores used on big.LITTLE. The focus will then be shifted to state of the art scheduling techniques for heterogeneous systems, and to the Linux scheduler, whose knowledge is required to develop the work proposed in this thesis.

2.1 ARM big.LITTLE Architecture

big.LITTLE is a technology developed by ARM combining in the same SoC processors designed for different power and performance budgets, thus allowing for a more efficient system [13].

In current big.LITTLE implementations, Cortex-A15 (higher performance) cores are combined with Cortex-A7 (energy efficient) cores which use the ARMv7-A ISA. Tasks can be assigned to these cores dynamically, according to their needs, with the potential to allow the system to: save energy by shifting simple tasks to the energy efficient cores; while maintaining performance of more demanding tasks by running them on the higher performance cores. Since all cores share the same ISA, existing programs can run natively in either cores and be transparently migrated.

Each cluster, which is a group of similar cores who share the same L2 cache, is composed by different core types (either A7 or A15). The merging of these two clusters of Central Processing Units (CPUs) is possible through the use of the CoreLink CCI-400 Cache Coherent Interconnect, and the GIC-400 Interrupt Control to distribute interrupts across the clusters. ARM has developed other cache coherent interconnects (CCN-504 and CCN-508) that can connect 4 and 8 clusters, respectively, and open the door to the creation of other types of heterogeneous systems with growing complexity and capacities. One particularity of these interconnects is their ability to connect hardware accelerators and media processors, such as the Mali-T600 series GPU, to all the CPUs. This not only preserves processor symmetry but also presents another interesting possibility for optimizations since most of current GPUs support OpenCL (allowing for general computations to be run on GPUs). A diagram of a possible implementation of a big.LITTLE system, featuring interconnects, (64-bit) CPUs and GPU is depicted in Figure 2.1 .

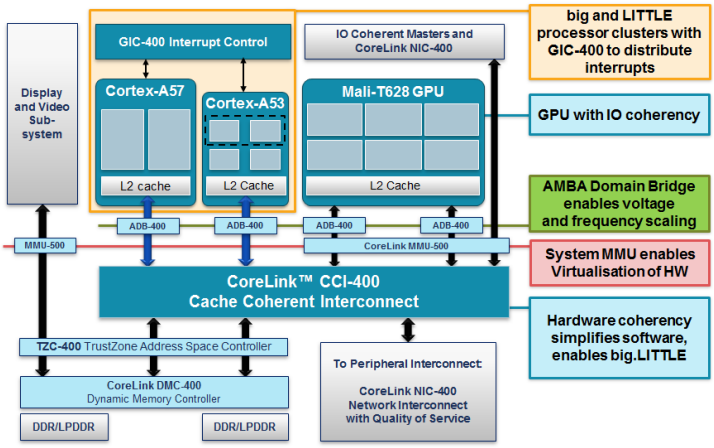

Figure 2.1: Diagram of a possible big.LITTLE system using the CCI-400 and 64-bit cores (Cortex-A57 and Cortex-A53) [16]

2.1.1 ARMv7-A Instruction Set Architecture (ISA)

One of the main characteristics of the ARMv7-A ISA (used on the Cortex-A7 and Cortex-A15) is that it can be implemented on a wide range of microarchitectures with different performance and efficiency levels allowing diversified types of CPUs to be developed. From the application point of view the ARMv7-A ISA implements 13 general-purpose 32-bit registers and 3 extra 32-bit registers providing special features [17, p. A2-50]. It uses a Reduced Instruction Set Computer (RISC) architecture enriched with the ability to combine arithmetic or logical operations with shifts on the same instruction and auto-increment (or decrement) addressing modes. The latter allows optimizing program loops to load (or

store) multiple consecutive memory addresses to the registers (or vice-versa) in a single instruction. This architecture further allows the conditional execution of several instructions to maximize data throughput. The by-product of all these factors is the balance of high performance, small program size, low power consumption and small die area which ARM processors can achieve.

Although the architecture to be used in this work will be ARMv7, which provides 32-bit instructions, a new architecture ARMv8 has now been created allowing for 64-bit instructions and paving the way for 64-bit computing on mobile and low power processors [18].

2.1.2 Cortex-A15 and Cortex-A7 Microarchitecture

It is interesting to make a brief analyses on the specific micro-architecture of both the Cortex-A7 and Cortex-A15 processors as these will be the ones used throughout this work. Both implement the full ARMv7-A ISA including Virtualization and Large Physical Address Extensions as well as NEON instruction set extensions [13], allowing for a seamless migration of tasks between processors apart from the performance difference.

The distinctions between cores are nevertheless clear at a micro-architecture level. The Cortex-A7 is an in-order, dual-issue processor with a pipeline structure of 8 to 10 stages as represented on Figure 2.2. On the other hand the Cortex-A15 is an out-of-order, triple-issue processor, with 15 to 24 pipeline stages (see Figure 2.3).

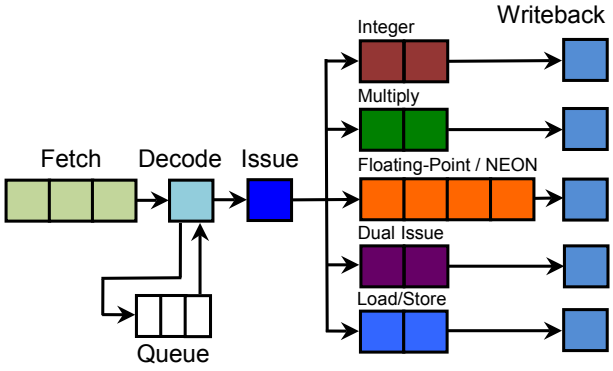

Figure 2.2: Pipeline diagram of the Cortex-A7 [13]

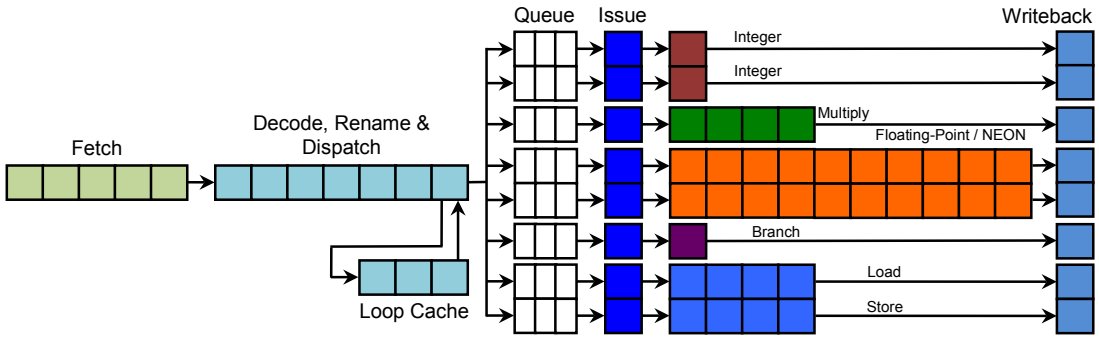

Figure 2.3: Pipeline diagram of the Cortex-A15 [13]

Since the energy spent in the execution of an instruction is in part related to the number of pipeline stages it must go through, a difference on this number significantly affects the energy consumption of

the core. This fact, allied with the increase in complexity (and consequently power) needed for the extra capabilities of the A15, makes the A7 a more power efficient core.

2.1.3 Performance Monitor Unit - Counters

Both A7 and A15 cores include performance counters as part of their Performance Monitor Unit [19, p. 18-5]. Both have a cycle counter in addition to six (in case of the A15) or four (A7) other counters, capable of counting events that occur in the processor. The absolute values recorded may present variations due to pipeline effects but these should be negligible, unless the counters are enabled only for a very short amount of instructions.

To trigger and access performance events the programmer must first configure the Performance Monitor Unit in order to assign an event to each of these counters, such as branch conditions, mispredicted branches, cache accesses, refills and write-backs and memory accesses. The standard events which are countable in all ARMv7-A processors are listed on Table A.1 of Appendix A. While there are other processor specific events that may represent a significant addition to the events listed there, their availability in each of the cores must be checked independently.

These counters can be used either with an external debugger or can be accessed by the processor itself in order to benchmark and profile the execution of tasks.

2.2 Task Migration Techniques

Two main methods are used for task migration in big.LITTLE systems: big.LITTLE Task Migration and big.LITTLE MP [13], as described bellow.

Some important common factors for both migration schemes are related to the handling of the migration. Due to the CCI-400 interconnect, memory coherency is guaranteed by the hardware without the need of software managing (i.e., the need for the CPU to copy the contents of the cache to the main memory so that they can be used on other components from there, such as a different cluster or GPU). The software will nevertheless have to handle the supporting mechanisms for the migration. Examples of these are the state save-restore tasks, bringing the processors in an out of coherency, control snooping in the interconnect and migrate interrupts (via the GIC-400 Interrupt Control).

2.2.1 big.LITTLE Task Migration Model

The big.LITTLE Task Migration Model represents the simplest way to migrate tasks on big.LITTLE platforms, and can be viewed as a natural extension to Dynamic Voltage and Frequency Scaling (DVFS) (that is the adjustment of the operational point of the processor).

Several Operating Systems (OSs) are already energy aware and implement DVFS in order to optimize their energy budgets by changing the CPU operating point (frequency and voltage) to a lower (or higher) one according to tasks' demands. For example, big.LITTLE can take advantage of this method by running less demanding tasks on the A7 cluster with lower voltage and frequency. When the A7 cluster reaches its maximum performance point, the task migration model migrates all the tasks on the

A7 cluster to the A15. DVFS could then continue to be applied with new (higher) performance points calibrated for this cluster.

Task migration requires several steps to be taken on the inbound and outbound clusters (or CPUs) to ensure that proper coherence is maintained. Such steps, illustrated in Figure 2.4, include the powering on and reset of the inbound processor, the transference of state, snooping enabling and finally the power down of the now inactive CPU (or cluster).

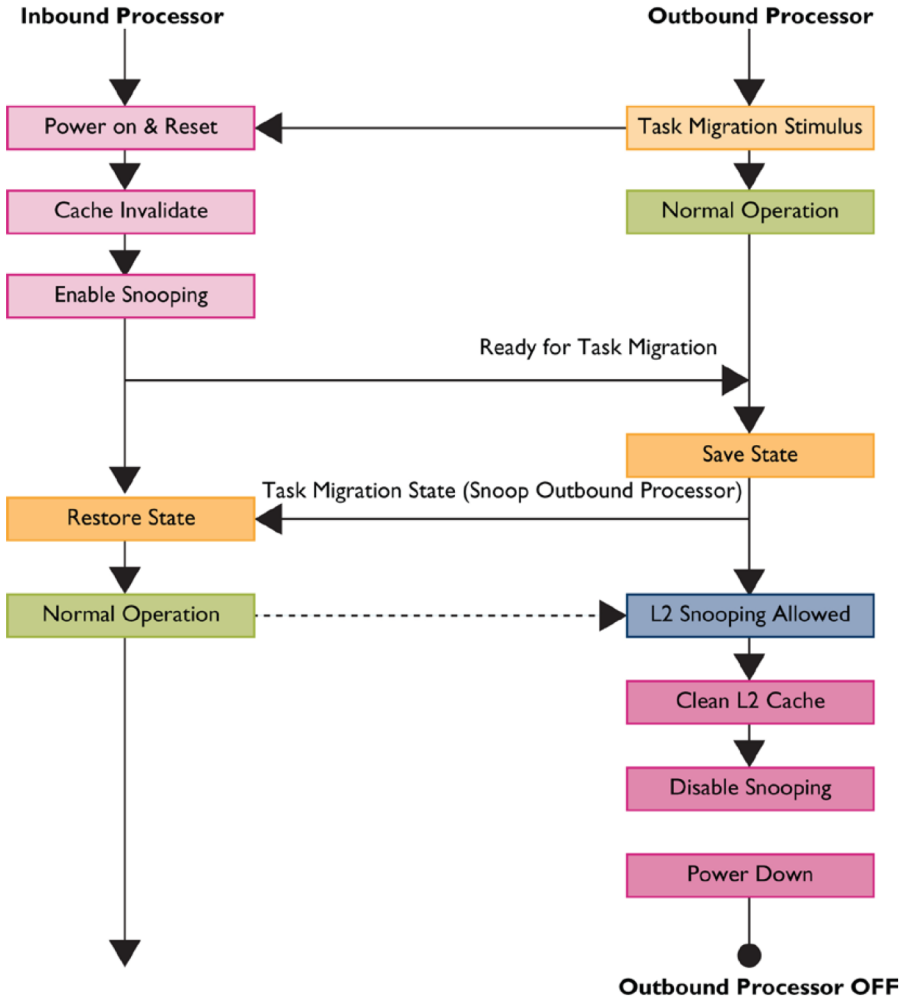

Figure 2.4: Steps to be taken while performing a task migration on the Task Migration Model [13]

One important factor to consider is the task migration cost (in time) and thus some additional ones should be deemed (i.e. don't allow a new switch in less than a given time interval). ARM reports [13] the system is designed to migrate in less than 20,000 cycles (or 20 µs at 1 GHz) while [11] reports times as high as 3.83 ms (frequency dependent). Another relevant point is that instead of migrating the tasks from one whole cluster to another, migration might be done between CPUs only, keeping both clusters running at the same time. As in cluster migration, the total available cores would not duplicate with only one of the cores (either A7 or A15) being active at a time.

2.2.2 big.LITTLE MP

Due to the use of the CCI-400 Cache Coherent Interface and the GIC-400 Interrupt Control all A7 and A15 cores can be simultaneously powered on and used to run applications. The big.LITTLE MP topology, which is the essence of HMP, allows for a better division of tasks between these cores. Since the A15s consume a high amount of power, they should only become active when one (or more) applications require a significant amount of computational power, while the A7s can keep on working on less demanding tasks. For instance, user interactive tasks (graphics related, for example) can be run on the A15s to ensure a smooth experience while background tasks (like email and notification checking) can be done in the A7s considering that a small delay is acceptable.

Implementing the big.LITTLE MP topology presents a greater challenge in comparison to both the previously presented Task Migration Model and homogeneous platforms, because the right core must be found for each task. This attribution is typically done by the scheduler that has to be aware of the core's topology and characteristics. Note that in the previously presented migration scheme a DVFS performance point would map directly to a given type of core for a task. Here, each cluster will have its own DVFS management and tasks must be migrated accordingly.

2.3 Scheduling techniques

Several task scheduling techniques have already been studied in the literature, both targeting homogeneous processors (to achieve thermal improvements through task distribution, for instance) and heterogeneous ones. Some of these techniques, considered to be relevant to the established objective, will be described briefly. Analysing them will reveal important background on state of the art scheduling techniques, building pillars where new schemes can settle on.

2.3.1 Processor demand scheduling

In this section some of the current techniques making use of metrics and measures obtained from the processor (as opposed to application feedback) are reviewed.

L. Sawalha et al. [9] proposed a method to assign tasks to cores according to the corresponding application phase. This method is aware of core heterogeneity and optimizes the task distribution according to which core is the most appropriate for a given phase of the current task. The identification of a phase occurs the first time it appears, and it is then used to predict the performance of the program when it resurges further on during execution. A phase can be determined by periodically monitoring an application execution, e.g. by the number of instructions per cycle. The first time a phase occurs, it is run across all available cores in order to evaluate the performance on each one. Having built a Signature History Table for the known phases in a thread, scheduling the tasks is accomplished by looking at said tables and choosing the best combination of performance levels for all the running phases (assumed known) of current threads. This technique requires tuning some parameters, namely the sample window size and does not consider DVFS.

P. Nie and Z. Duan [10] suggest another scheduling technique for heterogeneous systems, by analysing the Average Stall Time per Instruction (ASTPI), which measures the average time an application waits for memory contents. Since an application is monitored periodically it also allows for the identification of different program phases. For this, the memory hierarchy of each core is assumed to be the same meaning the stall time should be similar independently of the type of core. A small ASTPI is then expected to translate in a greater ratio of arithmetic over memory operations and thus the more benefits a task can get from migrating to a fast core. It was verified by the authors that the ASTPI was in fact constant across core types[1] and that the speed up factor of a thread is monotonically decreasing as ASTPI grows. One clear advantage of this method is the lack of the need to test a task across all types of cores.

A different approach was adopted by P. McKenney et al. [20], where the impact of non-performance-critical work invoked by performance-critical code is reduced. In this specific case the authors addressed the read-copy update mechanism which is used on the Linux Kernel (and now also in user space) to read-write lock data (several threads can read the same data but only one can write it). Two methods to achieve power reduction on this operation are considered: one option is to reduce the number of scheduling-clock interrupts appearing on an otherwise idle processor while a wait-for-readers operation is pending (replacing it with sparser, timed checks); another option is to offload the referred operations to a lower performance CPU. The combination of these two methods does not yield better results in terms of energy performance since both methods aim to reduce the same busy period on a high performance CPU, and can actually lead to a degradation of performance due to the additional overhead. These methods equally reduced power consumption but enforcing a larger idle period had a more significant (although low) impact on performance due to the increase on the grace-period.

Another interesting contribution was made by J. Chen and L. John [21] who suggested to use the euclidean distance between a core's capabilities and the program's resource needs as a metric to guide the scheduling of tasks. In this method, the programs are first profiled to obtain a given number of relevant metrics. These metrics are then projected onto a unified multi-dimensional space where metrics coming from the processors configuration are also projected. The metrics on the processor side can be: the issue width, branch predictor and L1 data cache size, for instance; while on the program side: instruction level parallelism, branch predictability, and data locality can be used. Then it is a matter of measuring the distance between task and core in the space where the projection occurs (where different dimensions of the vectors can have different weights) and attributing tasks to the closer processor. This method, which was called Weighted Euclidean Distances (WED), proved to have a good correlation with Energy Delay Product (EDP) with greater distances yielding a worst match between core and task.

An alternative method based on offline profiling of tasks could be implemented by sampling all possible voltage and frequency combinations for different programs with a given input set in order to profile their execution. This method would allow not only to pick the right level of DVFS for a task but also to consider task migration across cores in order to join tasks with similar requirements on the same clusters. The downside of this approach is the cost of the offline profiling and the fact that the input set used

[1]The test was run on both an Intel and AMD multicore machine with different core types simulated by setting different frequencies and enabling/disabling cores to create differentiated memory organization schemes.

may not reflect the behaviour of an online system.

2.3.2 Quality of Service Aware Scheduling

A more interesting metric for migration is the QoS of a task and not its total execution time, specially for tasks interacting with the user. Tasks such as music or video playback require a minimum processing rate to be respected, but a maximum one can also be set.

An important question on QoS awareness is how to measure it. Hoffmann et al. [12] proposed an API to measure QoS called Application Heartbeats. This API was developed with the intent of improving task efficiency by adjusting resources in order to strictly fulfil the task's necessities. A drawback of this approach is that it requires changes in the application, although these changes can be made in less that half a dozen lines of code. The application can then set a minimum and maximum heartbeat rate (or a latency, if that is more adequate) which can be analysed internally or externally to adjust either the program or the resources attributed to it.

T. Muthukaruppan et al. [11] designed an integrated scheduling solution which can control several parameters of the processor with the help of Application Heartbeats for QoS tasks. The approach taken was directly applied to big.LITTLE and thus proves the applicability of most of the discussed concepts to the target platform of this thesis. Due to the fitting of this research in the scope of this thesis a more in depth analyses of it will be made herein.

The technique proposed in [11] consists on building a Proportional-Integral-Derivative (PID) feedback control loop which analyses and adjusts several parameters, such as tuning QoS targets, adjusting DVFS to a given cluster and migrating tasks when the core where they are executing does not deliver an appropriate performance level. An overview of the control loop that implements this per-task resource share controller, per-cluster DVFS controller and per-task QoS controller is depicted in Figure 2.5. This diagram shows the control flow of the whole system, with a chip-level power allocator specifying a throttling level for QoS tasks and CPU utilization factor, a block (top blue square) responsible for applying, if necessary, the specified throttling, a resource share controller and balancer block (bottom left) and a cluster DVFS controller (bottom right). The migration is activated if a task cannot fill its specified QoS while running at the maximum capacity on the A7 core.

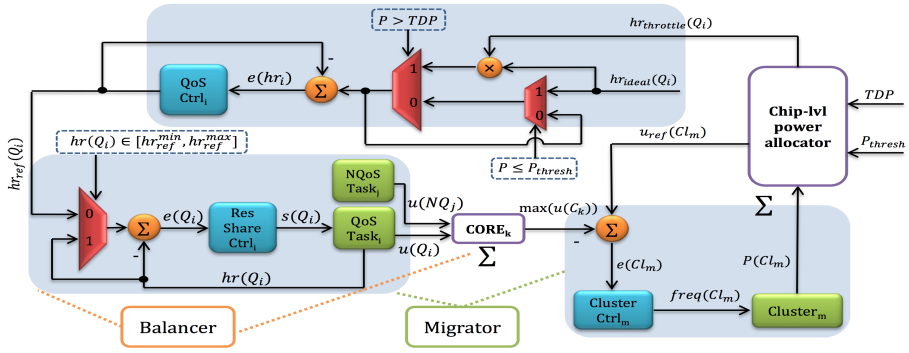

Figure 2.5: Control loop used on T. Muthukaruppan's work composed by a chip-level power allocator, a per-task QoS controller, a load balancer and migrator and a per cluster DVFS controller [11]

A central focus was given to the Thermal Design Power (TDP), responsible for setting a limit to the power a chip can consume in order to respect thermal requirements. If the power consumption is above the limit, a throttle to QoS is set along with a request for higher CPU utilization (resulting on a drop of frequency). These controllers are called at different intervals in order not to cause any performance drop, by flooding the CPU with scheduling tasks, and because some effects (like thermal response) have some inherit latency. The per-task resource share controller and load balancer (bottom left of Figure 2.5) are thus invoked at a shorter interval while the per-cluster DVFS (bottom rigth) and per-task QoS controller (top) are called in larger intervals. The chip power allocator (absolute right) is called less frequently since the thermal response is the slowest of all.

While the work proposed by T. Muthukaruppan et al. [11] achieved significant energy savings, it still leaves room for improvement, since non-QoS tasks are simply left with the performance that remains from the QoS tasks.

Although it is a deviation from the main focus of this work (which is the CPU), QoS could also yield energy improvements in GPUs. Since frames on a GPU are expected to be ready in a given interval of time, dropping ones whose processing is not done on time, it is possible to have a high GPU utilization with a relatively low frame rate. If a maximum frame rate is set, the GPU will not try to maximize its performance to get above it (thus not working constantly at full power). By setting an appropriate performance goal the user would not feel a performance drop and power which could be otherwise wasted (specially if several frames are being dropped) is saved.

2.4 Looking into the scheduler

The Linux Process Scheduler is the portion of the Linux Kernel responsible for managing a system's tasks workload distribution. Being an essential component of a system it affects a diverse number of metrics, namely the throughput, latency and fairness. As such, most of the techniques presented in the previous sections require alterations of it to be implemented. Although such modifications are diverse, most of them can be seen as an extension to the existing scheduler, making it important to give a brief explanation on how the typical scheduler of a current Linux system works. The scheduler has thousands of lines of code causing a full description of its inner workings and subtleties to be impractical. Instead a brief description of the standard scheduler will be given.

As in other operating systems, scheduling policies in Linux evolved (and are evolving) gradually with alternatives and optimizations appearing constantly [22]. Mainline Linux distributions started to incorporate the idea of scheduling classes in version 2.2 allowing for different policies to be taken for real-time tasks, non-real-time tasks, and non-preemptible tasks.

Version 2.4 of the Kernel implemented an O(N)[2] schedule that assigned time slices to tasks allowing them to execute up to a maximum (fixed) time interval. Version 2.6 introduced a new scheduler (called the O(1) Scheduler), solving some issues of the previous scheduler. One of the main differences was

[2]The O(•) notation means that an algorithm can execute on a time which is proportional to •. O(1) means the the algorithm can be done in constant time (independent of the number of inputs), O(N) linear time (N being the number of inputs), O(log(N)) a time proportional to the logarithm of the number of inputs and so on. In the scheduler's case, the inputs are the tasks to schedule.

that it did not have to iterate over all the tasks (thus the O(1)). The downside however, was the complexity that its implementation required and thus the Completely Fair Scheduler (CFS), which was built based on another promising schedule at the time, was introduced.

The CFS main objective is to maintain fairness, that is, balancing the available CPU time among tasks equally. This scheduler also includes a concept of sleeper fairness to ensure that sleeping tasks (like tasks waiting for I/O) get a fair share of the processor when they actually require it. A key aspect of the CFS is that it maintains tasks in a time-ordered red-black tree (Figure 2.6). This tree has the properties of being self balancing (a path in the tree is never longer than twice the distance of any other) and that operations (such as reinsertions on the tree) can be done in O(log(N)), allowing for fast insertions and deletions from the tree. Note that the O(1) scheduler has a better performance than the CFS, but due to the simplified structure of the latter, the choice to drop the O(1) in favour of the CFS was made.

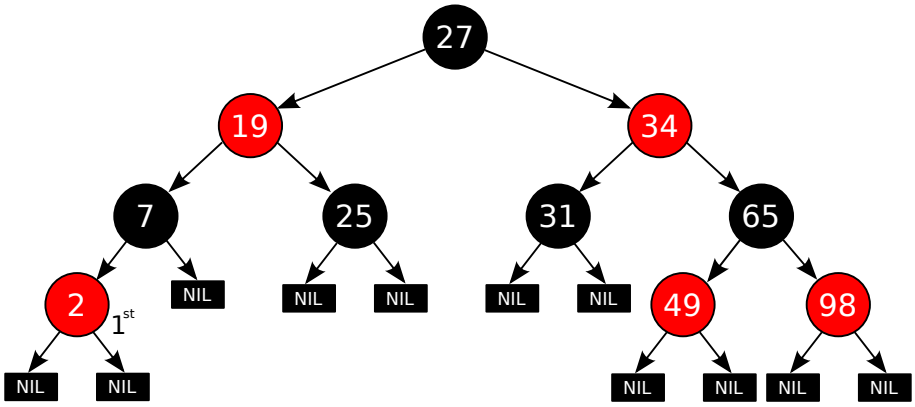

Figure 2.6: Red-black tree example, with the node with the highest priority marked as first

As depicted on Figure 2.6, the scheduler picks the node on the left-most side of the time-ordered tree (marked as 1^{st}) which will be the one with lower runtime and then, after the task it represents is running, the scheduler will update the runtime of the task and reinsert it into the tree. This creates a migration from right to left in the tree which allows the scheduler to maintain fairness.

CFS handles the specification of a task priority by scaling down the time a task is allowed to execute when its priority is dropped (or scaling up if it is increased). It further ensures fairness thanks to group scheduling. Group scheduling is particularly useful in tasks that spawn many other tasks, such as a server handling several input connections, and attempts to provide a fair behaviour in each group. For instance, using the previous example, the server task will be allocated a given slice of usage (divided among the other tasks in the system) and its sub-tasks would receive an equal fraction of that time. This way fairness is maintained in several levels (groups).

There is also the possibility of having different scheduler classes on the same system and allowing tasks to be attributed to the range of the referred classes. This can be useful to change the scheduling scheme of tasks that require real time processing. Another important concept is scheduling domains, creating the possibility of grouping processors hierarchically to make a better load balance and segregation of tasks between them. Since processors can share scheduling policies this possibility turns out to have a significant value on multiprocessor systems, particularly on the case of HMP.

2.5 Summary

In this section the ARM architecture that will be the target of this work was presented, and a collection of proposed migration schemes is discussed. The architecture is designed for low power operation while providing performance capable of keeping up with today's demand. big.LITTLE further develops this idea by combining power efficient cores with performance optimized ones on the same SoC.

As for task migration and scheduling, several topologies were studied. Some were based on processor dictated metrics, such as memory access times, CPU utilization, and offline studies of the program behaviour so that performance tables can be built. Other methods used QoS to decide which resources should be assigned to each task. These methods have the potential to make significant power savings without the user experiencing a degrade on performance. In any case, after gathering the necessary metrics, a scheduling scheme can be devised.

A brief introduction to the scheduler (and CFS in particular) was also made. Familiarity with this component of the Linux Kernel will be of high importance during the course of this work. This component assigns and schedules tasks, making it the most appropriate place to introduce changes in the scheduling and power control policy.

3

Preliminary big.LITTLE evaluation

In the previous chapter a general overview of existing scheduling methods and topologies was presented. This chapter will summarize the state of the art on development platforms powered by ARM. Benchmark suites capable of evaluating the system's performance shall also be introduced followed by some preliminary experiments.

3.1 Target platforms

Development platforms are an essential tool to develop and analyse new scheduling policies since they provide the possibility of implementing and testing these policies on real hardware and to experimentally evaluate how performance is affected. There is a variety of platforms powered by SoCs with ARM's Cortex-A processors (some of them with big.LITTLE). An important factor to consider when the platform is chosen is the possibility of running Linux (preferably Linaro) on these boards, but the ones presented here are capable of doing so and as such that will not represent a problem.

A Versatile Express

Versatile Express is a development platform built by ARM. It is the most complete development solution, allowing for a wide range of configuration options, from processor type to additional logic boards with Field-Programmable Gate Arrays (FPGAs) that can provide additional (custom) logic to the system.

The interest herein lies in a specific processor option: the ARM CoreTile Express for Cortex-A15, Cortex-A7 which implements two clusters, one with two Cortex-A15 cores and another one with three Cortex-A7 cores, supporting all task migration modes available on big.LITTLE. It also includes 2 GB of DDR2 RAM.

Due to its characteristics, the Versatile Express is the most suitable platform for development with

big.LITTLE, since all the capabilities of this platform are available. Current measuring sensors are installed on this board, allowing the calculation of the power delivered to each cluster. These can be used to directly measure the impact of new schedulers. Per cluster DVFS is also available.

B Odroid-XU-E

The Odroid-XU-E is a small development board equipped with a Samsung Exynos 5 Octa core processor (4x A7 and 4x A15) model 5410, 2 GB of LPDDR3 RAM and a PowerVR SGX544MP3 GPU (Capable of OpenCL 1.1).

This board's CPU has an implementation of big.LITTLE where only cluster migration works, leaving out both core migration and big.LITTLE MP (were all cores are available). This limits the work that can be developed on this board, but it still proves to be useful for some tasks, namely to analyse the individual performance of the A7 and A15 chips, to study migration techniques based on DVFS alone (switching clusters when a given threshold is reached), and also offers the possibility of using the GPU which has advanced features like the OpenCL compatibility. Per cluster DVFS and current sensors (for each CPU cluster, GPU and memory) are available.

C Arndale Octa Board

The most recent development board herein presented is the Arndale Octa Board which has the current Samsung Exynos 5 Octa core processor, model 5420, also with four cores of each type (A7 and A15). It includes on the same chip the Mali-T628 MP6 GPU as well as 2 GB of LPDDR3e RAM.

The advantage of this board over the Odroid-XU-E lies in the upgraded version of the multicore, solving some issues that were identified in model 5410 and opening the door for the big.LITTLE MP task migration scheme. Since information about the implementation of big.LITTLE on this chip is still sparse, probably due to its recent introduction on the market, further research will have to be conducted to verify whether or not this platform will qualify for the pretended work. This will be done at a later stage, but indications point to the real possibility to explore all big.LITTLE migration topologies.

D Other development boards

During the search that has been done in the scope of this thesis that led to the selection of the previously mentioned boards, other ARM development platforms were found but they were discarded since the above ones were the only ones that provide the big.LITTLE resources and features.

Some of the other boards that were considered were the ZedBoard (using the Zynq SoC with a dual Cortex-A9 implementation), the Odroid-U and Origen Board (powered by a Samsung Exynos 4 quad core Cortex-A9), the PandaBoard (with a dual core Cortex-A9 implemented by Texas Instruments), the YSE5250 and Arndale Board (both with a Samsung Exynos 5 dual core Cortex-A15) and the Beagle-Board (powered by a Texas Instruments Cortex-A8). All of these boards use ARM's ARMv7-A architecture, but since they don't implement big.LITTLE they weren't considered as potential boards to be used in this thesis.

A Summary of the main characteristics of the Versatile Express, Odroid-XU-E and Arndale Octa Board is presented in Table 3.1. Some of the details herein presented (such as Maximum frequencies and migration modes available) are subject to further research and will only be confirmed during the development of the work.

Table 3.1: Main characteristics of the development platforms considered

	Versatile Express	Odroid-XU-E	Arndale Octa Board
CPUs	3 x Cortex-A7; 2 x Cortex-A15	4 x Cortex-A7; 4 x Cortex-A15	4 x Cortex-A7; 4 x Cortex-A15
RAM	2 GB DDR2	2 GB of LPDDR3	2 GB of LPDDR3e
Maximum Frequency	1 GHz	1.6 GHz	1.8 GHz
big.LITTLE modes	Cluster/Core Migration; MP	Cluster Migration	Cluster/Core Migration; MP
Available Accelerators	Optional FPGA Logic Core	PowerVR SGX544MP3 GPU	ARM Mali-T628 MP6 GPU
DVFS	Per cluster	Per cluster	Per cluster

3.2 Experimental benchmarking

A critical step during the course of this thesis is to evaluate the performance of the scheduling solutions and experimentally assessing their weaknesses and strengths. Benchmarks are the tool for this job since they offer repeatable tests whose results will differ only due to system behaviour (and task load at the time, which should be limited as much as possible to the benchmark itself for reliable testing).

3.2.1 Benchmarks

Several benchmark test suites are available targeting different computing environments and platforms. From these, some are used to test mobile embedded system platforms, which take special relevance in the scope of this work and have been used in the state of the art. As an example, in [11] both The San Diego Vision Benchmark Suite [23] and PARSEC [24] are used. Another available suite is the Phoronix Test Suite [25]. This suite is popular among users and developers providing a wide variety of tests for various components of a system (e.g., CPU, GPU, hard drive and memory specific tests, capable of running on a range of systems from desktops to mobile devices).

One important aspect for choosing the most suitable benchmark suites is the availability of the source code of the suites. QoS metrics may require alterations to this code, but it should not pose a problem since the source code is available for the majority, if not all, of the tests. Due to the variety and real world tests run by the Phoronix Test Suite (making use of existing programs and libraries) changes on the source code of these may become a hard task, but PARSEC and VIsion (dedicated to research purposes) make this task easier and thus these shall be chosen when source code alterations are needed.

A brief presentation of the mentioned benchmarks and their characteristics are discussed on the following text. These will be considered when running the tests according to how they adjust themselves

to what is needed (e.g., if QoS is required in a test, a test suite whose code is more accessible will be chosen).

A The San Diego Vision Benchmark Suite

The San Diego Vision Benchmark Suite [23] provides a set of vision oriented applications, which often provide a high level of data parallelism, making them ideal to test multicore systems. The authors provide the benchmark both in MATLAB and C programming languages. The C version was chosen since it makes the benchmark runnable in a wider variety of systems and it is easy to introduce changes to the tests (e.g., those required to assess QoS). Input sets for these applications are also available. A brief description of the 9 benchmark programs that are included on this suite is made on Table 3.2.

Table 3.2: Available tests on the The San Diego Vision benchmark suite

Test name	Function
Disparity Map	Motion, Tracking and Stereo Vision
Feature Tracking	Motion, Tracking and Stereo Vision
Image Segmentation	Image Analysis
Scale Invariant Feature Transform (SIFT)	Image Analysis
Maximally Stable Regions (MSER)	Image Analysis
Robot Localization	Image Understanding
Support Vector Machines (SVM)	Image Understanding
Image Stitch	Image Processing and Formation
Texture Synthesis	Image Processing and Formation

B PARSEC

Christian Bienia [24] created the Princeton Application Repository for Shared-Memory Computers (PARSEC) benchmark suite. Its key features are the focus on multithreaded workloads, diversity (exploring several application domains) and non high-performance computing focus. The workload includes applications ranging from image and video processing to content similarity search servers and partial differential equation handling programs. The source code is also available. Table 3.3 briefly presents the 13 available tests on this suite.

C Phoronix Test Suite

The Phoronix Test Suite [25], developed by Phoronix, is an open source testing platform with a large community support and a variety of available tests. It can be used to evaluate several system components with dozens of available tests for each. A large number of these tests is directed at the processor, providing a series of real world tasks (based on programs typically found on computer systems, such as the 7-Zip Compression). Although not intended for research proposes, it may also prove useful on this field. The downside of this suite lies on its simplicity: the fact that its tests are mainly focused on existing programs (running specific and predetermined workloads) makes it harder to change the source code if needed. Due to the variety and quantity of tests this suite can run (more than 70 processor testing profiles) a summary of these benchmarks is not herein presented.

Table 3.3: Available tests on the PARSEC benchmark suite

Test name	Function
blackscholes	Option pricing with Black-Scholes Partial Differential Equation (PDE)
bodytrack	Body tracking of a person
canneal	Simulated cache-aware annealing to optimize routing cost of a chip design
dedup	Next-generation compression with data deduplication
facesim	Simulates the motions of a human face
ferret	Content similarity search server
fluidanimate	Fluid dynamics for animation purposes with Smoothed Particle Hydrodynamics (SPH) method
freqmine	Frequent itemset mining
raytrace	Real-time raytracing
streamcluster	Online clustering of an input stream
swaptions	Pricing of a portfolio of swaptions
vips	Image processing (Project Website)
x264	H.264 video encoding (Project Website)

3.2.2 Cortex-A7 and Cortex-A15 performance

The Odroid-XU-E development board was used to take some preliminary performance readings from the Cortex-A7 and Cortex-A15 in order to experience the differences in the cores available on this board. Due to the straightforwardness of this tests, the benchmark suite used was the Phoronix Test Suite.

The Odroid can operate at the following frequencies, ranging from 250 MHz to 1.8 GHz: 250 MHz, 300 MHz, 350 MHz, 400 MHz, 450 MHz, 500 MHz, 550 MHz, 600 MHz, 800 MHz, 900 MHz, 1.00 GHz, 1.10 GHz, 1.20 GHz, 1.30 GHz, 1.40 GHz, 1.50 GHz, 1.60 GHz, 1.70 GHz and 1.80 GHz. The transition between the A7 and the A15 occurs at the 600 - 800 MHz boundary. Thus at operating frequencies ranging from 800 MHz to 1.8 GHz the applications are run on the A15 core and at lower frequencies on the A7.

The test procedure can be described with the following steps, with 4 cores available at a time: 1) Fixing the clock speed at 600 MHz (forcing the use of only the A7s); 2) Running the benchmarks while acquiring power data; 3) Fixing the clock speed at 800 MHz (forcing the use of only the A15s) 4) Running the benchmarks while acquiring power data; 5) After gathering the results, and since both frequencies and voltages changed with the cores, the benchmarks results were scaled by $\frac{1}{F}$ and, since $P \approx F \times V^{\,2} \times \alpha$, the power readings by $\frac{1}{F \times V^2}$ to get comparable values. The previous expression accounts for the dynamic power required by the digital circuitry (assumed significantly greater than the static). The α accounts for constant circuit parameters (e.g., capacitances). The power was sampled once every second so that the CPU would not be loaded by the readings.

This combined results show that, at the same performance levels, the A7 cores are much more power efficient than the A15s (Table 3.4 presents the average power and Figures 3.1 and 3.2 show how power is consumed during a run of the test), but when higher performance is needed the A15s should be capable of higher frequencies (and performance) at the cost of power efficiency. On these tests, looking at the normalized results in Table 3.5, there wasn't a clear performance improvement on the A15 cores as might be expected. This may be due to other factors external to the CPU, such as memory accesses, being responsible for a performance bottleneck.

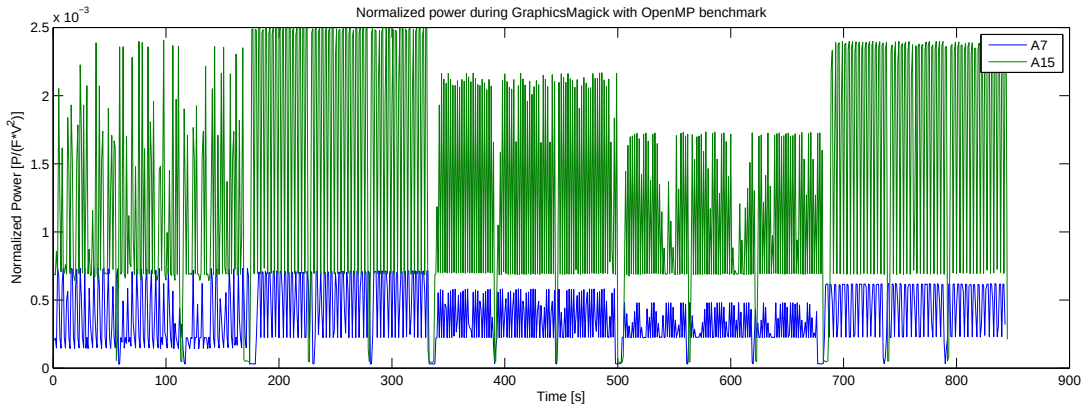

Figure 3.1: Power consumption during the GraphicsMagick benchmark for the A7s and A15s

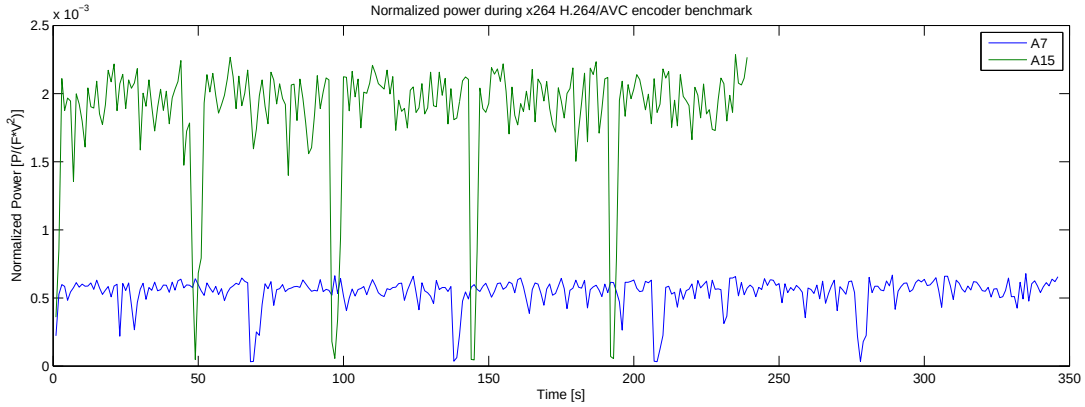

Figure 3.2: Power consumption during the x268 benchmark for the A7s and A15s

Table 3.4: Power: a) Absolute and b) Normalized average results

(a)

Benchmark	A15	A7
GraphicsMagick [W]	0.9195	0.3490
x264 [W]	1.2514	0.4882

(b)

Benchmark	A15	A7	Ratio
GraphicsMagick	1.369	0.389	3.51
x264	1.863	0.544	3.42

Table 3.5: a) Absolute and b) Normalized results

(a)

Benchmark	A15	A7
GraphicsMagick [iter./min]	109	96
x264 [av. fps]	12.02	8.19

(b)

Benchmark	A15	A7
GraphicsMagick	0.14	0.16
x264	0.02	0.01

Although the frequencies chosen for analyses were 600 MHz and 800 MHz, due to their similarity and the fact that they represent the frontier between the Cortex-A7 and A15, the tests were run for other frequencies. Those results are available on Table 3.6 and 3.7. An observation that can be made is that performance scales linearly with frequency independently of the type of the core used (probably due to factors external to the CPU and whose understanding will be required in future tests), but power behaviour is clearly different on different cores.

Table 3.6: Benchmark results for all tested frequencies

	250 MHz A7	450 MHz A7	600 MHz A7	800 MHz A15	1.2 GHz A15	1.6 GHz A15
GraphicsMagick [iter./min]	44	74	96	109	162	210
x264 [av. fps]	3.68	6.39	8.19	12.02	17.98	23.62

Table 3.7: Average power consumption during benchmarking for all tested frequencies

	250 MHz A7	450 MHz A7	600 MHz A7	800 MHz A15	1.2 GHz A15	1.6 GHz A15
GraphicsMagick [W]	0.0849	0.1816	0.3490	0.9195	1.6862	3.0688
x264 [W]	0.1267	0.2629	0.4882	1.2514	2.4577	4.5211
Voltage [V]	0.9488	1.0363	1.2225	0.9163	1.0400	1.1788

3.3 Summary

This section gathers a collection of data about available ARM platforms and benchmark suites that can be used to test such systems. Only the development platforms with big.LITTLE were considered. Some research must still be done on the ones powered by the Exynos 5 Octa SoC to understand up to what point big.LITTLE can be used on them, and which migration options are available. At the time of writing, a new version of this processor (model 5420) was introduced so documentation is sparse and close to non-existing, thus further research must be done. In any case, the Versatile Express would be the ideal testing platform due to its support.

A brief list of benchmark suites was also presented in this chapter. These suites incorporate diverse tests that can be used to evaluate the performance attained with various scheduling and migration techniques on the CPU. Some preliminary tests where also run on the Odroid-XU-E which show the increase of energy consumption when tasks are migrated to the A15.

4

Work proposal

Given the previous research, a work proposal is herein presented, whose objective is to improve scheduling and migration techniques in order to reduce energy consumption while maintaining the required performance for user satisfaction. Due to the complexity of this goal several steps must be taken to gradually achieve it. Such steps are summarized on the following list:

1. Build the state of the art of current methodologies;
2. Control DVFS based on a given metric (QoS and/or performance counter data);
3. Develop metrics to evaluate the most appropriate core for a task considering the whole system;
4. Combine the gathered knowledge in order to achieve a task scheduling technique applicable to a variety of tasks on embedded heterogeneous systems;
5. Evaluate the performance obtained with the new scheduler by measuring throughput and power consumption.

The first approach should consist in analysing the DVFS based migration across clusters which is implemented on the ODROID-XU-E platform. Since this migration technique requires all tasks to be migrated across clusters, improvements in performance or energy consumption are not expected. However, this is a a fundamental step to gather familiarity with simple scheduling techniques.

After the initial task migration step, an important aspect to analyse is how to monitor application performance using performance counters, and other parameters which can help to understand processor usage. These should be available within the scheduler environment so that they can be used to guide the scheduling.

The next step shall be to verify how an API such as Application Heartbeats (that allows for the monitoring of threads with a metric specified by the application programmer) can be used inside the scheduler to guide the task migration and scheduling policies.

The previous two steps can be verified on any platform since they consist on analysing the Linux scheduler. They can then be tested by deciding DVFS levels based both on performance counters and application feedback via QoS APIs. This will also yield relevant results on energy savings and performance.

The final and more challenging step is to combine the previous knowledge in a scheduling technique which achieves significant energy savings for different types of tasks and does not forcefully require changes in the current programs. For this, a scheduling technique like the one presented in [11] shall be built with the inclusion of metrics that can also analyse and throttle non-QoS tasks as well as QoS ones.

Finally, performance of the scheduling method will be evaluated based on power consumption and program throughput or other available QoS metric.

4.1 Work plan

The previously proposed work should be structured according to what is presented on Figure 4.1. The delivery goal is set for July 2014 but, due to the nature of every research project, unforeseeable circumstances may postpone this date further on up to September. Nevertheless, the tasks to be done are expected to be manageable enough to fulfil the plan.

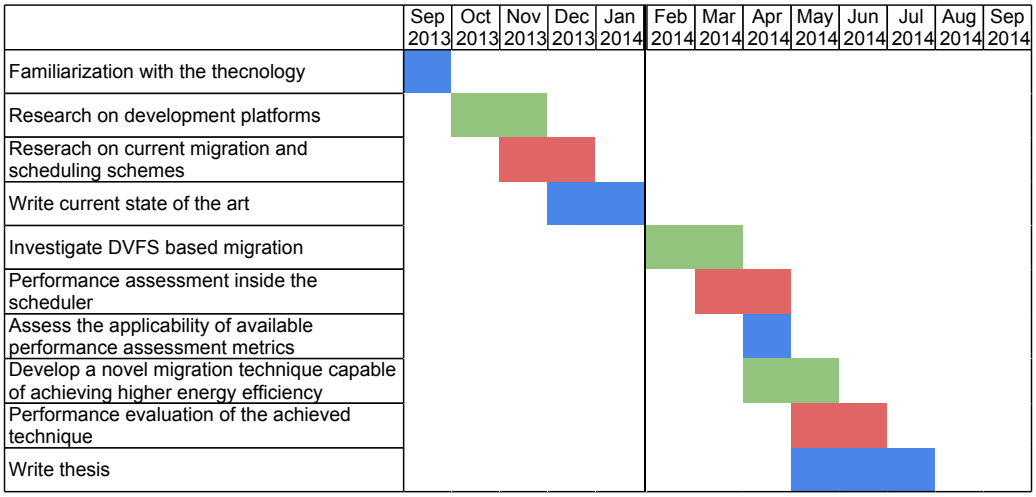

Figure 4.1: Proposed work schedule

5

Conclusions

The work developed so far is mainly focused on analysing the current state of the art of heterogeneous systems, specially the big.LITTLE platform. Since these systems are relatively new in the field of mobile and low power devices, information about them is hard to come by. Nevertheless, some research work has already been done on them, as described in this document. Furthermore, there are several methods to perform task migration reported on the literature, some with the potential to be applied to this new type of systems.

The focus of this work is to merge existing solutions in order to build a new system controller, capable of saving energy in a wide variety of tasks while minimizing the performance drop from the user point of view. Due to the research required for this work, this report does not present preliminary implementations of the scheduling topology to be developed, focusing instead on reporting existing topologies. With the information herein gathered, the development of a scheduling topology may now begin.

This report provides an extremely important starting point for the thesis since it summarizes the most relevant scheduling techniques that can be applied to big.LITTLE systems and how to test them. Based on the material here reported, benchmark test suites can be defined, PARSEC and Vision can be used for tests which require source code alterations and the Phoronix Test Suite may be used to get a wider variety of tests at the cost of limited customization. The work regarding big.LITTLE MP (using all the cores) should be made on the Versatile Express platform or the Arndale Octa Board, leaving the Odroid-XU-E as a platform for cluster migration tests.

It is expected that the work plan has enough slack and can be fulfilled, but it would also be unsurprising that the state of the art had to be altered during the course of the work (since new platforms and techniques are constantly appearing). This could be done in real time while the work is conducted and is not expected to be the source of major delays.

References

[1] Marcello Coppola, Babak Falsafi, John Goodacre, and George Kornaros. From embedded multi-core socs to scale-out processors. In Proceedings of the Conference on Design, Automation and Test in Europe, DATE '13, pages 947–951, San Jose, CA, USA, 2013. EDA Consortium.

[2] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André Wacrenier. Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Concurrency and Computation: Practice and Experience, 23(2):187–198, 2011.

[3] Andrew Lukefahr, Shruti Padmanabha, Reetuparna Das, Faissal M. Sleiman, Ronald Dreslinski, Thomas F. Wenisch, and Scott Mahlke. Composite cores: Pushing heterogeneity into a core. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-45, pages 317–328, Washington, DC, USA, 2012. IEEE Computer Society.

[4] Jeonghwan Choi, Chen-Yong Cher, Hubertus Franke, Henrdrik Hamann, Alan Weger, and Pradip Bose. Thermal-aware task scheduling at the system software level. In Proceedings of the 2007 International Symposium on Low Power Electronics and Design, ISLPED '07, pages 213–218, New York, NY, USA, 2007. ACM.

[5] V. Hanumaiah and S. Vrudhula. Energy-efficient operation of multi-core processors by dvfs, task migration and active cooling. Computers, IEEE Transactions on, PP(99):1–12, 2012.

[6] Stephen Ziemba, Gautam Upadhyaya, and Vijay S Pai. Analyzing the effectiveness of multicore scheduling using performance counters. In Proceedings of Workshop on the Interaction between Operating Systems and Computer Architecture, 2008.

[7] Juan Carlos Saez, Manuel Prieto, Alexandra Fedorova, and Sergey Blagodurov. A comprehensive scheduler for asymmetric multicore systems. In Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, pages 139–152, New York, NY, USA, 2010. ACM.

[8] Tong Li, Dan Baumberger, David A. Koufaty, and Scott Hahn. Efficient operating system scheduling for performance-asymmetric multi-core architectures. In Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC '07, pages 53:1–53:11, New York, NY, USA, 2007. ACM.

[9] Lina Sawalha, Monte P. Tull, and Ronald D. Barnes. Thread scheduling for heterogeneous multicore processors using phase identification. SIGMETRICS Perform. Eval. Rev., 39(3):125–127, December 2011.

[10] Pengcheng Nie and Zhenhua Duan. Efficient and scalable scheduling for performance heterogeneous multicore systems. Journal of Parallel and Distributed Computing, 72(3):353 – 361, 2012.

[11] T.S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. "Hierarchical power management for asymmetric multi-core in dark silicon era". pages 1–9, 2013.

[12] Henry Hoffmann, Jonathan Eastep, Marco D. Santambrogio, Jason E. Miller, and Anant Agarwal. Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments. In Proceedings of the 7th International Conference on Autonomic Computing, ICAC '10, pages 79–88, New York, NY, USA, 2010. ACM.

[13] P. Greenhalgh. "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7". ARM White Paper, 2011.

[14] Tegra processors. http://www.nvidia.com/object/tegra-4-processor.html. Accessed: 06/12/2013.

[15] Snapdragon processor features. http://www.qualcomm.com/snapdragon/features. Accessed: 06/12/2013.

[16] Corelink cci-400 cache coherent interconnect. http://www.arm.com/products/system-ip/interconnect/corelink-cci-400.php, December 2013. Accessed: 12/12/2013.

[17] ARM. ARM Architecture Reference Manual, armv7-a and armv7-r edition, 2012.

[18] ARM. ARM Architecture Reference Manual, armv8, for armv8-a architecture profile, errata markup, beta edition, 2013.

[19] ARM. CortexTM-A Series Programmer's Guide, version: 3.0 edition, 2012.

[20] Paul E McKenney, Dietmar Eggeman, and Robin Randhawa. Improving energy efficiency on asymmetric multiprocessing systems. Technical report, Technical report paulmck. 2013.03.07a http://rdrop.com/users/paulmck/techreports/AMPenergy.2013.03.07a.pdf, 2013.

[21] Jian Chen and Lizy K. John. Efficient program scheduling for heterogeneous multi-core processors. In Proceedings of the 46th Annual Design Automation Conference, DAC '09, pages 927–930, New York, NY, USA, 2009. ACM.

[22] M. Tim Jones. Inside the linux 2.6 completely fair scheduler. http://www.ibm.com/developerworks/library/l-completely-fair-scheduler/, December 2009. Accessed: 25/11/2013.

[23] S.K. Venkata, I. Ahn, Donghwan Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M.B. Taylor. Sd-vbs: The san diego vision benchmark suite. In Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on, pages 55–64, 2009.

[24] Christian Bienia. Benchmarking Modern Multiprocessors. PhD thesis, Princeton University, January 2011.

[25] Phoronix test suite - an automated, open-source testing framework. http://www.phoronix-test-suite.com/. Accessed: 30/11/2013.

Countable events on the ARMv7-A ISA

Table A.1: Countable events through performance counters on the ARMv7-A ISA [17]

Number	Event counted
0x00	Software increment of the Software Increment Register
0x01	Instruction fetch that causes a Level 1 instruction cache refill
0x02	Instruction fetch that causes a Level 1 instruction TLB refill
0x03	Memory Read or Write operation that causes a Level 1 instruction TLB refill
0x04	Memory Read or Write operation that causes a Level 1 data cache access
0x05	Memory Read or Write operation that causes a Level 1 data TLB refill
0x06	Memory-reading instruction executed
0x07	Memory-writing instruction executed
0x09	Exception taken
0x0A	Exception return executed
0x0B	Instruction that writes to the Context ID register
0x0C	Software change of program counter
0x0D	Immediate branch instruction executed
0x0F	Unaligned load or store
0x10	Branch mispredicted or not predicted
0x11	Cycle count; the register is incremented on every cycle
0x12	Predictable branch speculatively executed
0x13	Data memory access
0x14	Level 1 instruction cache access
0x15	Level 1 data cache write-back
0x16	Level 1 data cache write-back
0x17	Level 2 data cache refill
0x18	Level 2 data cache write-back
0x19	Bus access
0x1A	Local memory error
0x1B	Instruction speculatively executed
0x1C	Instruction write to TTBR
0x1D	Bus cycle
0x1E-0x3F	Reserved