NTNU – Trondheim
Norwegian University of
Science and Technology

Energy Efficiency Experiments on Exynos 5 using OmpSs

Rune Holmgren

December 2014

TDT4501 - Computer Science Specialisation Project
Department of Computer and Information Science
Norwegian University of Science and Technology

Supervisor: Professor Lasse Natvig
Co supervisor: Antonio Garcia Guirado

Problem statement

The report is the pilot project for my planned master thesis in the spring of 2015. The goal of the project is to present preliminary experiment data about the energy efficiency of the Exynos 5 using OmpSs. The experiments should unveil the potential of task based programming on the Exynos 5. They should also explore the ARM big.LITTLE architecture, and discuss how well suited task based programming is to achieve parallelism on heterogeneous platforms.

Another important purpose of this project, is to get familiar with the OmpSs system and experiment platforms. Extensive deep digging experiments does not fit within the scope of this project, and it is more important that the research lay down a foundation for future work.

Acknowledgments

I would like to thank my supervisor Professor Lasse Natvig, who has been guiding me during the last months. I would also like to thank my co-supervisor PhD Antonio Garcia Guirado, who has been helping me with all small and large problems that have come up during the research. They have both been enthusiastically following my work, and continuously guiding me and supplying me with valuable feedback. I am sincerely thankful for all their help.

Abstract

In this project, energy efficiency experiments are implemented in OmpSs, a task based programming model. The experiments are run on two separate Exynos 5 SoC based platforms, with and without ARMs big.LITTLE heterogeneous architecture. Some performance experiments were run on Arndale Board with a dual core ARM Cortex-A15. The goal of this was to get OmpSs running on a known platform, and attempt to recreate Trond Inge Lillesands results from his master thesis. The majority of the experiments were carried out on the ODROID-XU3 with 4 small Cortex-A7 cores and 4 large Cortex-A15 cores. They are able to run simultaneously with ARM's big.LITTLE technology. ODROID-XU3 permitted experiments with detailed energy readings executed on a range of different processor core configurations. The application used for the experiments is 2D-Convolution adapted for OmpSs. The application was chosen because it had already been used with success in an earlier project at the CARD group.

The results show how large cores can be disabled to save energy, although with a large performance trade off. The total energy saved for the system running on only the small cores is significant. There is however a static energy consumption by the other system components, worsening the energy efficiency gain of running on the small cores. For real life applications, it is efficient to run applications requiring performance on the full system, while background processes and tasks with no deadline for completion, can be run with less power on the small cores alone. The task based programming model used by OmpSs proved well suited for heterogeneous architectures, without manually adapting the program. The results indicate that OmpSs did maintain high performance and energy efficiency on the heterogeneous system, and scaled well as the number of cores was increased.

Contents

Chapter 1

Introduction

1.1 Motivation

Increase of performance and power efficiency are the main goal of processor designers. Unfortunately we are currently reaching the limits of our current techniques. For some time, hardware developers have been struggling to achieve increased performance. Heat prevent us from driving the clock frequencies higher, while memory performance is lagging more and more behind. A solution to enable continued performance growth is multi core processors, and for the last decade this has been the focus of research. Unfortunately adding cores will not be a sustainable solution forever. As the amount of cores grow, they are still competing for the same system resources and may have to wait for each other to complete calculations on data with dependencies [1].

A promising solution to this issue is heterogeneous multi-processor systems. Heterogeneous multi-processor systems utilize multiple different processor cores in the same system. This allow different parts of a program to be executed on a suitable processor. By using a suitable core for each part of the program it is possible to achieve better performance and energy efficiency than homogeneous multi-processor systems [2]. Programming of such systems is challenging, as it require manual labor to adapt the system for different processors. Task based programming may be a solution, as it allow the programmer to write the tasks in a general manner, leaving the scheduler to distribute the tasks.

1.2 Project Scope and Goal

This pilot projects main goal is to do preliminary research and experiments on the energy efficiency of the Exynos 5 processor, with the intent to use the results next spring in my master thesis. The goal of this research is to explore the potential of the task based programming model on heterogeneous multi-processor systems.

1.3 Problem Statement Interpretation and Approach

- Task 1: Get the OmpSs task based programming model framework running on both test platforms.
- Task 2: Implement or adapt a suitable experiment application for testing energy efficiency.
- Task 3: Implement some energy efficiency measurement application for ODROID-XU3.
- Task 4: Optimize experiment applications for both platforms.
- Task 5: Gather performance and energy efficiency results from the experiment applications on both platforms.
- Task 6: Analyze, present and discuss experiment results.

Task 1

This project will present results on programs utilizing task based programming. It is essential to get the OmpSs framework, which will be used in the experiments, running on both test platforms. This task should be solved before progressing any further, as any issues here would render further work useless.

Task 2

To be able to determine anything about the potency of the task based programming model on our test platforms, we need a test application. The application should be suitable for the parallel environment of our test platforms, and include some kind of performance metric. The purpose of developing or adapting such an application is to have a way of gathering data about the platform. As a result of this, it is not important what the application does, as long as it does something meaningful.

Task 3

The ODROID-XU3 include current and voltage sensors. Recording data from these sensors during experiment execution will be the source of energy metrics. A software solution to gather these data will have to be implemented before any of the data necessary for the rest of the project can be gathered.

Task 4

Parallel applications will always behave different on different platforms. It is possible to manually tune such applications to specific platforms. This is work that can be done very thoroughly,

but this is time consuming work that is not the main goal of this project. There should at least be made an effort to make some optimizations to the applications, to ensure that they are performing well on the test platforms.

Task 5

When all preceding steps are completed, it is time to gather data. The application from task 2 should be tuned with the optimizations from task 4. Then it should be executed with the energy measurements tool from task 3. The resulting data should hopefully contain interesting information to discuss.

Task 6

The data from the application should be analyzed. The focus of this analysis should be the energy efficiency with attention to trade off against performance. It is interesting to see how the different test platforms and configurations perform compared to each other. There will be many comparisons to make here. The promising results should be elaborated, and their potential discussed.

1.4 Outline

This is the outline of the report, explaining what each part of it contains.

Introduction

In the introduction chapter, the report and it's purpose are introduced. The motivation of the paper is presented, together with its goal. There is also an explanation of how the problem was interpreted and approached.

Background

In the background chapter, existing theory and related work is presented. Most of the general concepts necessary to understand the rest of the project is presented briefly. The main areas explained are energy efficiency and measurements, vector instructions, the task based programming model and heterogeneous multi-processors. In addition, the experiments used in the paper and the platforms they were run on are introduced.

Setup and methodology

In the setup and methodology chapter, the technical details about how the experiments were run are explained. This includes listings of software used, with version numbers and details on

how it was used, including flags used for experiment compilation. There are also explanations about how performance and energy efficiency measurement were gathered.

Results and discussion

In this chapter the results of the experiments are presented and discussed. The experiments presented are covering optimizations, performance, and power and energy measurements. The discussion elaborates what the results mean for the potential of task based programming on heterogeneous multi-processors.

Conclusion

In this chapter the conclusions we are able to draw from the discussions are presented. Central here are the results indicating the potential of task based programming on heterogeneous multi-processors.

Future work

In the future work chapter, the unexplored aspects of the research are presented. There are a lot of tasks that were not included in this project's scope, as this was a pilot project. Many of these tasks will hopefully be addressed in the planned master thesis built on what was learned from this project. These tasks are presented in this chapter.

Chapter 2

Background

2.1 Energy efficiency

Computers consume energy. Achieving low energy consumption is an area of great interest, both in super computers where power is expensive, and in consumer hardware where low power mean longer battery life. The energy consumption of a system depends both on hardware and software. The hardware of the system will typically run within a narrow range of voltages, and the components of the system have some current draw when they are working. The programmer can not affect how large these values are. What can be done however, is to ensure that the software does not consume energy unnecessary. Some of the components in the system that are not used, can be turned off. Some of the features of a processor may also cause it to consume more power. An example is NEON vector instructions, which have been shown to increase the power usage of the system by up to 20% [3] when used. Turning off components or features does save power, but there will often be a performance loss. When energy efficiency is a goal, there is always questions about how much power can be saved, before the execution time is extended so much that there is a total loss in energy consumed. Because of this, it is interesting to analyze programs with regard to energy consumption.

Energy measurement

The interesting power properties of a system is voltage and current. In systems featuring sensors for such data, the data can be gathered while executing a program. By analyzing the data it is possible to optimize the program for energy efficiency.

The power of the system is the product of the voltage and current, as shown in Equation 2.1. This is interesting to observe in applications that run when the system is standing by. When execution time doesn't matter, the energy consumed per time becomes the only important metric.

$$Power = Current \times Voltage \tag{2.1}$$

Often the system is executing a program for a duration, and as soon as it is done the system can be shut down. In such cases it is interesting to measure how much energy the system consumed during execution. The product of the systems power and the problems execution time, is as shown in Equation 2.2 the total amount of energy consumed.

$$Energy\ consumption(Joule) = Power(Watt) \times Execution\ time(Second) \tag{2.2}$$

There are many cases where a balance between energy efficiency and the performance trade off is interesting. This is when the energy delay product of the system become interesting. This is an interesting property for both scientific experiments and programs executed on regular systems. In research experiments the goal may be to obtain energy efficiency or performance. In either case, it is sensible to remember that the actual goal will normally be a balance between the two. On regular systems in real life applications, both energy and performance matter. Users do not want to wait longer than necessary for the system to complete its task. Neither do they want the system to run out of battery, or the power bill to grow too big.

$$Energy\ delay\ product = Energy \times Execution\ time \tag{2.3}$$

The energy delay product of a system is calculated by multiplying the execution time of the application with its energy, as shown in Equation 2.3.

2.2 NEON

NEON is a general-purpose single input multiple data (SIMD) technology implemented in the ARM Cortex-A series of processors. It is able to run SIMD instructions on 128 bit registers. By utilizing the NEON unit of the ARM processors, it is possible to achieve parallelism in each separate core. This will often open for great performance boost on problems like the one explored in this paper. Each register may be filled with single precision floating point numbers ranging from 8 to 64-bit each. This mean that the programmer may choose to fill them with 16 8-bit numbers, 8 16-bit numbers, 4 32-bit numbers, 2 64-bit numbers or a single 128-bit number. This allows a single thread to achieve parallelism. In future generations of the ARM ISA there will be support for other data types as well [4]. Different implementations of NEON exist in the Cortex-A cores. While the even the simple implementations in smaller cores like the Cortex-A7 can give great performance boost, the implementations present in the newest cores are performing even better. The A15 offer two NEON units, and the instruction pipeline to start the cores are shorter than in simpler implementations.

2.3 Task based programming

Task based programming allow a programmer to work with parallel programs, with an abstraction from the parallelization itself. When programming with this model, the program can be split into tasks which can run in parallel. When the program run, it will run a task manager as part of the program. This task manager can dynamically assign tasks to the processors, and the programmer does not have to handle all the time consuming tasks related to manual parallelization. As long as the programmer correctly handle task dependencies in the parallelized code, it will be possible to write this kind of code as if it was serial.

The task based programming model also allow simpler development of portable programs. When the program is running tasks on available CPUs, it is not a problem to allow it to run on larger or smaller numbers of processors, and even clusters can support the program. This model even allow the tasks to run on different types of processors in a heterogeneous environment.

2.4 OpenMP Super scalar

OpenMP Super scalar (OmpSs) is a extension of the OpenMP API to integrate features from the StarSs programming model. It is currently under development at the Barcelona Supercomputing Center. The goal of OmpSs is to extend the programming model to support a wide range of processors. The OmpSs programming model run on a wide variety of different systems, such as traditional personal computers, clusters, shared memory systems and heterogeneous processors [5]. While the software is not yet completed or fully tested, there have been several reports exploring it's potential [3][6]. The results have proven OmpSs as an efficient solution on a wide range of platforms.

2.5 Heterogeneous multi-processor

Heterogeneous multi-processor systems have multiple different processors, opposed to traditional multi-processor systems. A typical modern processor have several processors, and a program can run effectively by having threads running parts of their work on each of them. This work is often of such a nature that it can run better on a different processor. Sometimes it can run just as well on multiple simple processor, while using less die space and energy. In other instances, an advanced processor with some special capabilities, like vector instructions, can be more efficient.

This kind of processors have a potential to help us overcome the challenges that are emerging in processor development. Unfortunately they also introduce several new challenges. Both regular programs and operating systems can demand changes to be able to utilize such systems.

2.6 ARM big.LITTLE

ARM big.LITTLE is a power-optimization technology that allow ARM processors with a common instruction set and feature set, but with different performance, to cooperate. A system with ARM big.LITTLE will feature high performance processor cores, which meet the performance demand from modern systems. It will also feature energy efficiency tuned low power cores with lower performance, which meet the energy efficiency demand of modern systems. It is possible to run high performance applications on the large processors, or even both the large and small processors simultaneously. Applications demanding less performance can be run without use of the high performance cores. In this way the system can save power, and increase battery life or lower the power bill of the system.

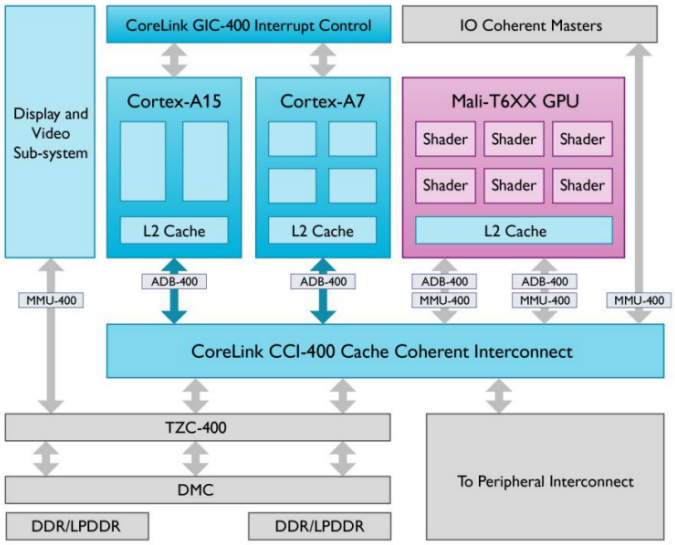

Figure 2.1: Big.LITTLE architecture with Cortex-A15 and Cortex-A7 cores.

When utilizing systems with ARM big.LITTLE, it is not necessary for the programmer to specifically program for this system. Big.LITTLE MP is an underlaying software that will automatically and seamlessly move workloads between processor cores. When the system is running general parallel problems, the added small cores in systems can give a performance increase of over 40%. At the same time, tasks with low performance demand can exhibit power savings in the range of 30%-50% on the SoC level [7]. In Figure 2.1 the typical architecture of a typical ARM big.LITTLE system with Cortex-A15 cores and Cortex-A7 cores is shown.

2.7 Experiment platforms

2.7.1 Arndale Board

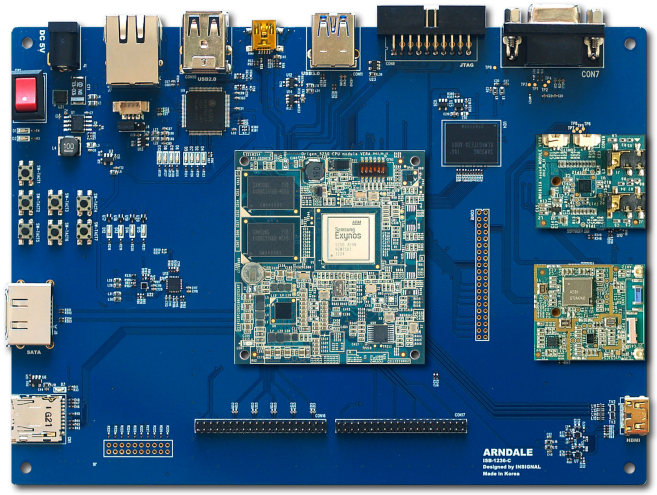

Figure 2.2: Arndale Board

The Arndale Board is the single board computing system shown in Figure 2.2. It is fitted with an Exynos 5250 SoC, which contain a dual core ARM Cortex-A15 , as well as an ARM Mali-T604 GPU. This computer offer a range of supported Linux distributions, as well as the OmpSs programming model. The Arndale Board was used for some preliminary research in this project. It was chosen because there was experience from earlier student projects using this board. The computer was used in the 2014 master thesis "Acceleration with OmpSs and Neon/OpenCL on ARM Processor" by Trond Inge Lillesand [3]. The thesis lay a lot of the foundation for this pilot project and planned master thesis. In Table 2.1 the system specifications of the board are listed.

SoC	Samsung Exynos 5250
CPU	
Model	ARM Cortex-A15
Manufacturing process	32nm
Maximum clock frequency	1.7GHz
Number of cores	2
L2 Cache	1MB
L1 Cache	32KB
GPU	
Model	ARM Mali-T604
Maximum clock frequency	600 MHz
Memory	
Available memory	2 GB
Maximum clock frequency	800MHz
Operating system	
Distribution	Linux Ubuntu

Table 2.1: Arndale Board specifications

2.7.2 ODROID-XU3

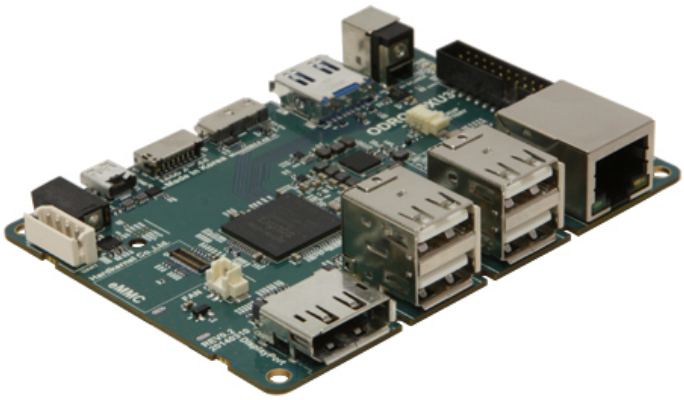

Figure 2.3: ODROID-XU3

The ODROID-XU3 is a new single-board computing system depicted in Figure 2.3, offering interesting properties for these experiments. The system has an Exynos 5422 heterogeneous SoC. Exynos 5422 has a quad core ARM Cortex-A15 CPU and a ARM Mali-T628 GPU, but also a smaller quad core ARM Cortex-A7 processor. These 3 different processing units can be used simultaneously to solve problems. They use ARMs big.LITTLE architecture, described in section 2.6, to achieve this heterogeneous cooperation. In this paper, and the planned master thesis following it, the potency of this kind of heterogeneous processor will be explored. This is the system that

will run most of the experiments, including all energy efficiency experiments. In Table 2.2 the system specifications of the board are listed.

SoC	Samsung Exynos 5422
CPU 1	
Model	ARM Cortex-A15
Manufacturing process	32nm
Maximum clock frequency	2.0GHz
Number of cores	4
L2 Cache	512KB
L1 Cache	32KB/32KB I/D
CPU 2	
Model	ARM Cortex-A7
Manufacturing process	32nm
Maximum clock frequency	1.4GHz
Number of cores	4
L2 Cache	2MB
L1 Cache	32KB/32KB I/D
GPU	
Model	ARM Mali-T628 MP6
Maximum clock frequency	600 MHz
Memory	
Available memory	2 GB
Maximum clock frequency	933MHz
Operating system	
Distribution	Linux odroid
Version	3.10.54+

Table 2.2: ODROID-XU3 specifications

Power monitoring

The ODROID-XU3 comes with integrated power monitoring tools. Implemented in hardware, it has got 4 current sensors sitting on the power pins of the Exynos SoC. The energy monitors are indicated in Figure 2.4. These monitor the current going through the large CPU cores, the small CPU cores, memory and GPU respectively. In addition to the current sensors, the power management for the SoC is also available to the programmer, making supply voltage to the components known. By using the voltage and current, the power consumption is known. As a whole,

the system offer frequency, voltage, current, temperature and power readings in real time. These fine energy and performance metrics make the system highly suitable for developers. They are able to run their programs, collect energy profiling data and optimize their software based on the result.

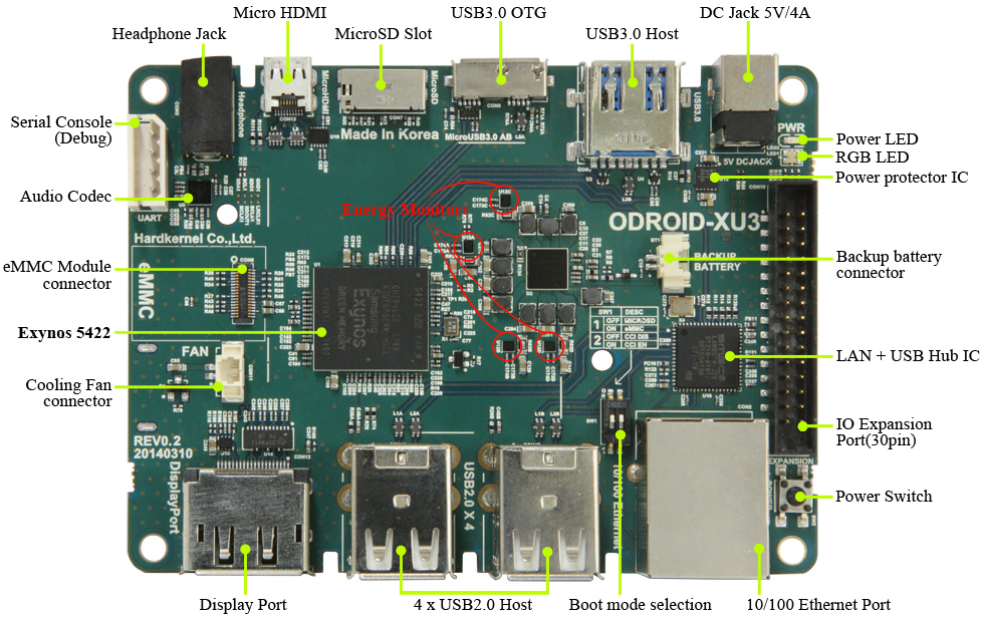

Figure 2.4: ODROID-XU3 annotated (hardkernel.com[8])

Performance

The ODROID-XU3 comes with the ARM Cortex-A15 limited to 2GHz and the ARM Cortex-A7 limited to 1.4GHz. There is 2GB of memory available running at 800 MHz, and the ARM Mali-T628 GPU run at 600 MHz. This performance place it at the higher end of SoCs, but not quite in the top, as it is beaten by systems like TODO A reference system here). The ODROID-XU3 feature the new eMMC 5.0 standard for storage. The performance of this standard outperform both older eMMC standards, as well as memory card readers, which other similar systems may contain. The ODROID-XU3 can achieve a read/write performance of 198/74 MB/s[8], which a lot better than older systems like the Arndale Board.

In addition to the eight processor cores, the system also feature a ARM Mali-T628. The Mali-T628 function both as a regular GPU, as well as a GPGPU. It support both OpenGL and DirectX, and is able to produce graphics for all but the most demanding gaming and simulation purposes. In addition to this it is able to run computations with OpenCL. This mean that problems with parallel parts, can be solved efficiently utilizing the GPU.

2.7.3 ARM Cortex-A15

The ARM Cortex-A15 is a high performance processor core designed for use in a wide range of applications, from small embedded systems, to larger consumer electronics. The specifications of the core are listed in Table 2.3. This processor core is present in both the Arndale Board and the ODROID-XU3, who have 2 and 4 such cores respectively. In the latter, 4 smaller ARM Cortex-A7 cores are also featured, and all can be used simultaneously.

Frequency	1.0 GHz to 2.5GHz
L1 Cache	64KB
L2 Cache	4 MB
L3 Cache	None in core, may be implemented shared in multi core system.
Architecture	ARMv7-A
Architecture	ARMv7-A
Supported features	TrustZone® security technology NEON™ Advanced SIMD DSP & SIMD extensions VFPv4 Floating point Hardware vitalization support Integer Divide Fused MAC Hypervisor debug instructions
Memory management	40-bit ARMv7 Memory Management Unit

Table 2.3: ARM Cortex-A15 specifications

2.7.4 ARM Cortex-A7

The ARM Cortex-A7 is designed to be a low power alternative to the ARM Cortex-A15 and ARM Cortex-A17, with the same supported ISA and features. The specifications of the core are listed in Table 2.4. This enable the ARM Cortex-A7 to be paired with it's larger relatives in a ARM big.LITTLE configuration. The Exynos 5422 SoC on the ODROID-XU3 board feature 4 of these cores.

Frequency	1.2 GHz to 1.6GHz
L1 Cache	8-64KB
L2 Cache	up to 1 MB
L3 Cache	None in core, may be implemented shared in multi core system.
Architecture	ARMv7-A
Supported features	ARM Thumb-2 TrustZone® security technology NEON™ Advanced SIMD DSP & SIMD extensions VFPv4 Floating point Hardware vitalization support Integer Divide Fused MAC Hypervisor debug instructions
Memory management	40-bit ARMv7 Memory Management Unit

Table 2.4: ARM Cortex-A7 specifications

2.7.5 ARM Mali-T604

This is a high performance low power mobile GPU. The specifications of the GPU are listed in Table 2.5. The Arndale Board feature one of these in it's Exynos 5250 SoC.

Frequency	533 MHz 17 GFLOPS
Multi core support	1-4 cores
API Support	OpenGL 1.1, 2.0, 3.0 and 3.1 OpenCL 1.1 DirectX 11 RenderScript
Anti-Aliasing	4xFSAA with minimal performance drop 16xFSAA
Cache	32-256KB L2 cache

Table 2.5: ARM Mali-T604 specifications

2.7.6 ARM Mali-T628

This is a high performance low power mobile GPU. The specifications of the GPU are listed in Table 2.6. The ODROID-XU3 board feature one of these in it's Exynos 5244 SoC.

Frequency	533/695 MHz 17/23.7 GFLOPS
Multi core support	1-8 cores
API Support	OpenGL 1.1, 2.0, 3.0 and 3.1 OpenCL 1.1 DirectX 11 RenderScript
Anti-Aliasing	4xFSAA with minimal performance drop 16xFSAA
Cache	32-256KB L2 cache

Table 2.6: ARM Mali-T628 specifications

2.8 2D-Convolution

Convolution is a mathematical operation to combine two functions into a third output function. The resulting output function maintain properties from both the input functions. It is useful in a wide range of problems within mathematics, statistics, electrical engineering and computer science. An example is computer vision, which can use convolutions to filter data. Use an image as the first input and a sobel filter as the second. The output would then be an image with all emphasized.

2D-Convolution on discrete input data will have an output with each pixel [m,n] defined as [3]:

$$y[m,n] = x[m,n] \times h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} input[i,j] \times filter[1-i, 1-j] \tag{2.4}$$

The limits are in reality smaller than ∞ as they can be set to the size of the input image and filter. 2D-Convolution is convenient for this paper, as each pixel of the output is independent. This computation for each pixel or small group of pixels can made a task in OmpSs.

In the implementation of 2D-Convolution used in this project, Equation 2.4 is run for each pixel in the input image. The limits are set to the size of the input. This result Equation 2.5 [3]. Note that this is the mostly the same implementation as Lillesand used in him master thesis, but with some code added for energy efficiency measurements.

$$Y = X \times H = \sum_{m=0}^{W} \sum_{n=0}^{H} \sum_{j=0}^{w} \sum_{i=0}^{h} input[i,j] \times filter[1-i, 1-j] \tag{2.5}$$

The 4 loops in Equation 2.5 show that the number of operations in this problem is $W \times H \times w \times h$. This is a time complexity of $\mathcal{O}(h^4)$ when W=H=w=h [9].

Chapter 3

Setup and Methodology

3.1 Compilation and running of applications

The 2D-Convolution test application is compiled with the following command:

```
mcc 2d-convolution.c -ompss -mfpu=neon
```

This generate an executable which can be run without any flags.

3.2 Configuring processor cores

The processor cores of the Exynos 5422 can be turned on and off with the system running. Turning off a processor core is done by the operating system without the process noticing that it was moved. The program will simply be moved, and complete its task on a new processor core. Because of this it is possible to run different parts of a program in different processor configurations.

Turning on and off a processor core is done by writing wither a 0 or 1 to /sys/devices/system/cpu/cpuN/online. Processor 0-3 are the 4 ARM Cortex-A7 cores. Processor 4-7 are the 4 ARM Cortex-A15 cores. It is not possible to turn processor 0 off by writing a 0 to the corresponding file. Because of this it is left running in all experiments. In future work on the platform, other methods for turning off processor 0 may be tried. This is an example of a command that will turn off processor 5:

```
sudo bash -c 'echo 0 > /sys/devices/system/cpu/cpu5/online'
```

3.3 Choice of experiment platforms

When work on the project started, Arndale Board was used. The goal at the time was to run experiments on Arndale Board, and then expand to Arndale Octa Board. The Arndale Octa Board feature an Samsung Exynos 5420 SoC, which is almost identical to the Samsung Exynos 5422 on

ODROID-XU3, but with somewhat lower performance. Then the time to start using the Arndale Octa Board came, we had just received the ODROID-XU3. It was decided that the Arndale Octa Board should be dropped in favor of the ODROID-XU3. This was because ODROID-XU3 is a newer board and feature the on board energy sensors used in this project.

3.4 Performance measurement

In the experiments being run in this pilot project, execution time was used as the primary metric for performance. While running the experiments, the POSIX function gettimeofday() is used before and after running the region of interest. The time difference is a used as a measurement of how good the performance is.

In the planned master thesis, PAPI (Performance API) may be used to measure more detailed aspects of the performance. PAPI is able to access a lot of different performance measurements from systems. The available metrics include cache utilization and hit rate, memory utilization, bus utilization and others.

3.5 Energy measurement on ODROID-XU3

As elaborated in Section 2.7.2, the ODROID-XU3 feature 4 current sensors. These current sensors are used to monitor the current flow to the large CPU, small CPU, GPU and memory respectively. In addition to the current sensors, it is possible to read the supply voltage to each of these components. Based on these two sensor readings, we can calculate the power consumption of each of the components in real time while running our applications, as shown in Equation 3.1. Using this scheme for power measurement, we are able to get readings with high precision of each component. The results will show how the consumption vary with different application configurations both for the application as a whole and for different stages of execution.

$$Power(Watt) = Current(Ampere) \times Voltage(Volt) \tag{3.1}$$

There is no way to avoid affecting performance with the energy measurements. To keep accuracy high with minimal interference with the problem execution, a delay of 200ms between each measurement was used.

Chapter 4

Result and Discussion

In this chapter the results of the experiments described in earlier chapters are presented discussed. The chapter is divided into three sections, covering the 3 different types experiments that were run. First the experimental optimization towards our systems properties are presented. Then the experiment results regarding system performance will be presented. Finally the results regarding the systems energy efficiency are presented. The performance trade off from running energy efficient is also asserted here.

4.1 Optimization

Different systems may perform better with different programs. To ensure that the results we find are represented by programs well suited for the system, the experiments were tested with different degrees of loop unrolling. By unrolling loop iterations, the task sizes increase. This reduce the overhead of initiating iterations. It also reduce overhead related to loop control, like end of loop tests and loading data into new memory locations necessary for each loop iteration. There is however a limit to how much unrolling can be done. At some point the size of the task will create difficulties like early cache eviction. Data used by each loop iteration may be evicted, but even more critical is evicted instructions. If the instructions for the loop does not fit in cache, there will be a lot of cache misses. The optimal amount of loop unrolling may vary from system to system. Because of this, experiments were run on all the processor configurations that will be used for later experiments, with different unroll degrees.

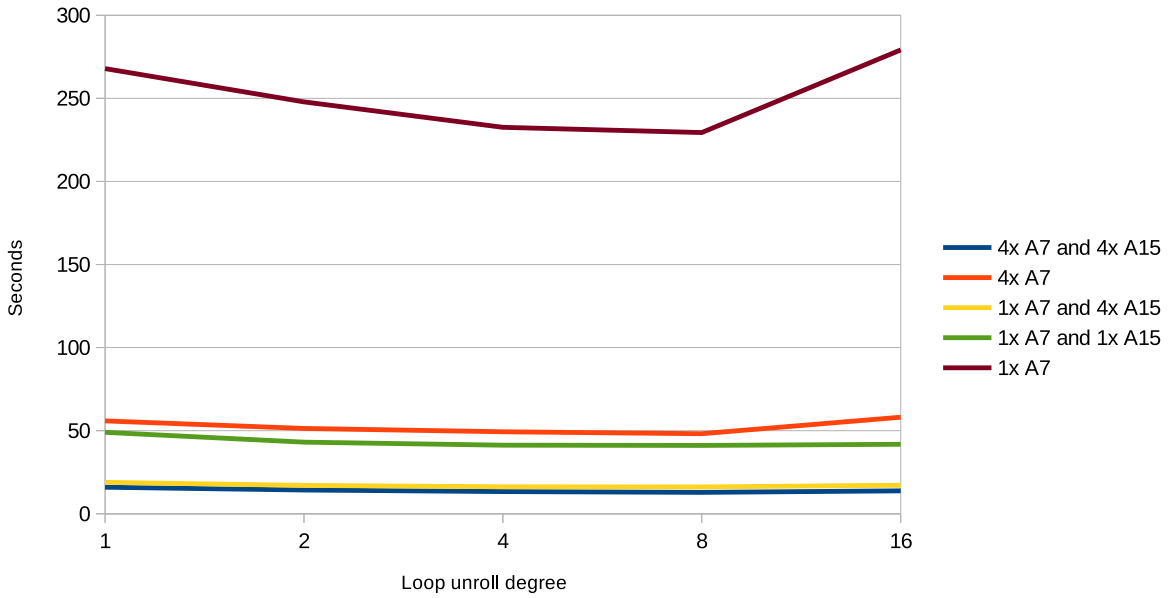

Figure 4.1: Execution time of 2D-Convolution with different degrees of loop unrolling running on different processor configurations.

Processor configuration	Loop unroll degree				
	1	2	4	8	16
4x Cortex-A7 and 4x Cortex-A15	15.9875	14.3006	13.3692	12.8891	13.8013
4x Cortex-A7	55.8885	51.3407	49.3267	48.1665	58.0834
1x Cortex-A7 and 4x Cortex-A15	18.9248	17.082	16.279	16.1658	17.1759
1x Cortex-A7 and 1x Cortex-A15	49.0195	43.1061	41.2549	41.1286	41.8136
1x Cortex-A7	267.8882	247.8783	232.5777	229.3884	279.1135

Table 4.1: Execution time of 2D-Convolution with different degrees of loop unrolling on different processor configurations.

The results in Figure 4.1 and Table 4.1 show that a loop unroll degree of 8 was optimal for this specific implementation of 2D-Convolution on ODROID-XU3. This was observed across the results with all tested processor configurations. Because of this result, the remaining experiments are all run with a loop unroll degree of 8.

As mentioned loop unrolling is limited. As the size of the loop body increase, the number of instructions for the loop increase. When the instructions no longer fit in cache, the execution of the loop will be halted to load more instructions each iteration. There is reason to suspect that this is the main reason for the performance loss at loop unroll degree 16. In addition data cache misses may also occur more often with larger loop bodies. As the amount of data accessed

by each iteration grow, there may occur more cache misses. In 2D-Convolution there are many read write operations in the loop body, but they are accessing data in such an order that it should not result in an increased cache miss rate.

4.2 Performance

Even with energy efficiency as the focus, performance data are still interesting. The data are both useful on their own to observe the power of the system, as well as being a way to compare energy results. When the energy efficiency of a system is measured, it is important to look at the energy data of the system in light of it's computational power. A system consuming low amounts of power is not as impressive if it is equally low performing.

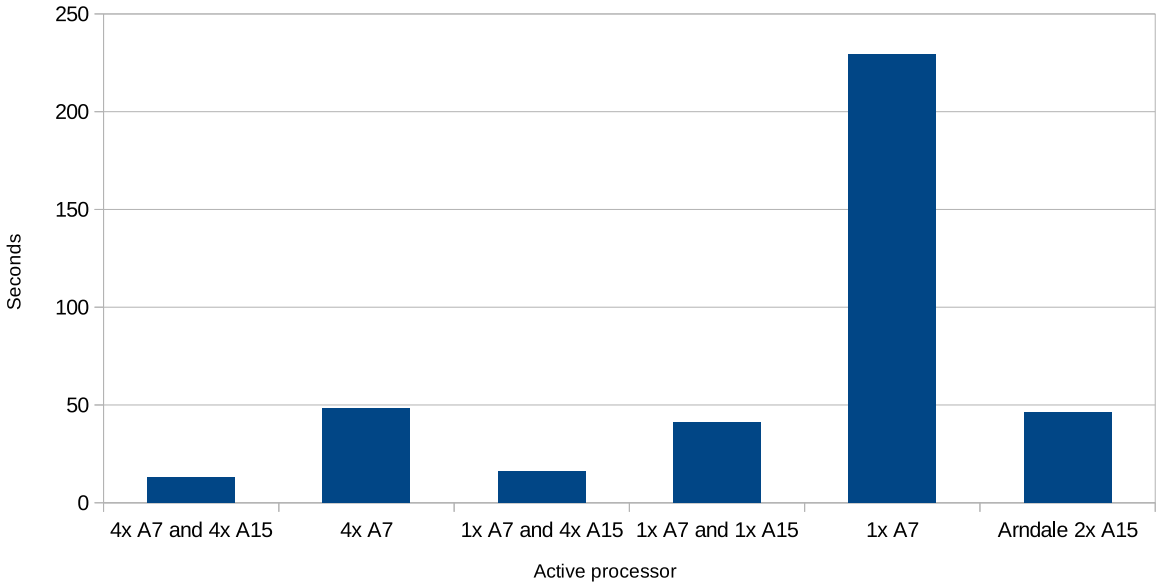

Figure 4.2: Execution time of 2D-Convolution on different processor configurations. All results are from ODROID-XU3 were nothing is specified. The Arndale Board is running at both it's ARM Cortex-A15 cores. The ARM Cortex-A15 cores of the ODROID-XU3 are running at 2.0 GHz, while they are running at 1.7 GHz on the Arndale Board.

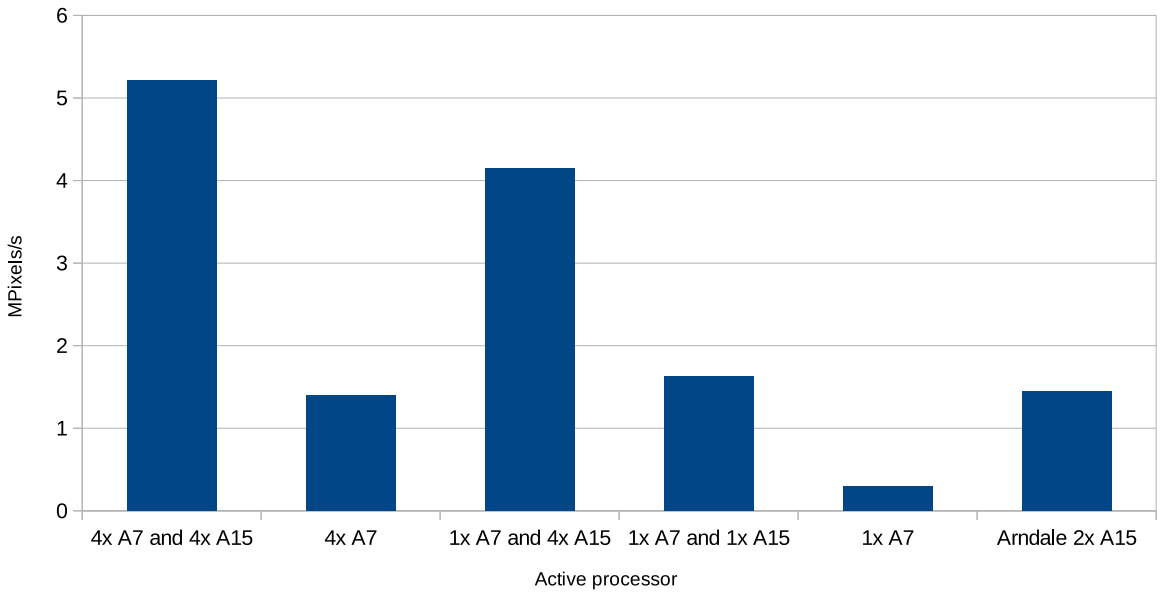

Figure 4.3: MPixels/s of 2D-Convolution running on different processor configurations. All results are from ODROID-XU3 were nothing is specified. The Arndale Board is running at both it's ARM Cortex-A15 cores. The ARM Cortex-A15 cores of the ODROID-XU3 are running at 2.0 GHz, while they are running at 1.7 GHz on the Arndale Board.

Processor configuration	Execution time (s)	Performance (MPixels/s)
4x Cortex-A7 and 4x Cortex-A15	12.8891	5.2066
4x Cortex-A7	48.1665	1.3932
1x Cortex-A7 and 4x Cortex-A15	16.1658	4.1512
1x Cortex-A7 and 1x Cortex-A15	41.1286	1.6316
1x Cortex-A7	229.3884	0.2925
Arndale Board with 2x Cortex-A15	46.3942	1.4465

Table 4.2: Performance of 2D-Convolution with different processor configurations. All results are from ODROID-XU3 were nothing is specified. The ARM Cortex-A15 cores of the ODROID-XU3 are running at 2.0 GHz, while they are running at 1.7 GHz on the Arndale Board.

In Figure 4.2, Figure 4.3 and Table 4.2 the results of running 2D-Convolution with a loop unroll degree of 8 of different processor configurations are presented. As was to be expected, the large cores perform a lot better than the small ones. Running the system with all 8 cores powered perform 3.7 times better than the small cores alone. This is a large performance trade off, but the small cores are still useful if they have a low enough power consumption.

An interesting result here is that adding cores scale sub linearly. The performance of 4 ARM Cortex-A7 cores and 4 ARM Cortex-A15 is only 3.19 times better than a single ARM Cortex-A7

and a single ARM Cortex-A15. The reason behind this is likely to be related to memory congestion. Even though 2D-Convolution is embarrassingly parallel, processor communication is not the only overhead with parallelization. When running more processors, there are more processors to fight over the memory bandwidth and cause congestion. This problem is likely to worsen with other more memory intensive problems, and problems with worse cache utilization. This may lead to some situations where processor configurations, that does not perform well with 2D-Convolution, are well suited. It would be interesting to explore this in the planned master thesis following this pilot project.

The performance of 4 ARM Cortex-A7 cores is 4.8 times better than a single ARM Cortex-A7. The small cores seem to scale better than the large ones. Super linear speedup when adding more cores to a problem, is often a result of each core having its own cache. The ARM Cortex-A7 each introduce an additional 32KB/32KB I/D L1 cache. More processors with cache give the system as a whole more available cache. This may also be a result of the other tasks the operating system is running. With less processing power available, the fraction of time spent running the operating system increase. Because of this, there is reason to suspect that the small cores don't really scale as well as the results indicate.

The performance experiments run on the older Arndale Board show that it have worse performance with its ARM Cortex-A15 cores than the ODROID-XU3. The two high power cores on the Arndale board have lower performance than a low power and a high power processor on the ODROID board. If we assume linear scaling and extrapolate, we see that a quad core ARM Cortex-A15 with Arndale specifications would reach a performance of 2.9 MPixels/s. On the ODROID-XU3, the similar setup with 4 ARM Cortex-A15 cores and an ARM Cortex-A7 core, had a performance of 4.2 MPixels/s. The gap between the two is too large to be attributed to the single ARM Cortex-A7. This performance difference is not surprising, as the Arndales cores run at a lower clock frequency. They are also paired with slower memory, which may also contribute to the lower performance.

4.3 Power measurements

As described in chapter 3, separate energy measurements for the different SoC components were gathered during execution of the experiments. Each of the 4 components value was logged every 200 ms. The Arndale Board does not feature any on board energy sensors, and measuring power for this board would have to be done for the board as a whole. As these results would not be comparable to the energy data from ODROID, they were not gathered. Figure 4.4 show the raw energy measurements for a single execution of 2D-Convolution running on all 8 processors. We can here observe the energy consumption of the program throughout the execution for different components. The total energy consumption can be calculated from the data. Observations regarding power usage during different execution stages can also be analyzed. The same kind of data were gathered for a range of different processor configurations.

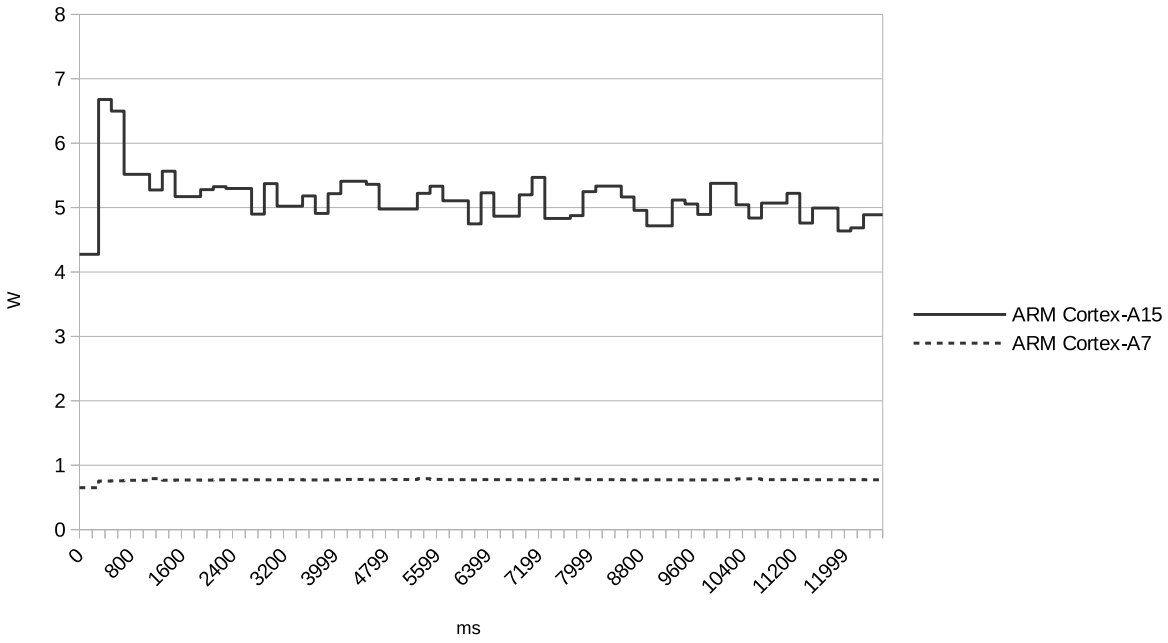

Figure 4.4: Power consumption for the large and small cores during execution of 2D-Convolution with unroll 8x. These results are from a full execution running on all 8 processors.

There is a spike in power consumption on both the small and large cores at the start of the execution. This spike can be caused by a lot of different reasons. There is not enough data to know what caused it. Typical reasons for such spikes are intense operations during initiation of the program, or simply hardware implementations causing a power surge when processor cores are suddenly powered up from idle state. For the rest of the execution there are variations in power consumption, but the consumption vary around the same general level.

For this project the high level of details of the real time power measurements were not used to its full potential. In the planned master thesis building on this report, these measurements may be used for dynamic tuning of the system. It is possible to run parts of problems with different processor configurations, turning on and of cores as they are useful. By running each part of the program on the right configuration, there is a potential to save much power if a suitable application is used.

4.4 Average power measurements

Using the data gathered for each component in different processor configurations we can examine their efficiency. These data are presented in Figure 4.5 and Table 4.3.

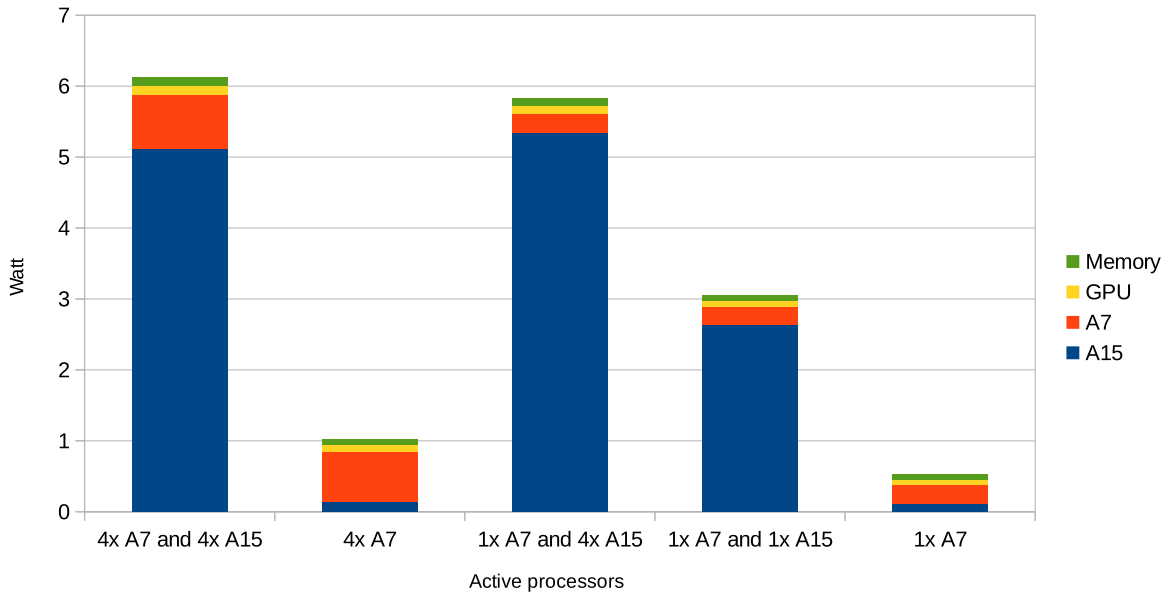

Figure 4.5: Average power consumed per second for different processors configurations running 2D-Convolution. The colors indicate different components, and the full height of each column is the accumulated total for the whole system.

Processor configuration	Power consumption per second for each component				
	Cortex-A15	Cortex-A7	Mali-T628	Memory	Total
4x Cortex-A7 and 4x Cortex-A15	5.1156	0.7693	0.1206	0.1208	6.1264
4x Cortex-A7	0.1362	0.7157	0.0899	0.0817	1.0238
1x Cortex-A7 and 4x Cortex-A15	5.3356	0.2758	0.1161	0.1098	5.8373
1x Cortex-A7 and 1x Cortex-A15	2.6341	0.2540	0.0818	0.0755	3.0456
1x Cortex-A7	0.1197	0.2535	0.0813	0.0709	0.5256

Table 4.3: Execution time of 2D-Convolution with different degrees of loop unrolling on different processor configurations.

As expected we see that the power of the ARM Cortex-A7s is a lot lower than the power of the ARM Cortex-A15. Turning off the power hungry Cortex-A15s reduce the total power to a sixth (0.17). This is useful for applications running that do not require high performance, and want to consume low power. This way of power saving can be increased even more by utilizing only a single Cortex-A7. This configuration has the lowest power consumption per second, and is useful for standing by, or executing minor calculations while standing by. Running on only a single Cortex-A7 reduce the power of the system to a tenth (0.10) of that of the fully powered 8 core system. More than half of this power was used by other components.

In addition to the total power of the system, the results also allow us to analyze the power of each component. Turning off 3 Cortex-A15 cores only reduce the power to about half (0.51). Even turning all the Cortex-A15 cores off still leave some power consumed by them. Because of this power, running the system with only four Cortex-A7 cores still spend 13% of the total energy on the Cortex-A15 cores, which are not doing any work. Similarly turning off 3 Cortex-A7 cores only reduce power to a bit more than two thirds (0.35). These results reveal that the energy for each set of cores are not spent solely on powering the cores themselves, but also some shared elements related to the cores. These shared elements can not be turned partially of, and will consume some power when at least one core is on. An example of such shared elements are the ADB-400 bus to each group of processor cores and the L2 cache, which both can be viewed in Figure 2.1. They even consume some power when the cores are all off. Because of this overhead power from partially powered processors, there rarely occur situations where this is optimal. The results for such processor configurations will be analyzed further in this project to observe whether or not this is the case here.

4.5 Energy consumption measurements

A The power usage of the different system components is interesting. There is however many cases where the system can be put to another task, or shut down, when it is finished. In such cases it is interesting to observe the total amount of energy consumed completing the task. In Figure 4.6 the data for each component power usage is multiplied with the execution time. The data height of each column is the amount of energy consumed.

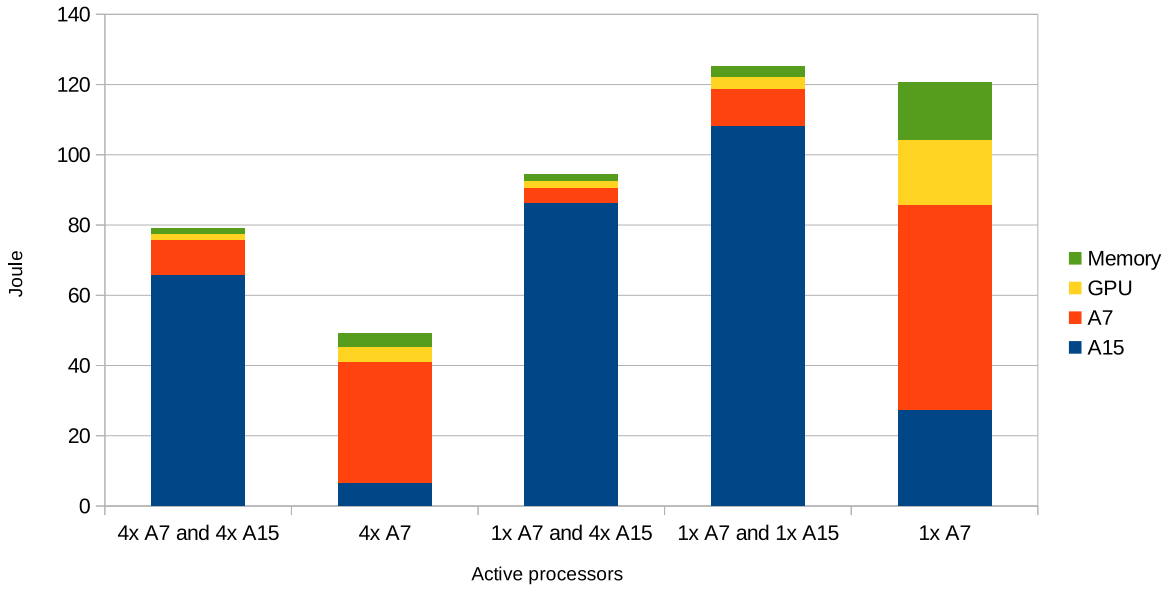

Figure 4.6: Total energy consumption execution the whole 2D-Convolution program. The colors indicate different components, and the full height of each column is the accumulated total power consumed for the whole system.

Figure 4.6 show that, even though the fully powered system with all eight processors operating performed better than all the other processor configurations, it is not the best at energy consumption. The power consumed by a full execution of 2D-Convolution on all 8 processors consume 60% more power than running it on the 4 Cortex-A7 cores alone. Running on the more energy efficient, processors cost performance, but it save energy. This is a trade off that is often observed in energy efficiency. It vary from application to application what is the preferred property, but it is always an issue of balancing the two. In section 4.6 the balance point between performance and energy of these experiments are explored in further depth.

Apart from the results for the full eight core system and the 4 small cores, experiments for the other processor configurations were also carried out. These results however, were not promising. All the other processor configurations had higher energy consumption than the two first configurations. They their performance was also worse than the full system, which completed the execution with a lower power consumption. In other words, they did neither stand out in energy efficiency or performance. There may exist potential situations with very specific maximum execution times and such, where these configurations may be viable. They are however not promising candidates for general applications.

4.6 Energy Delay Product (EDP) measurements

While the energy consumption is a good energy measure in many situations, it is often interesting to compare the energy consumed with it's performance trade off. As explained in section 2.1 energy delay product measurements can be used to examine this trade off. These are the energy data from chapter 4.5 multiplied with the execution time of the respective application execution.

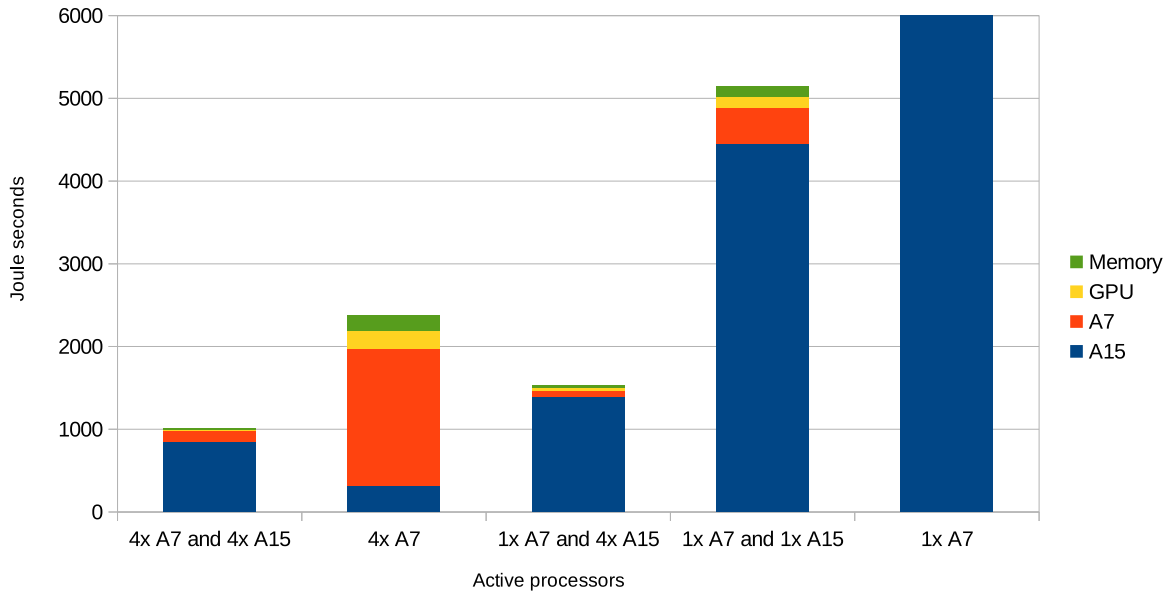

Figure 4.7: Energy delay product after execution the whole 2D-Convolution program with different processor configurations. Note that the EDP of the ARM Cortex-A7 goes beyond the scale of the graph with a value of 27658, to allow for better readability. The colors indicate different components, and the full height of each column is the accumulated energy delay product for the whole system.

The data in Figure 4.7 clearly show that with energy delay product as out goal, running the fully powered system achieve the best results. The energy efficiency gain by switching to the smaller and more energy efficient processors, is not worth the performance trade off. This mean that for applications which require a balanced performance trade off, the more energy efficient Cortex-A7 cores will not be a great alternative in most cases.

It is worth noting that the configuration running on only 1 of the small cores and all the large ones have a decent EDP. There exist special cases where this processor configuration may be useful. An example is applications where task dependencies make it inefficient to utilize all processors. In such a case, the program may run faster by turning off small cores. It is also obvious that running on the single Cortex-A7 processor, while having the lowest power measurement, is unsuited for any application with performance demand.

Chapter 5

Conclusion

In this project, the performance and energy efficiency of the heterogeneous Samsung Exynos 5 is explored running parallel programs with a task based programming model. The potential of different configurations of processor cores was explored. The goal was to uncover the energy efficiency potential of using OmpSs, as well as exploring heterogeneous processors in such use cases. 2D-Convolution was used as a experiment application, as it is easily parallelizable and fit the task based programming model well. The experiments use the OpenMP Super Scalar programming model, developed mainly by Barcelona Supercomputing Center.

5.1 Performance

Several combinations of the large ARM Cortex-A15 cores and small ARM Cortex-A7 cores were tested. The results showed that there is a large performance trade off by turning off the large cores. The performance drop demand great energy savings before the core configuration is worth it. It was also discovered that the performance of the processors scale well, but not quite linearly. This mean that there is potential in exploring applications where parallelization overhead may demand toggling cores dynamically to gain performance.

5.2 Energy efficiency

The small processor cores proved to have a lot lower power consumption than the large ones. This give them both use for low power standby applications, as well as applications where energy is more important than performance. The total energy consumed by a full execution of 2D-Convolution was lowest when running on only the 4 small cores. For tasks requiring low energy consumption with no limit on execution time, running on the 4 small Cortex-A7 processors is clearly optimal. When there is a necessity for balance between performance and energy, the fully powered 8 core system is most promising. The energy delay product of the fully powered system was clearly best for the full system. It is however likely that there exist many applications,

that do not parallelize as well as 2D-Convolution, which will perform better at other processor configurations.

5.3 Heterogeneous multi-processors and the task based programming model

The experiments showed that using OmpSs for parallelization works well on heterogeneous systems. When the number of processors was varied, both the performance and energy efficiency scaled well.

Chapter 6

Future Work

These are some suggestions for future work that may build upon the work in this project.

6.1 OmpSs with OpenCL kernels

A new feature of OmpSs is it's ability to manage OpenCL kernels as tasks. It is possible to issue OpenCL kernels as OmpSs tasks, and have the task manager assign them to GPUs and CPUs. This allow for portable code that can run effectively on a range of different system. It would be interesting to examine the potency of this way of utilizing the GPU, as it save the programmer from the job of manually tuning the load balance between GPU and CPU.

6.2 ARMv8-A 64-bit processors

ARM have created the next generation ARM processors. They run a new instruction set, with support for both 32- and 64-bit instructions. Running similar experiments on such a processor would be interesting.

6.3 Performance measurement

In this project, only simple forms of performance measurement was used. Running the same or similar experiments with access to other performance counters would be interesting. Cache hit rate and utilization, memory utilization, bus utilization and other counters, could help understanding the strengths and weaknesses of the system.

Bibliography

[1] Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. 2007.

[2] Rakesh Kumar et al. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. 2003.

[3] Trond Inge Lillesand. Acceleration with ompss and neon/opencl on arm processor. 2013.

[4] Arm software development tools. http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0473c/CIHDIBDG.html. Accessed: 2014-12-10.

[5] Barcelona Supercomputing Center. Ompss user guide. 2014.

[6] Lien et al. Case studies of multi-core energy efficiency in task based programs. 2012.

[7] Brian Jeff. big.little technology moves towards fully heterogeneous global task scheduling. 2013.

[8] Hardkernel.com. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127&tab_idx=2. Accessed: 2014-10-25.

[9] Wilhelm Burger and Mark J. Bruge. *Principles of Digital Image Processing*. Springer, March 2009.