A Shallow Water Model on Heterogeneous ARM Architectures

Progress Report

Student: Juncheng Li
Supervisors: Graham D. Riley, John R. Gurd
School of Computer Science
University of Manchester
Contents

Chapter 1  Introduction .......................................................... 1
  1.1 Project Objectives ........................................................... 1
  1.2 Report Structure ............................................................ 2

Chapter 2  Background ............................................................ 3
  2.1 Shallow Water Equations .................................................. 4
  2.2 Hardware Architectures .................................................... 5
    2.2.1 Modern Processor Architectures .................................... 5
    2.2.2 Arndale2 ................................................................. 6
    2.2.3 ARM Cortex -A15 Processor ......................................... 6
    2.2.4 ARM Mali-T604 GPU ................................................. 7
  2.3 APIs for Parallelism ....................................................... 9
    2.3.1 Current APIs ........................................................... 9
    2.3.2 OpenMP ................................................................. 11
    2.3.3 OpenCL ................................................................. 12
  2.4 Power Efficiency in HPC .................................................. 13
    2.4.1 Current Studies of Power Efficiency ............................... 13
    2.4.2 Methods for Power Measurement ................................... 15
    2.4.3 Power Measurement for This Project .............................. 16
  2.5 Summary .......................................................................... 16

Chapter 3  Project Progress ...................................................... 18
  3.1 OpenMP on ARM Cortex-A15 Dual CPU ............................... 18
    3.1.1 Compiling and Running ............................................... 18
    3.1.2 Performance ............................................................ 19
  3.2 OpenCL on Mali T-604 GPU .............................................. 20
    3.2.1 Compile And Running ................................................ 20
    3.2.2 Modification to Standard Code ..................................... 22
List of Figures

Figure 2.2.1 ARM Cortex-A15 CPU architecture. .............................................8
Figure 2.2.2 Mali-T604 GPU architecture. .....................................................9
Figure 3.1.1 Performance of OpenMP implementation. ..................................20
Figure 3.2.1 An error returned in the OpenCL benchmark .........................22
Figure 3.2.2 Performance of OpenCL code on GPU and OpenMP on CPU. 25
Figure 4.3.1 The modified project plan. .........................................................30

List of Tables

Table 2.2.1 ARM Cortex-A15 specifications. ..................................................7
Table 2.2.2 ARM Mali-T604 GPU specifications. ..........................................8
Table 2.4.1 Top five supercomputers. .............................................................14
Table 3.1.1 Timings result of OpenMP naïve implementation on ARM CPU. 19
Table 3.2.1 Performance timings of OpenCL implementation on Mali GPU. 25
Table 3.4.1 Performance of OpenCL implementation on Mali and other devices. 26
Abstract

Two previous MSc projects have focused on parallel implementations of a shallow water benchmark on different machines with heterogeneous architectures. The CPUs in these heterogeneous architectures are Complex Instruction Set Computing (CISC) architecture CPUs. In recent years, ARM processors with Reduced Instruction Set Computing (RISC) architecture have seen a success in the commercial market. As ARMv7-A architecture is starting to support multi-core, native floating point and SIMD (single instruction multiple data) instructions, heterogeneous ARM architecture is becoming more and more interesting for high performance computing (HPC).

This MSc project will parallelise and optimise the shallow water benchmark on heterogeneous ARM architecture to achieve a competitive performance and explore the power efficiency of the benchmark. To achieve the aim of the project, a review is made of current heterogeneous hardware architectures, APIs (Application programming interface) for high performance computing, as well as research concerning power efficiency. In terms of progress to-date, naïve parallel implementations of the shallow water benchmark, including OpenMP (Open Multi-Processing) and OpenCL (Open Computing Language) versions, have been run on heterogeneous ARM architecture. To support further research and performance results of these implementations are presented with some initial analysis.
Chapter 1
Introduction

With the development of high performance computing, the trend for exploring software parallelism has been changed from homogeneous architectures which consist of only one type of multi-core CPU to heterogeneous architecture. Heterogeneous architectures usually contain different types of computing units including multi-core CPUs, Graphical Processing Units (GPUs) and accelerators.

Previously, two MSc projects have focused on parallel implementations of a shallow water benchmark on different machines with heterogeneous architectures [1,2]. The CPUs used in these projects are Complex Instruction Set Computing architecture CPUs.

Current HPC systems are established with CISC architecture CPUs, and aim to achieve maximum performance, with rare consideration about energy efficiency. In contrast, the ARM processors is based on Reduced Instruction Set Computing architecture and is mainly designed for mobile and embedded devices [14]. As ARMv7-A architecture is starting to support multi-core, native floating point and SIMD instructions, ARM CPU with area efficiency, energy efficiency and cost-efficiency makes it more interesting in the high performance computing field [21]. However, in recent years, only a few studies in the HPC field about RISC architecture have emerged [21,32,33].

1.1 Project Objectives

This project focuses on a heterogeneous ARM architecture which consists of ARM Cortex-A15 Dual-core CPU and ARM Mali GPU. The following objectives will be targeted:
- Parallelise sequential shallow water benchmark using OpenMP on ARM Cortex-A15 CPU and investigate execution behaviour of the benchmark on ARM and try to optimise overheads.
- Parallelise sequential shallow water benchmark by OpenCL on ARM Mali GPU. Try to optimise overheads with understanding of Mali architecture.
- Try to parallelise the benchmark on heterogeneous architecture and explore its performance.
- Explore power efficiency of both OpenMP and OpenCL implementations on the heterogeneous ARM architecture.

### 1.2 Report Structure

This progress report introduces background of the project, progress of the project made so far, methodology for the project and a revised project plan based on milestones achieved.

Background information about the project is in Chapter 2. Shallow water equations is introduced firstly. Following part is a overview for modern processors, ARM CPU and Mali GPU. Then it comes to popular concurrent programming languages. Final part is about power efficiency in HPC field.

Chapter 3 presents the process of running the naive OpenMP benchmark on ARM CPU and OpenCL benchmark on Mali GPU. A problem has been found to run the OpenCL benchmark on Mali. The process of fixing the bug is given. Finally in the chapter, some initial analysis is presented.

In Chapter 4, the methodology to optimise the implementations, evaluate data and measure power consumption is summarised. Then, milestones achieved so far are concluded. Based on achieved milestones, further plan of the project is shown with a Gantt Chart.

Chapter 5 is a conclusion of this progress report.
Chapter 2
Background

This chapter provides background information for the project. The background contains the theory of shallow water equations, ARM hardware architectures, current concurrent programming languages and APIs, and power efficiency in the HPC field.

The shallow water benchmark is one of the implementations of shallow water equations. Shallow water equations model the propagation of disturbances in water and other incompressible fluids[1]. Section 2.1 gives a brief introduction to the shallow water equations and their application.

Section 2.2 is about heterogeneous hardware architectures. Section 2.2.1 is an overview of modern processor architectures. Section 2.2.2 is an introduction to a machine named Andale2, which is used in the project. The rest of the section is about ARM Cortex–A15 processor and Mali GPU. Specifications of Cortex–A15 and Mali T-604 are given.

A large number of programming languages have been developed to support parallel computing, including MPI[35], OpenMP[3], CUDA[28], OpenCL[26] and OpenACC[27]. A brief review is given in Section 2.3. Detailed information about OpenMP and OpenCL are provided as these languages are used in the project.

Section 2.4 contains recent studies for power efficiency in the high performance computing field. Comparison of methods to measure real power consumption on computing devices are given; advantages and disadvantages of these methods are provided. In Section 2.4.2, a power measurement device for the project is described.
2.1 Shallow Water Equations

Shallow water equations are a set of hyperbolic partial differential equations which model the propagation of disturbances in water and other incompressible fluids [17]. The basic assumption for shallow water equations is that, compared to the wave length of the disturbance, the depth of the wave is smaller. Combined with other assumptions [18], three dimensional wave propagation is simplified to two dimensional hyperbolic partial differential equations. The reduced complexity makes the equations attractive for realistic modelling [19].

Shallow water equations are widely used to model and describe flows in oceans and rivers. However, flows are not limited to water flow. Certain atmospheric flow can be also modelled by shallow water equations, which have been widely used to model weather prediction and atmospheric motion [1,2]. Shallow water equations are also used to simulate tsunami wave propagations [19]. Reference [18] gives the reason why shallow water equations approximation provides a reasonable model in simulation of a tsunami wave.

The shallow water model used in this project is from Sadourny's paper presented in 1975 [20]. The implementation of the shallow water model simulates physical variables such as velocities and potential pressure of fluids on a discrete grid of points in different problem sizes. The \( u \) and \( v \) in the implementation represents the fluids velocities in the \( x \) and \( y \) directions, and \( p \) is the potential pressure. Initial values are given to these variables in the implementation, and these variable are updated over a number of timesteps according to a discretised version the shallow water equations.

The two previous MSc dissertations [1,2] have made brief summaries of Sadourny's paper and mathematical equations of shallow water model. A brief introduction and implementation of shallow water equations on Matlab can be
found in [17]. The theory of shallow water equations is complex and is out of the scope of this project. The book ‘Numerical Methods for Shallow-Water Flow’ [18] provides a detailed description of shallow water equations.

2.2 Hardware Architectures

Section 2.2.1 presents popular heterogeneous architectures in current HPC systems. Section 2.2.2 introduces the machine Arndale2. Sections 2.2.3 is a summary of ARM Cortex-A15 CPU. The last Section 2.2.4 is a overview of Mali GPU.

2.2.1 Modern Processor Architectures

Traditional multi-core processors are composed of many high clock frequency cores and large caches with a good ability to perform task-parallel and data-parallel workloads. With the emergence of various accelerators, such as GPUs and coprocessors, heterogeneous architectures have been attracted more attentions in the HPC era in recent years.

Traditional GPUs only handle graphic computations. While modern graphic architectures provide more powerful computation on 2D and 3D graphs and a larger memory bandwidth, with many shader units supporting vector operations and IEEE floating point precision. The Vector capability of GPUs enable them to implement SIMD execution. More and more GPUs with vector capability are used to do general purpose computing which was traditionally done by CPUs. General-purpose computing on GPU is known as GPGPU [30]. In this situation, GPUs are often treated as accelerators which receive work offloaded from CPU to do computation.

Intel Xeon Phi coprocessors are based on Intel Many Integrated Core (MIC) architecture. Intel Xeon Phi’s core contains vector processing unit (VPU) which features a 512-bit SIMD instruction set. Because a single operation can deal
with a large of workloads, vector units are considered to be power efficient for HPC workloads [16].

ARM architecture is based on a Reduced Instruction Set Computing (RISC) architecture developed by ARM Holdings. ARM Holdings only licenses processor designs, and it does not produce actual processors. Companies including Apple, NVIDIA, Samsung Electronics, implement and manufacture ARM processors [21]. New ARM CPUs with ARMv7-A architecture have multiple cores and support native floating point and SIMD instructions. The ARM Mali GPU designed for mobile devices also contains multiple shader units. These new hardware types make it interesting for ARM heterogeneous architecture to be used in high performance computing. And with a low power consumption design principle, ARM heterogeneous architecture can give a high power efficiency.

2.2.2 Arndale2

Parallelism of shallow water benchmark will mainly be explored on Arndale2, a chrome book with a Samsung Exynos 5 Dual SoC (System-on-Chip). Samsung Exynos 5 Dual contains a ARM Cortex-A15 based 1.7GHz Dual-Core mobile application processor and a Mali-T604 GPU [22]. Arndale2 can be accessed by command ‘ssh username@arndale2.cs.man.ac.uk’.

2.2.3 ARM Cortex-A15 Processor

As mentioned before, Samsung Exynos 5 Dual SoC implemented world’s first dual-core ARM Cortex-A15 processor. ARM Cortex-A15 processor has two cores with a clocking rate of 1.7GHz. Specifications of ARM Cortex-A15 processor are listed in Table 2.2.1.

Figure 2.2.1 shows the architecture of ARM Cortex-A15 CPU with four cores. As we can see from the figure, each core has a ARMv7 32b CPU, Neon Data
(SIMD) Engine, floating point unit and separate data and instruction cache. For memory architecture, it has a shared L2 cache with private L1 cache to each core.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>ARMv7-A architecture with additional architecture extensions for MP and virtualization</td>
</tr>
<tr>
<td>Core</td>
<td>Dual core with a 1.7 GHz clock speed</td>
</tr>
<tr>
<td>Cache</td>
<td>32KB (Instruction)/32KB (Data) L1 Cache and 1MB L2 Cache</td>
</tr>
<tr>
<td>Bus</td>
<td>128-bit Multi-layered bus architecture</td>
</tr>
<tr>
<td>SIMD Engine</td>
<td>ARM Neon SIMD Engine</td>
</tr>
</tbody>
</table>

Table 2.2.1 ARM Cortex-A15 specifications.

### 2.2.4 ARM Mali-T604 GPU

ARM Mali-T604 GPU is designed for visual computing, especially for 2D and 3D graphics. It was built on Midgard architecture and offers scalability from one to four shader cores. A wide range of APIs are supported by Mali-T604, such as OpenGL ES 1.1, OpenGL ES 2.0, OpenGL ES 3.0, DirectX 11 and OpenCL 1.1. The specifications of ARM Mali-T604 GPU are shown in Table 2.2.2.
Figure 2.2.1 ARM Cortex-A15 CPU architecture.

(Source: http://www.arm.com/images/Cortex_A15_large.png [Accessed 01/05/2014])

<table>
<thead>
<tr>
<th>Features</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Midgard architecture</td>
</tr>
<tr>
<td>Cores</td>
<td>4 shader cores</td>
</tr>
<tr>
<td>Max Clock Rate</td>
<td>533 MHz</td>
</tr>
<tr>
<td>Address Bits</td>
<td>64 Bits</td>
</tr>
</tbody>
</table>

Table 2.2.2 ARM Mali-T604 GPU specifications.

Figure 2.1 provides the architecture of ARM Mali-T604 GPU. The memory of the GPU is managed by MMU (Memory Management Unit). As can be seen from the figure, all cores share the unified level 2 cache. There is no private cache for each core.
ARM Mali-T604 GPU supports OpenCL version 1.1. When using OpenCL for ARM Mali-T604 GPU, some optimisation methods must be taken to get a higher performance. Some of these optimisation methods are introduced in Section 2.3.3.

### 2.3 APIs for Parallelism

This section gives an introduction to APIs for parallelism. Current concurrent programming languages and APIs are in Section 2.3.1. Following two sections, Section 2.3.1 and Section 2.3.2, summarise features of OpenMP and OpenCL.

#### 2.3.1 Current APIs

Many concurrent programming languages and APIs have been developed for parallel computing. Shared memory and message passing are the most commonly used programming models [3]. They mainly explore task level parallelism for traditional multi-core CPU systems. OpenMP is widely used
shared memory APIs. Variables stored in shared memory are utilized by these APIs to communicate between multiple threads. Distributed memory uses message passing (MPI) to share variables [35].

With the development of GPUs and coprocessors (FPGAs [8], DSPs [8]), the complexity for parallel programs to utilize these hardware systems has been increased. New extensions (like OpenCL and CUDA) to traditional programming languages, such as C and Fortran, have been created to program these coprocessors. CUDA is a single instruction multiple thread (SIMT) parallel computing programming model invented by NVIDIA and it has gained widespread popularity [24]. However, CUDA only supports NVIDIA’s own GPUs. OpenCL is an industry standard for task-parallel and data-parallel heterogeneous computing [25]. Compared with CUDA, OpenCL supports different types of computational devices such as multi-core CPUs, GPUs, or other accelerators (such as Intel Xeon Phi) [26].

OpenACC provides a directive-based approach for accelerator programming [27]. It allows applications to use accelerators, without managing data or program transfers between CPUs and accelerators, or accelerator startup and shutdown [27]. One paper demonstrates that OpenACC is a viable model for accelerators and it can improve programmer productivity and achieve better performance than OpenCL and CUDA [28].

To investigate the parallelism of shallow water benchmark on shared memory programming model, OpenMP will be used.

ARM holdings has released Mali OpenCL SDK [29] for developing OpenCL 1.1 application on ARM Mali based platforms. In this case, OpenCL will be used to explore performance of ARM heterogeneous architecture. Other APIs including CUDA, and OpenACC does not support ARM architecture [24][27].
2.3.2 OpenMP

OpenMP (Open Multi-Processing) is a set of compiler directives and callable runtime library routines that extend Fortran, C and C++, separately, to achieve shared memory parallelism [3].

A program with OpenMP usually starts with a single thread called the master thread. The program runs sequentially until meeting control structures (e.g. PARALLEL DO). Listing 2.3.1 is an example of OpenMP code on Fortran. The ‘!$OMP PARALLEL DO’ applies to the do loop that immediately follow the directive.

```
!$OMP PARALLEL DO
  do i=1,m
    y(1,i) = 0.
    do j=1,i
      l(j,i) = 2.
    end do
  end do
end do
```

Listing 2.3.1 An example of OpenMP code on Fortran

When the master thread encounters this directive, it creates a set number of threads. The workload of the DO loop is distributed among these threads. Data environment is specified at the beginning of the construct by PRIVATE, SHARED or REDUCTION to different variables [30]. After finishing the execution of the parallel construct, all threads synchronise and master thread continue to execute [31].

Two types of synchronisation in OpenMP are implicit and explicit synchronisation. Information about synchronisation can be found in [3] as well as other details that OpenMP provides, such as runtime library and environment variables.
2.3.3 OpenCL

The Open Computing Language, OpenCL, is an open industry standard for program execution on heterogeneous architecture systems to achieve task-parallel and data-parallel computing.

OpenCL is developed by the Khronos Group which was founded in 2000 by a number of leading companies, including Intel, NVIDIA, and Sun Microsystems. Now, it has developed into a non-profit, member-funded consortium focused on the creation of royalty-free open standards for parallel computing, graphics and dynamic media on a wide variety of platforms and devices.

OpenCL is supported by a wide range of CPUs, GPUs and other microprocessors. A full list of supported hardware can be found in [34].

OpenCL is mainly designed for portability [29]:

- It uses an abstracted memory and execution model.
- There is no requirement to know the application processor or GPU instruction set.
- There is scope for specific hardware optimizations.

However, OpenCL does not guarantee that a kernel can achieve peak performance on different hardware systems [25].

An OpenCL application consists of host code and OpenCL kernels. Host code calls OpenCL APIs, compiles kernels, manages memory buffers, and sets up environments. OpenCL kernels are codes written in OpenCL C language and run on devices such as GPU to perform parallel processing. To get peak performance, these two parts must be written properly.

In OpenCL, a hierarchy of four models are used, including (a)Platform Model, (b)Memory Model, (c)Execution Model and (d)Programming Model. The details of these four models can be found in OpenCL specification [26].
The Mali GPU used in the project, has a different memory model to desktop computers for using OpenCL. Traditional desktop computers have separate global, local and private memories, especially because a graphics-card often has its on-chip local memory. In this case, data must be copied to the local memory and back.

The Mali GPU, as introduced in Section 2.2.4, has a unified memory system in which local memory is physically global memory. So the data copy operations that are used in desktop computers is not required on the heterogeneous ARM architecture [29].

2.4 Power Efficiency in HPC

This section gives an overview of power efficiency in HPC and power measurement methods. Section 2.4.1 contains prior publications about power efficiency. Section 2.4.2 discusses some power measurement methods. Section 2.4.3 introduces a power measurement device that will be used for the project.

2.4.1 Current Studies of Power Efficiency

Prior to 2005, the development of microprocessors relied on increasing clock rate to increase performance. However, faster clock rate leads to more power consumption and some physical limits are being reached [12]. To solve this problem, the development trend has switched to build multi-core processors in recent years [12].

In multi-core era, one system can have many thousands of cores [6]. The following table is the top five supercomputer in the world [13]. It shows the cores and power consumption of each computer.

As can be seen from the table, the top supercomputer, Tianhe-2, has a total number of 3,120,000 cores, and the power of the system is 17,808 kW. Long
hours of computing on these systems can lead to enormous power consumption. Improving the power efficiency for these systems has become an interesting topic. Power efficiency, or power performance, is usually represented by Flop/s per Watt [5].

<table>
<thead>
<tr>
<th>Rank</th>
<th>System</th>
<th>Cores</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tianhe-2 (MilkyWay-2)</td>
<td>3,120,000</td>
<td>17,808</td>
</tr>
<tr>
<td>2</td>
<td>Titan</td>
<td>560,640</td>
<td>8,209</td>
</tr>
<tr>
<td>3</td>
<td>Sequoia</td>
<td>1,572,864</td>
<td>7,890</td>
</tr>
<tr>
<td>4</td>
<td>K computer</td>
<td>705,024</td>
<td>12,660</td>
</tr>
<tr>
<td>5</td>
<td>Mira</td>
<td>786,432</td>
<td>3,945</td>
</tr>
</tbody>
</table>

Table 2.4.1 Top five supercomputers.

Many studies have shown that heterogeneous architectures have a better power performance compared with homogenous multi-core processors [5,6]. However, most studies only focused on CISC CPUs.

ARM processors have dominated mobile-chip market for around 10 years [15]. In recent years, mobile phones deal with more than phone calls and messages. Internet access, music, gaming are also widely used on mobile phones. These various tasks require mobile CPUs to provide higher performance, while the power must be managed and conserved, so that the device can have a longer battery life [15]. In this case, investigating the power performance in ARM systems is meaningful.
2.4.2 Methods for Power Measurement

To calculate power efficiency of a HPC system, performance and power consumption are needed. Compared to power consumption, performance of various types of computing units can be easily obtained [8].

However, a precise power consumption for a given program is hard to measure. Currently, there are two main research directions for power consumption in HPC field. One is to use tools and devices to measure real power consumption of the system or for specific computing unit(s). The other one uses software to model hardware and compute power consumption from the simulation data.

Methods to measure real power consumption also vary. In one paper, they used external connected power meter to measure the whole system’s power consumption [6]. In their experiment, they used a real machine equipped with Intel Core I7 CPU and AMD 4870 GPU. However, in such a system, the power consumption of mother board, hard disk, memory and other devices is considerable and cannot be ignored. Therefore, the result for this experiment may be not very precise.

There is an improved method in another paper [7]. In their approach, each device involved in the computation was measured separately. For example, input current and voltage of the CPU were measured at its 8-pin power plug. Although this method is also an approximate way to measure the power, it is better than the previous one since it distinguishes the power consumed by different devices from the overall system’s consumption.

Further, there is a method mentioned where efficiency was calculated using the peak power consumption of each device, ignoring supporting infrastructure such as RAM, hard disks, networks etc. [8]. However, when program is running, some devices are involved in computation including RAM and hard disk. And
CPU may not reach the peak power consumption. So this method is a rough approximation and may not reflect the real power consumption. Compared with two previous methods, the final result for this experiment is not so accurate.

The project will not use software simulation to get the power consumption, but it is an interesting research topic. For these methods please refer to [9,10,11].

2.4.3 Power Measurement for This Project

To support the research of power efficiency for this project, an ARM based Arndale board in the University of Manchester will be used [Need Ref.]. The Arndale board has the same hardware as the chrome book Arndale2, but it does not contain a monitor. A power measurement device has been inserted on the input power plug of the board which can collect the electric current of the whole system. Power measurement for this project is based on the fixed supply voltage and the measured varying current [32].

Although it measures the whole system’s power, which seems the same as [6], the Arndale board is a integrated board rather than a desktop computer. The Arndale board does not contain devices such as hard disk, CPU fan, etc. So the result of measuring whole system’s power of Arndale board can be more accurate than that present in [6].

2.5 Summary

This chapter gives the background information for the project. An introduction to shallow water equations and its practical applications are in Section 2.1. Section 2.2 contains information about modern processor architecture and the heterogeneous ARM architecture used in the project. Current development trend of concurrent programming languages and APIs is presented in Section 2.3. As OpenMP and OpenCL are two APIs used to parallelise the benchmark,
detailed information of these APIs is given. The Section 2.4 gives the research trend of power efficiency in the HPC field. Mainly focuses were on traditional desktops and servers, while some new studies had emerged in embedded systems. Methods to measure power consumption are compared and discussed. In Section 2.4.2, a power measurement device to collect the input current to the Arndale board for the project is introduced.
Chapter 3
Project Progress

This chapter shows the project progress achieved so far. As mentioned in the introduction chapter and initial report, the aim is to parallelise the code by OpenMP and OpenCL on heterogeneous ARM architecture. Section 3.1 is a naïve implementation running on ARM Cortex-A15 CPU and an initial performance analysis. Section 3.2 contains process of running a naïve implementation on Mali GPU. A problem in the implementation has been fix and initial performance is analysed. Section 3.4 is a brief performance comparison between the OpenCL benchmark on Mali and other architectures.

3.1 OpenMP on ARM Cortex-A15 Dual CPU

Large HPC systems are usually made out of many small shared memory nodes. For codes like shallow water benchmark, which are based on computational grids, as the number of nodes increases, the number of grid points in each node becomes smaller. According to this, the performance of shallow water benchmarks is investigated with a small grid point size with OpenMP on the Arndale2. The dual core CPU can be considered a systems with two nodes. Following is an attempt to run a naïve OpenMP benchmark and obtain some run-time result for further analysis.

3.1.1 Compiling and Running

A previous MSc project [2] implemented and optimised the shallow water benchmark on a machine called Chronos which consists of 4 AMD Opteron 8378 Quad-core processors, 16 cores in total. However, this implementation cannot run properly on Arndale2. The output of computation shows ‘-nan’ which suggests that values in variables are NULL. This problem is arranged in further investigation.
The base implementation of shallow water equations was provided by the supervisor G.D. Riley. It consists of two files, including shallow_base_openmp_v3.c and wtime.c.

To compile the file, the following command is used:

```plaintext
gcc -O2 -fopenmp -o shallowwater shallow_base_openmp_v3.c wtime.c -lm
```

Three problem sizes, 2D grids of size 32x32, 64x64 and 128x128, are run with threads number of 1,2,4,8 and 16. As ARM Cortex-A15 used there is a dual core CPU, so only two threads can be run simultaneously. Running more than 2 is not expected to improve performance. The Section 3.1.2 gives the performance of this implementation on ARM dual core CPU.

### 3.1.2 Performance

The run-time results are presented in Table 3.1.1. The time in the table is the average value of 5 runs in seconds.

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Number of Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>32x32</td>
<td>0.63</td>
</tr>
<tr>
<td>64x64</td>
<td>1.74</td>
</tr>
<tr>
<td>128x128</td>
<td>11.05</td>
</tr>
</tbody>
</table>

Table 3.1.1 Timings result of OpenMP naïve implementation on ARM CPU.

Performance on a range of parallel configuration is the main concern to judge a parallel program is successful or not. Performance using P processors is usually defined as: 1/Tp, where Tp is the time to execute the parallel program.

Figure 3.1.1 gives the performance of OpenMP implementation. X-axis in the figure is the number of threads vary from 1 to 16. Y-axis is the performance of the program.
As can be seen from the table, running in 2 threads gives the best performance. Performance of running in 1 thread is sequential program performance. Compared to 1 thread, 2 threads see a performance gain, but it is not doubled. Overheads can influence the execution behaviour, and investigation will be done to reduce the overheads. As expected in Section 3.3.1, as the number of threads used grows, the runtime start to increase.

This is a simple analysis for the timing results. A more detailed investigation of the parallel overheads involved is planned in Section 4.3.

### 3.2 OpenCL on Mali T-604 GPU

As Mali T-604 only consists of 4 shader units, the implementation running on Mali GPU is considered as running on 4 nodes in a HPC system. Following is an attempt to run an OpenCL benchmark targeted to NVIDIA Quadro 2000 GPU.

#### 3.2.1 Compile And Running

This base implementation is taken from [1] and was targeted to the NVIDIA Quadro 2000 GPU. However, the Mali GPU has a different architecture, so this
implementation on Mali may not achieve the best performance. The reasons for this are the subject of further work and has been arranged in Section 4.3.

It is mentioned in [29] that, to develop OpenCL programs for Mali GPUs, a platform with a Mali-T600 series GPU is required, along with an implementation of OpenCL for Mali-T600 series GPU. The implementation of OpenCL used on Arndale2 is Mali OpenCL SDK v1.1 released in February 2013.

The SDK has been installed in path ‘/scratch/Mali_OpenCL_SDK_v1.1.0’ on the Arndale2. The original Makefile of the OpenCL implantation was written for NVIDIA OpenCL SDK, so changes must be made to direct the compiler to new libraries and header files. The following listing 3.2.1 shows the new Makefile of the host program.

```
CC = g++
CFLAGS = -ggdb -Wall -O0 -fpermissive
ROOT = /scratch/Mali_OpenCL_SDK_v1.1.0
INCS = -I$(ROOT)/include -I$(ROOT)/common -I.
LFLAGS = -L$(ROOT)/lib -L$(ROOT)/common -lOpenCL -lCommon
shallow: shallow.c shallow.h opencl_common.o wtime.o
   $(CC) $(CFLAGS) $(INCS) -o shallow opencl_common.c wtime.c shallow.c
$(LFLAGS)
opencl_common.o: opencl_common.c opencl_common.h
   $(CC) -c $(CFLAGS) $(INCS) opencl_common.c
wtime.o:
   $(CC) -c $(CFLAGS) wtime.c
clean:
   rm *.o shallow
```

Listing 3.2.1 Makefile of OpenCL implementation on Mali GPU.
After compiling, the file ‘shallow’ is produced as the execution file of the host program. The next step is to run the host program to compile the kernel program and execute on Mali GPU.

However, it was found that when the kernel is initialising data, OpenCL returns an error, CL_OUT_OF_HOST_MEMORY. Then, execution of kernel is aborted. Section 3.2.2 provides a solution that was developed to solve this problem.

![Error Image]

Figure 3.2.1 An error returned in the OpenCL benchmark

### 3.2.2 Modification to Standard Code

The error is returned in function '__kernel void init1'. This function initialises data used in computation. The original code of the function is shown in Listing 3.2.2.

```
__kernel void init1(double a, double di, double dj, double pcf, __global double *p, __global double *psi){
  int x = get_global_id(0);
  int y = get_global_id(1);
  psi[y*M_LEN + x] = a * sin((y + 0.5) * di) * sin((x + 0.5) * dj);
  p[y*M_LEN + x] = pcf * (cos(2.0 * y * di) + cos(2.0 * x * dj)) + 50000.0;
}
```

Listing 3.2.2 Original code of function init1

According to the OpenCL Specification [26], CL_OUT_OF_HOST_MEMORY returns if there is a failure to allocate resources required by the OpenCL implementation on the host. However, in the host program, all variables used in this function have been correctly initialised and memory has been successfully allocated.
Many attempts were undertaken to try to fix this problem. The final solution is that 4 temporary “float” variables are defined and used to do computation inside sin and cos. The following 3.2.3 listing shows the changed code of function init1. However, if the 4 variables are defined as “double”, the error returns.

```c
__kernel void init1(double a, double di, double dj, double pcf,
        __global double *p,  __global double *psi){
    int x = get_global_id(0);
    int y = get_global_id(1);
    float yphdi;
    float xphdj;
    float tydi;
    float txdj;
    yphdi = (y+0.5) * di;
    xphdj = (x+0.5) * dj;
    tydi = 2.0 * y * di;
    txdj = 2.0 * x * dj;
    psi[y*M_LEN + x] = a * sin(yphdi) * sin(xphdj);
    p[y*M_LEN + x] = pcf * cos(tydi) + cos(txdj) + 50000.0;
}
```

Listing 3.2.3 Original code of function init1

After changing the code, the kernel can run properly. This suggest that the current Mali OpenCL SKD does not fully support mathematical intrinsic functions. Performance analysis of this shallow water benchmark is presented in Section 3.2.3.

However, as a result of changing variables from float to double, the precision of the variables is influenced. A comparison of the results of the original benchmark was done between the original OpenMP code and the modified
OpenCL code. The problem size compared is 32x32. Part of output from these runs is given in Listing 3.2.4.

<table>
<thead>
<tr>
<th>Original Code (OpenMP with Double)</th>
<th>Modified code (OpenCL with Float)</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagonal elements of p</td>
<td>diagonal elements of p</td>
</tr>
<tr>
<td>50001.915177 50001.758802 50001.334333 50000.705612</td>
<td>50001.950261 50001.791141 50001.358981 50000.718848</td>
</tr>
<tr>
<td>diagonal elements of u</td>
<td>diagonal elements of u</td>
</tr>
<tr>
<td>-0.188904 -0.526145 -0.768654 -0.879514 -0.841853 -0.661407</td>
<td>-0.188904 -0.526145 -0.768656 -0.879515 -0.841853 -0.661408</td>
</tr>
<tr>
<td>diagonal elements of v</td>
<td>diagonal elements of v</td>
</tr>
<tr>
<td>0.188062 0.525375 0.768086 0.879244 0.841927 0.661812 0.366312</td>
<td>0.188054 0.525382 0.768108 0.879278 0.841966 0.661852 0.366347</td>
</tr>
</tbody>
</table>

Listing 3.2.4 A comparison of computation results

The difference in the results is at the second or third position after radix point. This suggests that the change in precision made to the initial data used by the benchmark does not lead to large errors in the benchmark results. This difference is considered acceptable in the shallow water equations computation.

### 3.2.3 Performance

The raw timing results of OpenCL implementation are presented in Table 3.2.2. A comparison is made between OpenMP and OpenCL implementation in Figure 3.2.2. To ensure fair comparisons in the future work, the change to
initial data required to allow the benchmark to run on the Mali GPU has also will be implemented in the OpenMP versions.

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x32</td>
<td>2.401004</td>
</tr>
<tr>
<td>64x64</td>
<td>4.840258</td>
</tr>
<tr>
<td>128x128</td>
<td>12.424474</td>
</tr>
</tbody>
</table>

Table 3.2.1 Performance timings of OpenCL implementation on Mali GPU.

For OpenMP implementation, the performance of running in 1 and 2 threads are given in the figure. 1 thread is the sequential performance of the benchmark and 2 threads achieve the best performance compared with other threads of numbers.

![Performance of OpenCL and OpenMP Implementations](image)

Figure 3.2.2 Performance of OpenCL code on GPU and OpenMP on CPU

Compared with the OpenMP implementation on the ARM CPU, the OpenCL version has a worse performance in all problem sizes. The performance gap is significant especially in small problem sizes. These performance difference will be investigated in the next phase of the project.
3.3 Comparison With Other Architectures

This section compares the OpenCL implementation on Mali GPU with it running on other accelerators. As mentioned in Section 3.2, this OpenCL code is implemented to NVIDIA Quadro 2000[1] with optimisations, so original performance of the code is involved in comparison.

Currently, another MSc student is doing a project with the shallow water benchmark targeted on other accelerators including NVIDIA Tesla K20m and Intel Xeon Phi 5110p. NVIDIA Tesla K20m is a powerful GPU with 2496 shading units and 706MHz GPU clock speed. Intel Xeon Phi Coprocessor 5110P contains 60 cores with 1.053 GHz clock speed.

The same implementation has been run on these two devices. The performance is shown in Table 3.4.1. As ARM architecture is targeted in small problem size, so only performance in problem size 128x128 is compared.

<table>
<thead>
<tr>
<th>Time (Seconds)</th>
<th>Mali T-604</th>
<th>Xeon Phi</th>
<th>Quadro 2000</th>
<th>Tesla K20m</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>12.42</td>
<td>27.26</td>
<td>0.73</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table 3.4.1 Performance of OpenCL implementation on Mali and other devices.

As shown in the table, the implementation running on original targeted device, NVIDIA Quadro 2000, has a very short run time. NVIDIA Tesla K20m is made by the same vendor as NVIDIA Quadro 2000. It even has a better performance.

However, performance on Mali and Xeon Phi is not as good as NVIDIA GPUs. As introduced in Section 2.2.3, although OpenCL program is portable, it cannot achieve peak performance on different devices by various vendors. This implementation is not targeted to Mali or Xeon Phi, so the worse performance
is expected. Further optimisation should be done to these hardware to have a better performance.

For Mali GPU, it is designed for mobile devices. The lower performance may be also related to its hardware architecture.

3.4 Summary

This chapter describes the progress of the project. An OpenMP implementation has been run on ARM Cortex-A15 and performance is given. An OpenCL implementation targeted on NVIDIA Quadro 2000 has been run on Mali GPU. Through the process, an error was found and had been fix. Finally, the performance on Mali GPU is compared with the same implementation on other devices.
Chapter 4
Project Plan

Section 4.1 summaries the methodology that will be used in the project to parallelise the benchmarks and investigate performance and power energy use. Section 4.2 presents the milestones achieved. Based on achieved milestones, modified plan of the project is introduced and deliverables of the project is also analysed.

4.1 Methodology

The implementation and optimisation methodology that will be used in the project is summarised as the technique of overhead analysis [35]. The naïve implementation of shallow water equations have been provided, the focus of the project is to analyse execution behaviour and then to optimise the code.

The execution behaviour is associated with algorithm and machine-related costs which reflects in the run-time as an idealised parallel execution time plus a set of parallel overheads. The optimisation of the code is to minimise significant overheads based on specific hardware architecture. The process of analysis is known as performance modelling which is introduced in detail in [35].

For performance result analysis, every execution is run 5 times and the average time is calculated. This is based on the method introduced in the COMP60611 course laboratory one. The time is measured by adding timers in programs.

For power consumption, the whole system’s power consumption is calculated based on input voltage and input current. The input voltage for the Arndale board is constant, and input current is measured by a device in the power plug of the board. The detail of the device is introduced in Section 2.4.
4.2 Milestones

The milestones achieved so far are listed in the following:

- Background of the project including hardware architecture, APIs and power efficiency for the project have been studied.
- A naïve OpenMP implementation has been successfully run on the ARM Cortex-A15 CPU.
- A naïve OpenCL implementation has been successfully run on the Mali GPU.
- A problem when running OpenCL implementation on Mali GPU was found. A modification has been made to the standard code and the impact of this change on the benchmark result has been checked.
- The OpenCL implementation has been successfully running on Mali GPU.
- Run-time results of two implementations has been collected. A initial comparison of performance has been made. These results can be used as a basis to analyse the execution behaviour and optimise code.

4.3 Modified Plan And Deliverables

Based on the milestones achieved, a modified project plan is presented in Figure 4.3.1.

As the methodology to optimise the parallel benchmark of the shallow water equations has been summarised, the following task is to use the method to optimise OpenMP and OpenCL code.

After that, time is allocated to implement the shallow water benchmark on heterogeneous ARM architecture with part of code running on CPU and part on GPU. According to the optimisation method, the execution behaviour should be analysed to support optimisation. Run-time data analysis and
evaluation will be undertaken during optimisation phase. When optimisation is finished, more time is needed to evaluate data and puts results in MSc dissertation.

![Project Plan Timeline](image)

Figure 4.3.1 The modified project plan.

Power measurement and power efficiency analysis is arranged after the start of implementation OpenCL on ARM heterogeneous architecture.

Writing the dissertation will take place through the whole process of the MSc project. The process and results of optimisation and implementation developments can be put into the dissertation as the work progresses.

The timetable of the project is properly arranged, and all equipment for the project is available. Methods to optimise implements, undertake data and power consumption analysis have been investigated and will be used in the project. So the project is considered to be achievable.
Chapter 5
Conclusion

This report contains the background information that support the project. A wide range of research topics related to the project are introduced and summarised including shallow water equations, concurrent programming languages and APIs, hardware architectures, and power efficiency in HPC filed.

The progress of the project achieved so far is also described. Two shallow water benchmarks have been run on heterogeneous ARM architecture and an initial analysis to performance have been given.

Based on the milestones achieved, the project plan has been modified. The project is considered to be achievable.
Bibliography


[22] Samsung, Samsung Exynos 5 Dual processor. Available at: 
http://www.samsung.com/global/business/semiconductor/minisite/Exynos/data/Exynos_5_DUAL.pdf [Accessed 05/05/2014]

[23] Samsung, Samsung Exynos 5 Dual (Exynos 5250) RISC Microprocessor, October 2012. Available at:
http://www.samsung.com/global/business/semiconductor/file/product/Exynos_5_Dual_User_Manual_Public_REV100-0.pdf [Accessed 05/05/2014]


[27] The OpenACC Application Programming version Interface 1.0, November 2011. Available at :
http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf [Accessed 05/05/2014]


[34] OpenCL Conformant Products. Available at: http://www.khronos.org/conformance/adopters/conformant-products/

[Accessed 06/05/2014]