Building and benchmarking a low power ARM cluster

Nikilesh Balakrishnan

August 24, 2012

Abstract

Mobile and hand-held devices are on the rise in the consumer market. The future of innovation in the semiconductor industry will be in ubiquitous computing. HPC is undergoing a paradigm shift towards low power architectures in order to meet the power requirements to build an Exascale system. GPGPUs and specialized architectures such as IBM BG/Q have already started making a mark in achieving high performance while consuming lesser energy. The HPC community is also looking at using mobile and embedded devices since the hardware architecture for these devices have been designed for low power consumption. ARM architecture processors dominate the mobile and embedded market and is a viable candidate to build a HPC system.

In this project we built a low power cluster consisting of 6 Pandaboard ES boards. We also configured 2 Raspberry Pi boards running different versions of Debian Linux. The goal was to run several benchmarks on the Pandaboard and the Raspberry Pi boards and analyse the raw performance obtained as well as the power consumed.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank my supervisor Mr. James Perry for the support and guidance provided throughout the project. I also thank Mr. Sean McGeever and the helpdesk team for all the support provided with hardware requirements.

1 Introduction

The large systems used today in HPC are dominated by processors that use the x86 and Power instruction sets supplied by three vendors, Intel, AMD and IBM. These processors have been designed to mainly cater to the server, desktop PC and laptop market. The processors provide very good single thread performance but suffer from high cost and power usage. One of the main goals in building an exascale HPC system is to stay within a power budget of around 20 MW. In order to achieve this ambitious goal, other low power processor architectures such as ARM are currently being explored since these processors have been primarily designed for the mobile and embedded devices market [1].

There is a growing trend towards using accelerators such as GPGPUs in order to build energy efficient HPC machines. GPUs offer greater FLOPS/watt than conventional CPUs [2]. However it can be a challenge to program these devices as developers are required to learn new programming models in order to utilise them effectively. This is further complicated by the need for vendor specific compilers such as Nvidia's CUDA compiler to program Nvidia GPUs in order to extract the last drop of performance.

Today we see a surge in special custom built machines like the IBM BlueGene/Q which have been designed for low power consumption while giving high performance. Top 20 machines in the Green500 as well as the Top500 lists are currently dominated by the IBM BlueGene/Q systems [3] [4].

Post-PC devices such as tablets, PDAs, smartphones and other mobile devices are gaining in popularity in the consumer space. This trend is predicted to grow as companies such as Apple, Google and Microsoft begin investing heavily in future products aimed at the mobile devices market. ARM processors currently dominate this space and the increase in demand for these devices will bring down the cost of manufacture of ARM processors even further.

ARM processors have also started to make an entry into the PC and server market. Future ARM processors are expected to be much more powerful in terms of performance while still maintaining low power usage. For the HPC market, this scenario presents a great opportunity to utilize ARM processors to build large HPC machines. We already see projects such as Mont-Blanc at the Barcelona supercomputing centre which plan to use ARM CPUs combined with Nvidia GPUs to achieve petascale compute performance.

A variety of single board computers such as the Beagleboard, Raspberry Pi and the Pandaboard have become popular among developers and programming enthusiasts as they are cheaper than desktop PCs and laptops. These boards use a variety of system on chips (SoCs) with various ARM architecture CPUs. This has led to the formation of various ARM development communities who spend time and effort porting a wide range of operating systems, applications, libraries and tools to the various ARM architecture currently popular in the market.

In this project we will explore the challenges of building a cluster using these single board ARM computers. We will also measure the performance of the cluster using various benchmarks while capturing the power usage. We are already aware of the performance and power usage of clusters built using commodity processors such as x86. This project will give us further insight into the progress made by ARM processors in addressing the issue of performance and power usage which is vital to HPC.

1.1 Report Organization

Chapter 2 in this report provides a background for the project. In this chapter we look at some of the factors causing a growth in interest in low power HPC. We look at some of the main trends in hardware in the recent past and how this trend is progressing as we move towards the Exascale era. We also look at the road map for ARM processors in the next few years and how ARM could be a major player in the high performance computing market.

Chapter 3 provides details about the ARM hardware used in this project. We examine the components present in the Pandaboard ES and Raspberry Pi boards. Aspects such as CPU features, memory hierarchy and inter-connect are discussed. Both these boards use different generations of the ARM architecture hence we can compare and contrast their capabilities.

In Chapter 4 we look at some of the existing energy-efficient HPC systems and also low power clusters built using ARM processors. Interest in low power ARM clusters is a very recent trend and is growing rapidly especially with advances being made in ARM processor designs.

Chapter 5 provides details regarding the steps taken to build the Pandaboard ES cluster. Here we will explore the hardware and software challenges encountered and their respective solutions.

In Chapter 6 and 7 we look at the various benchmarks that were identified to be run on the Pandaboard ES and the Raspberry Pi boards. We also analyse the performance and power consumption of these boards. There are 6 dual-core Pandaboards available and hence we look to scale some of the benchmarks over 12 cores and provide a detailed analysis of the results.

In Chapter 8 and 9 we provide possible improvements in future projects involving ARM processors and also provide conclusions obtained from undertaking this project.

2 Background

HPC has seen an exponential growth in computational power over the last couple of decades. This has been mainly due to the evolution of microprocessor technology. Until around 2003, chip manufacturers improved the serial performance of their processors by adding more and more performance optimization technology in hardware such as branch prediction, speculative execution, increase of on chip cache sizes etc. The clock frequency of the chips also kept getting faster while the down side was an increase in power consumption. However due the power wall problem, manufactures began adding multiple cores on a single chip in order to prevent the CPU from melting due to overheating [7]. Today, multi-core machines are the norm and we expect the number of cores within a node to increase in future.

In this chapter we will look at the issue of power consumption of HPC machines and also explore some of the emerging technologies in hardware to address the problem of power consumption.

2.1 Exascale Challenge

In 2007, DARPA conducted a study in order to identify the challenges involved in building an exascale HPC machine. The study identified several challenges such as power consumption, hardware & software resilience, memory bandwidth, storage of peta-bytes of data and application scalability [6]. The main challenge among these is the power consumption of an exascale machine. The energy budget fixed by DARPA is around $20 million USD per year. The power budget of an exascale system can be estimated to be around 20 MW since a mega watt of power costs approximately $1 million USD.

HPC is currently in the petascale era and the fastest supercomputer today performs at approximately 16 petaflops [3]. If we are to follow the growth trend of supercomputing power, it is estimated that we will reach exascale computing capabilities by 2018. This is shown in Figure 2.1.

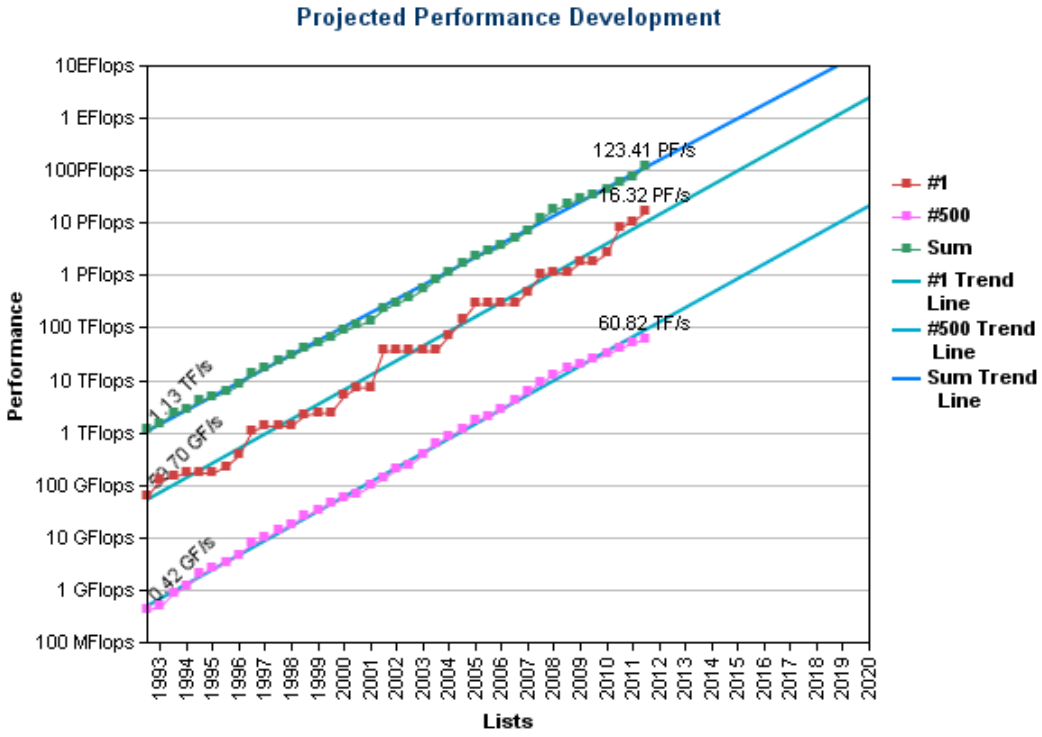

Figure 2.1: Timeframe to reach exascale (Source: Top500.org)

From a chip manufacturers perspective, HPC is only 1% of the total market for their products. The research and innovations done by these companies are targeted at the remaining 99% of the market which is dominated by products catering to the PC and laptop consumer market. With mobile and embedded devices gaining popularity, the major chip manufacturers such as Intel and AMD have started investing in low power chip designs. An increase in quantity of low power processors will bring down the per unit cost.

Today we see a variety of low power technologies being used to build large HPC machines. The trend is towards using low power CPUs, accelerators, coprocessors or a combination of these technologies within each compute node. In the section below we will focus on some of these emerging technologies and their impact on current and future HPC systems.

2.2 Low power hardware trends

2.2.1 GPU

Graphics processing units were originally developed for the gaming industry. GPUs are made of a large number of very simple cores focussed purely on performing computations. Due to the high level of inherent parallelism present in the processor, the HPC community became interested in utilizing the GPU to achieve parallelism. However, many of the initial GPU designs did not have a double precision floating point unit and those that did have one did not perform up to the expectation of the HPC development community [8]. Over the last 3 years, GPU designs have undergone major changes and have become a much more acceptable part of the HPC landscape.

The two big players in GPU technology currently are Nvidia and AMD with Nvidia being the more popular choice in HPC. The selling point of GPUs is the performance/watt that they promise. The Nvidia Fermi GPU performs at approximately 3 gigaflops/watt [9]. Future Nvidia GPUs are expected to provide even better performance/watt. Similarly AMD has invested in an integrated CPU-GPU design called AMD Fusion [10]. One of the bottleneck in extracting performance from GPUs today is the PCIe bus through which the CPU offloads computation intensive work to the GPU. Future designs from both AMD and Nvidia are looking to address this problem by integrating both the CPU and GPU into a single die.

2.2.2 Many Integrated Cores (MIC)

In 2010, Intel announced its new multiprocessor architecture targeting the HPC market based on the Larrabee many core architecture [11]. This family of processors was later named as the Xeon Phi at the ISC 2012 in Hamburg. The Xeon Phi has been marketed as a coprocessor rather than as an accelerator. Figure 2.1 is an image of the Xeon Phi.

The main goal of the Xeon Phi family of coprocessors is to achieve one teraflop or more of double precision floating point performance for under 100 W of power [12]. This is achieved by using many simple 64 bit x86 architecture cores with wide SIMD sharing memory. Since the architecture of the processor is x86 based, the programming model used can be the same as that used in programming shared memory CPUs. Hence programming models like OpenMP and MPI which are most prevalent in HPC application codes can be ported to the MIC without rewriting them. This is a major advantage for the Xeon Phi family of coprocessors over an accelerator such as Nvidia GPUs.

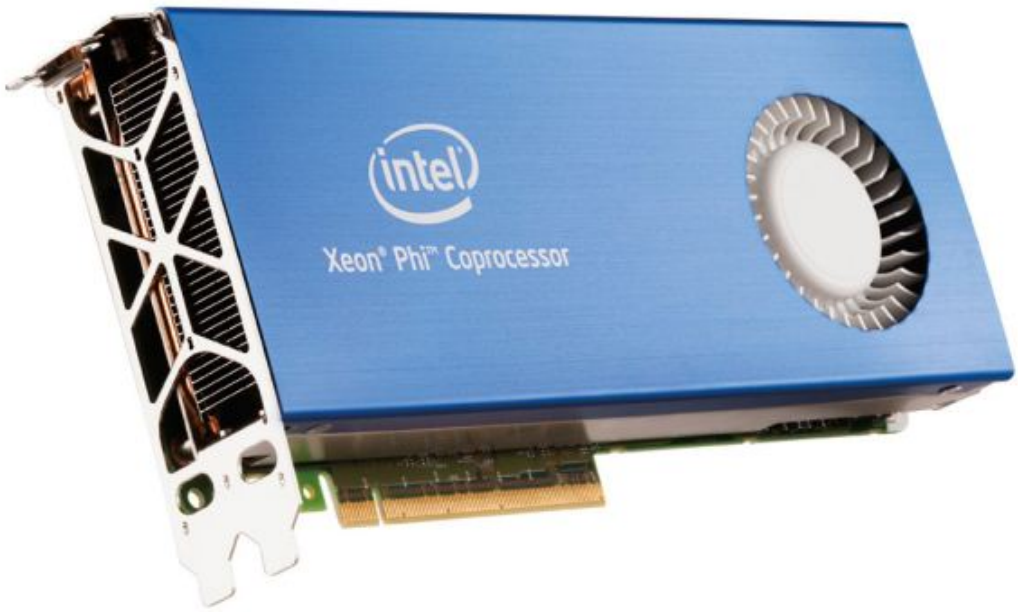

Figure 2.1: Image of Intel Xeon Phi coprocessor (Source: www.anandtech.com)

The MIC coprocessor can be used both in offload and native modes unlike current GPU designs which can only be used in offload mode. With the native mode we can obtain higher performance, however the bottleneck could be memory access on the MIC card. The MIC coprocessors come with at least 8 GB GDDR5 memory and the width of the memory bus could be 256-bits or 512-bits [13].

2.2.3 ARM processors

The ARM instruction set, developed by ARM Holdings is an implementation of the reduced instruction set computer (RISC) ISA. The ARM architecture is licensable and there are a number of companies such as Texas Instruments, Nvidia, Samsung, Broadcom etc., that are licensees. The primary consumers of the ARM processors have been manufacturers of mobile and embedded devices. ARM processors have captured over 95% of the smart phone market and 10% of the mobile computer market. It is predicted that ARM processors will capture approximately 23% of the PC market share by 2015 [14].

ARM processors have not yet made an impact in the high performance server market mainly because the processors use a 32-bit instruction set architecture. This is changing quickly with the new ARMv8 specification which will have support for 64-bit operations along side the conventional 32-bit execution [15]. Companies like Calexeda have also recently launched a low power SoC called Calxeda EnergyCore which uses 4 quad-core ARM Cortex-A9 processors, an integrated 80-gigabit crossbar switch and an integrated management engine all on a single piece of silicon [16]. The ARM Cotex-A9 processor is based on the 32-bit ARMv7 architecture and hence may not be suitable for large HPC application that use a lot of double precision floating point calculations. However it can be used for BigData analytic applications, web-servers, cloud computing etc. We can see future EnergyCore products using ARMv8 architecture and that can be used to build a 64-bit energy efficient HPC system.

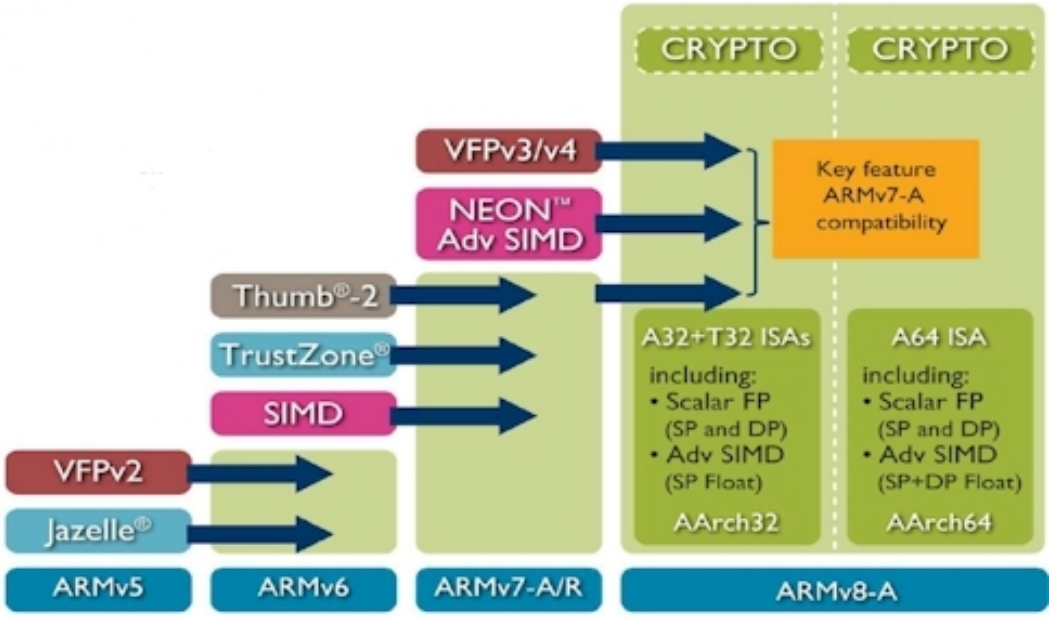

Figure 2.3: The ARM RISC processor roadmap for applications (Source: Richard Grisenthwaite, 2011 ARM TechCon)

The support for 64-bit operations in the hardware is crucial for ARM to be considered seriously in the HPC market. As shown in Figure 2.3 the ARMv8 architecture offers this capability along with efficient energy usage. The ARMv8 also offers double precision NEON SIMD capability in addition to the existing single precision NEON SIMD available in ARMv7 processors.

Nvidia has already started working on a 64-bit ARM architecture CPU called “Project Denver” specifically targeting the HPC and server industry [17]. One of the main design feature with this initiative is the integration of an 8-core custom Nvidia 64-bit ARM CPU and an Nvidia GeForce 600-class GPU on the same chip. GPU computing has gained a lot of interest among the HPC community due to its superior FLOPS/watt performance. However the bottleneck of copying data from the CPU to the GPU and back through the PCIe bus has always been a detriment to gaining good performance. By combining the CPU and GPU on a single chip we can obtain very high bandwidth while keeping the latency extremely low.

2.2.4 DSP

A new paradigm in HPC is beginning to emerge with the use of Digital Signal Processors (DSPs) to perform compute intensive operations. The multi-core DSPs offered by Texas Instruments deliver greater than 500 GFLOPS of compute performance while using only 50 W of power [18]. These DSPs deliver a very impressive FLOPS/watt ratio and can be used as add on cards through the PCIe. The newer model full length cards are expected to give over a teraflop of performance. The multi-core DSPs support programming languages such as C and OpenMP. Texas Instruments also offers a multi-core SDK along with optimizing libraries for scientific computing so that developers can quickly program the DSPs and get maximum performance out of them.

3 Project Hardware

In this chapter we look at the high level specifications for 2 single board ARM processor computers, the Pandaboard ES and the Raspberry Pi used in this project. Both systems use ARM processors from different generations and have differing processing power and capabilities.

3.1 Pandaboard ES

The Pandaboard ES is a low power, low cost single board computer that uses the OMAP4460 SoC. The OMAP4460 SoC contains a dual-core 1.2 Ghz ARM-Cortex A9 MPCore CPU, a PowerVR SGX 540 GPU and a C64x DSP subsystem [19]. In this project we will focus on the performance benchmarking of the ARM CPU.

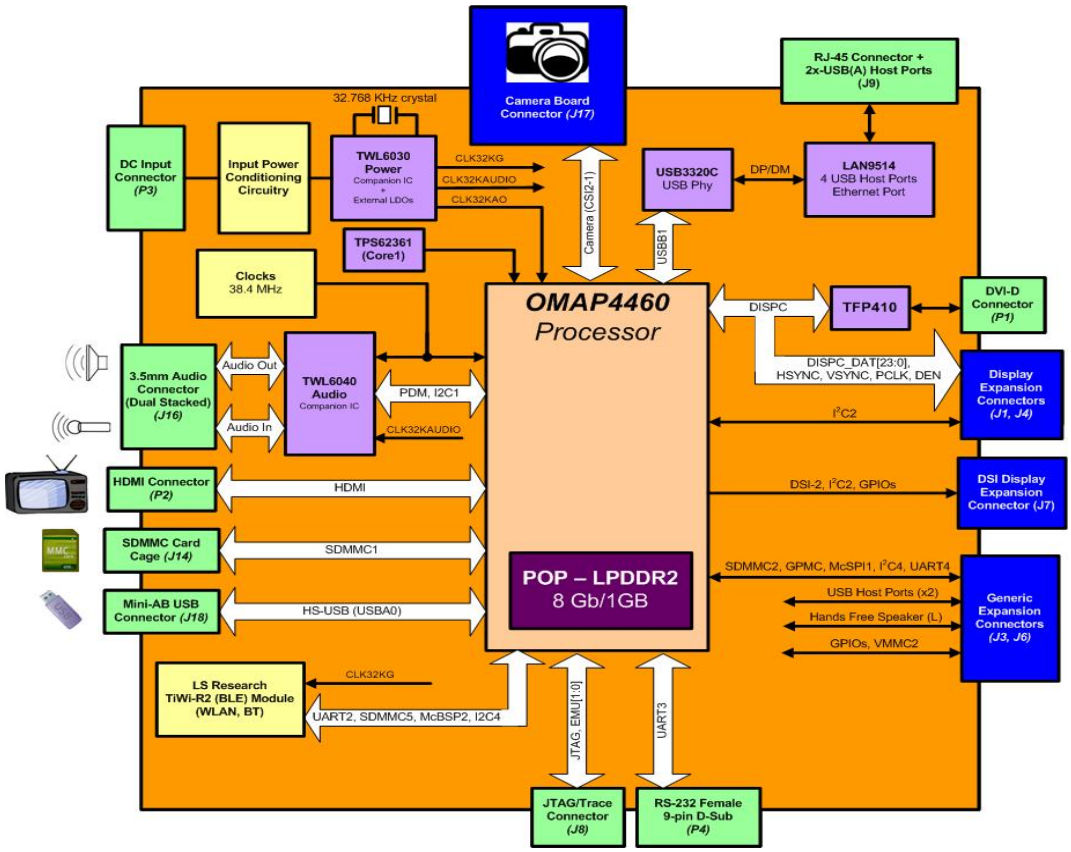

Figure 3.1: Pandaboard ES block diagram (Source: www.pandaboard.org)

The 2 ARM cores share a 1 MB L2 cache and the RAM available on the board is 1 GB dual channel LPDDR2 (Low Power DDR2). The Pandaboard also has an 10/100 Mbps Ethernet port. This gives us the capability to cluster multiple boards on a network and run parallel applications and benchmarks. The OMAP4460 was originally designed for mobile devices and hence we see support for audio and video. There board also provides general purpose expansion headers, camera expansion headers and LCD signal expansion capabilities.

There are 2 USB 2.0 high-speed ports on the board through which we can connect a keyboard, mice, SSD or any external device having USB support. The Operating System can be loaded on an external SSD drive connected via USB or through the on-board slot for high speed SD card.

The Pandaboard ES is an improved version of its predecessor the Pandaboard which uses an older OMAP4430 SoC. The OMAP4430 ARM CPU has a lower clock speed (1 Ghz) when compared to the OMAP4460 ARM CPU.

3.1.1 ARM Cortex-A9

The ARM Cortex-A9 is a 32-bit multi-core processor which implements the ARMv7 instruction set architecture. The cortex-A9 can have a maximum of 4 cache-coherent cores and clock frequency ranging from 800 to 2000 Mhz [20]. Each core in the cortex-A9 CPU has a 32 KB instruction and a 32 KB data cache.

One of the key features of the ARM Cortex-A series processors is the option of having Advanced SIMD (NEON) extensions. NEON is a 128-bit SIMD instruction set that accelerates applications such as multimedia, signal processing, video encode/decode, gaming, image processing etc. The features of NEON include separate register files, independent execution hardware and a comprehensive instruction set. It supports 8, 16, 32 and 64 bit integer as well as single precision 32-bit floating point SIMD operations [21]. Support for 64-bit double precision floating point SIMD operations is part of the ARMv8 NEON extension.

The cortex-A9 also has hardware support for half, single and double precision floating point arithmetic. The floating point unit on the ARM is called Vector Floating Point (VFP). The cortex-A9 uses the VFPv3 version for its floating point calculation. It should be noted that although the VFP unit is intended to support vector mode, the VFP instructions work on vector elements sequentially and hence does not offer true (SIMD) vector parallelism [22].

3.2 Raspberry Pi

The Raspberry Pi is a cheap, low power, credit card size single board computer that was designed in order to create interest in programming among school children. The Raspberry Pi Foundation is responsible for developing this device.

The Raspberry Pi has a Broadcom BCM2835 SoC which in turn comprises of a 700 Mhz ARM11 CPU and a VideoCode IV GPU [23]. The Raspberry Pi comes in 2 models A and B. The difference in functionality between the two models is that model B has an ethernet port and 2 USB ports while model A has no ethernet port and only 1 USB port.

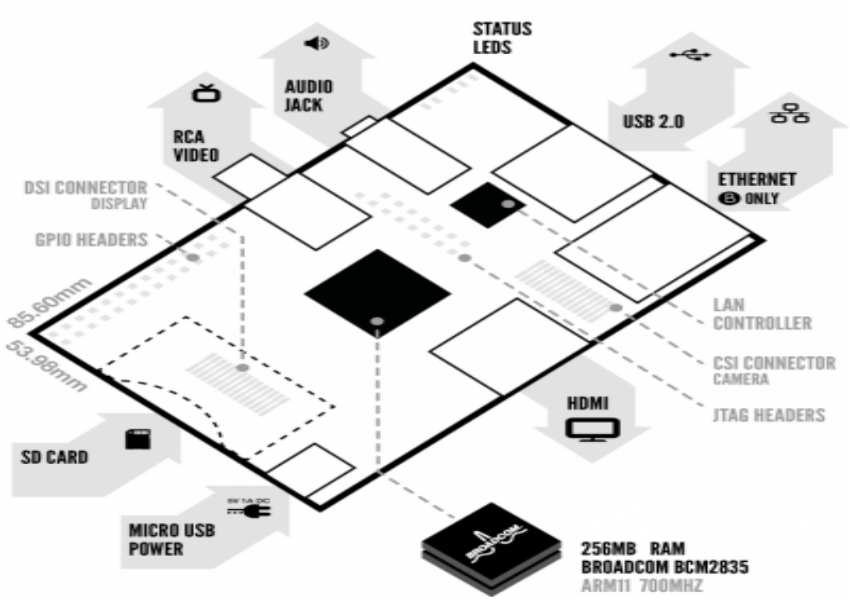

Figure 3.2: Raspberry Pi Board layout (Source: www.raspberrypi.org)

The Raspberry Pi has a 256 MB POP SDRAM shared between the CPU and GPU in any of 128/128, 192/64, 224/32 MB CPU/GPU ratio. The L1 cache is 32KB for the instruction and data. The 128KB L2 cache by default is reserved for use by the GPU. However it can be explicitly enabled for use by the CPU. This is done by the Raspbian OS which is based on Debian armhf and is optimized for the Raspberry Pi.

The board provides HDMI video and audio outputs. It also provides a SD card slot on which the operating system is loaded. There is a FAT32 partition on the SD card. The SD card also contains a kernel image along with GPU firmware. It also has a EXT2 partition with the rootfs. The system boots via the GPU using a sequence of steps and starts the ARM CPU.

3.2.1 ARM11

ARM11 is a 32-bit RISC microprocessor that implements the ARMv6 instruction set architecture. The ARM11 processor family is used to power smart phones, home and embedded applications. The clock speeds of these processors can range from 350 Mhz to 1Ghz and are extremely energy efficient.

The ARM11 CPU has a VFP unit which implements the VFPv2 vector floating point architecture [24]. The VFP unit provides low-cost, IEEE 754 compliant floating point computation support which is essential for multimedia, graphics and other floating-point intensive applications. Using the right compiler flags and support from the operating systems we can get maximum performance from the floating point unit.

4 Literature Review

In this chapter we will look at the design details of the most energy efficient HPC machine in the world today, the IBM Blue Gene/Q. We will also explore some low power clusters built using ARM processors. Understanding the design strategies used and problems encountered in building clusters using ARM hardware is of particular interest for this project.

4.1 Energy efficient HPC systems

We see a variety of strategies being adopted in HPC to address the problem of power consumption as we transition through the petascale era and approach exascale age. Some of these strategies involve special designs in hardware, new heterogeneous approaches to computing and innovative ways to design cooling systems for large HPC machines.

4.1.1 IBM Blue Gene/Q

The Blue Gene/Q (BGQ) is the latest design in the Blue Gene series of HPC machines. The BGQ design focusses on building massively parallel supercomputers which provide the highest FLOPS/watt ratio [26]. The BGQ is designed to handle computation intensive as well as data intensive HPC applications. This is evident from the rankings of the BGQ machines in the Top500, Green500 and the Grap500 lists released recently. There are 4 BGQ machines among the top ten machines in the Top500 list released in June 2012 [3]. The top 20 machines in the Green500 lists released in June 2012 are all BGQ machines [4]. The Graph500 benchmarks solves a large graph problem and is a good indicator of the performance for data intensive HPC applications. There are 5 machines in the top ten machines of the Graph500 list [25].

4.1.1.1 BGQ Desgin

The BGQ compute chip is a SoC that integrates processing cores, memory and neworking login onto a single chip. The core on the SoC use a four-way hyperthreaded, 64-bit PowerPC A2 chip technology [26]. The chip resembles a large shared memory processor (SMP) machine.

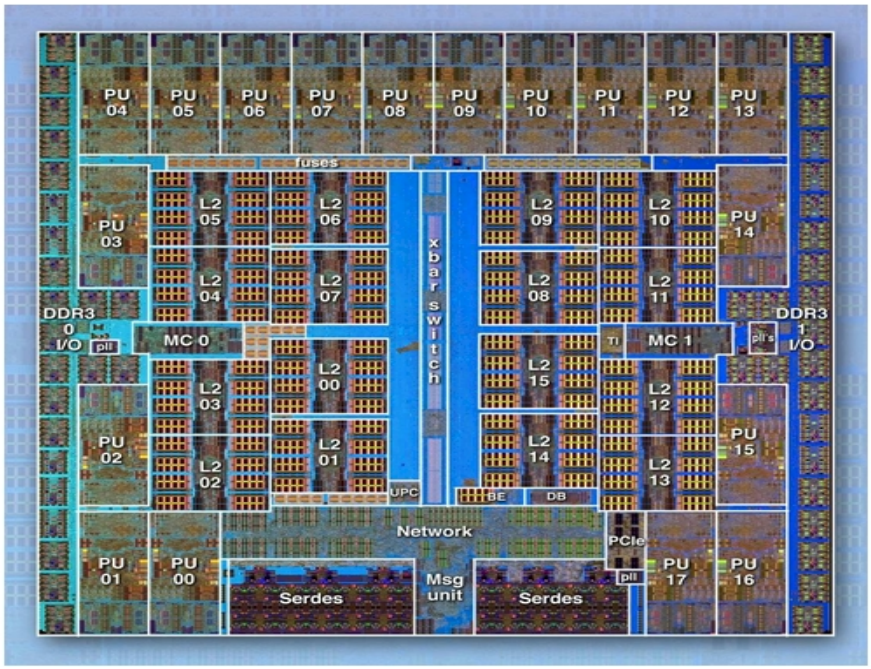

Figure 4.1: Blue Gene/Q Computer Chip (Source: Rudd Haring / IBM Blue Gene Team)

The BGQ compute chip consists of 16 user cores used to perform computations plus one service core to handle OS tasks such as asynchronous I/O, interrupts and scheduling, messaging assists and RAS [26]. A dedicated core for the OS services reduces the OS noise and jitter on the 16 compute cores. There is also an 18th core which is a redundant spare in case one of the cores becomes damaged during manufacturing.

Each core in the BGQ chip has a clock frequency of 1.6 Ghz and the total power consumed by the chip is approximately 55 W at peak load giving a peak performance of 200 GFLOPS. This yields a performance/watt value of 3.6 GFLOPS/Watt per chip. To further reduce power consumption, the chip makes extensive use of clock gating.

Each processor core has a SIMD Quad-Vector double precision floating point unit. Each processor core also has a L1 cache with 16KB for instruction and 16 KB for the data. The L2 cache is 32 MB and 16-way set associative and the main memory is 16GB in size [26]. The BGQ chip also implements transactional memory and gives IBM the distinction of becoming the first company to deliver commercial chips with this technology [27].

In a BGQ compute rack there can be up to 16,384 cores and 16 TB of memory. Each of these racks is water cooled and connect to a 5D network torus [28].

These specialized node design, cooling techniques and interconnect make the BGQ a very high performing energy efficient machine. However the cost of building such a machine is huge. Also the power usage of current BGQ systems is still not within the 20 MW limit when scaled to an exascale HPC machine. In order to reach exascale with a reasonable power budget, there needs to be changes in all layers of the hardware and software stack.

4.2 ARM clusters

A couple of decades ago commodity PC microprocessors began entering the HPC market as people looked for cheaper options to build massively parallel computers with thousands of cores. This was dubbed the age of the micro-killers where commodity processors were cheaper as there was a huge demand for PC processors. This made the cost of manufacturing of microprocessors cheap. We eventually saw the decline of vector processors. We see this trend today with processors for the mobile market. As the demand for mobile devices increases, the cost of manufacture of mobile processors will come down. Also processors used in mobile devices take much lesser power compared to processors used in PCs and servers. We may be entering the age of the mobile-killers. We are already beginning to see an interest in using mobile processors such as ARM processors to build clusters. In the subsections below we will look at a few of these projects.

4.2.1 Mont-Blanc BSC

The Mont-blanc project is an European exascale approach to build a low power exascale machine using embedded and mobile processors. This project has a budget of over 14 million Euros and is managed by the Barcelona Supercomputing Center (BSC) [29].

The objectives of the Mont-Blanc project is to build a 50 petaflop machine that uses approximately 7 MW by 2014 and then to design and build a 200 petaflop machine using 10 MW of power using the latest mobile processor designs at that time.

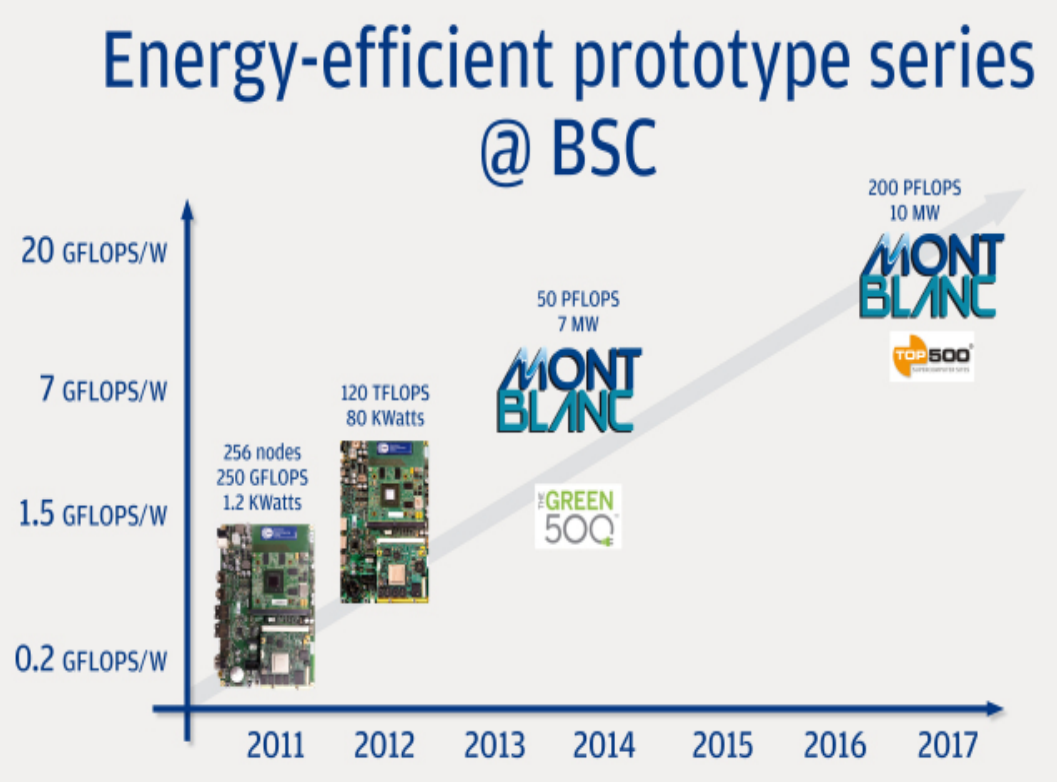

Figure 4.2: Energy-efficient prototypes roadmap (Source: BSC)

There are 2 hardware design approaches that have been put forward. One is to use a homogeneous multi-core ARM processors. The other is to use ARM CPUs combined with a discrete or integrated GPU. In the homogeneous approach, although ARM processors are extremely energy-efficient, it is important to have high multi-core density to ensure that the CPU becomes the main power sink. The rest of the power is “glue power” and goes to components such as memory, interconnect and storage. Currently ARM processor designs available in the market

have a maximum of 4 cores. However future designs will support more cores on the same chip as mobile application demands increase and ARM starts making inroads into other markets such as servers and desktops [30].

The other approach is to use an ARM CPU and a GPU accelerator. The offloading of work to the GPU could be through the PCIe or with newer designs the GPU can share memory with the CPU. This design will ensure that there is no power wasted on the PCIe bus and GDDR5 memory [30].

The interconnect used between nodes for communication is either an SoC integrated Ethernet or an External NIC through PCIe, LLI or USB 3.0 [30]. In order to hide communication latency, the programming model must ensure that computations and communications are overlapped. The BSC has developed the OmpSs programming model along with an intelligent runtime system that can deal with heterogeneity [31]. The OmpSs runtime automatically overlaps MPI message communication with computation [30].

4.2.2 Phoronix Pandaboard cluster

In June 2012, Phoronix an internet media company built a 12 core ARM cluster using 6 Pandaboard ES development boards [32]. The operating system and GCC compiler used were Ubuntu 12.04 and GCC 4.6 respectively. The main objective of was to run various benchmarks on the cluster and measure the performance/watt of the system. The results were then compared against the Intel Atom, Ivy Bridge and AMD Fusion processors.

Three programs, discrete 3D Fast Fourier Transform, Embarrassingly Parallel and Lower-Upper Gauss-Seidel Solver from the NAS parallel benchmark suite were run on the cluster. Under peak load, the ARM cluster consumed approximately 30 W of power and when idle the power consumption was below 20 W [32]. The performance/watt measurements of the Pandaboard cluster was far superior to that of the Intel Atom while the Ivy Bridge outperformed the Pandaboard cluster both in terms of raw performance and performance/watt. The cost of purchasing an Ivy Bridge Intel Core i7 3770 is less than the cost of 6 Pandaboards. The AMD Fusion

E-350 APU was better in terms of raw performance however the Pandaboard delivered a better performance/watt score [32].

By end of June 2012 a team from MIT with the help of Phoronix assembled a 96 core ARM cluster using 48 Pandaboards [34]. The cluster used a solar panel to supply power to the nodes. Unlike the 12-core Pandaboard ES cluster, this cluster from MIT used the older OMAP4430 SoC which is part of the first version of Pandaboard. The SoC has a 1 Ghz ARM Cortex-A9 CPU along with a PowerVR GPU. The boards were stacked vertically in order to save space and increase density. The idle power consumption of the cluster was approximately 170 Watts and during peak load the power consumption was just over 200 Watts. The Ubunutu 12.04 port for OMAP4 was installed on all the boards. The MIT team is yet to publish the benchmark results for this cluster.

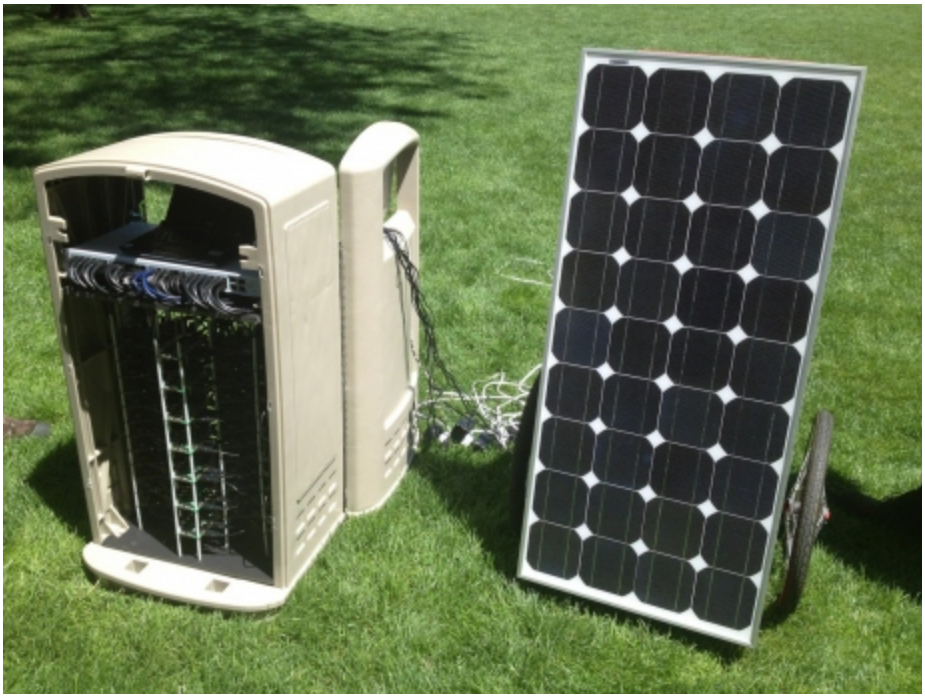

Figure 4.3: Solar powered 96 core ARM cluster (Source: www.phoronix.com)

4.2.3 ARM cluster for HPC in the cloud

A 40 node Pandaboard cluster was built by a team comprising of people various universities in the US. The goal of the project was to compare the performance of the dual-core Pandaboard cluster against a cluster of Intel Core 2 Duo machines. The Pandaboard has a 1 Ghz dual core ARM Cortex-A9 CPU. The dual-core Intel node had a clock frequency of 3 Ghz and was run in 32-bit mode since the ARM Cortex-A9 is a 32-bit processor [35].

The performance comparison was intended to compare 2 domains in HPC. The first comparison was when used as a standard dedicated cluster. The second comparison was for a cluster of Qemu virtual machines running on the cloud. The study also involved scaling from 1 node up to 40 nodes.

The dedicated cluster was benchmarked using the NAS parallel benchmark suite which had around 6 problems commonly used in CFD applications within HPC. The cluster was also benchmarked and compared for the time taken to checkpoint the NAS/LU.A and the Qemu VM.

During peak load, the Intel Core 2 Duo consumed up to 70 Watts of power while the ARM Cortex-A9 used 5 Watts. The performance per watt measurements on the ARM Cortex-A9 was 1.3 to 6.2 times greater than that of the Intel Core 2 Duo. The Intel processor was however 5 times greater in raw performance. This is mainly due to the higher clock speed of the Intel processor. The results also showed that for the largest problem size running on a cluster size of 1-4 nodes, the energy efficiency ratio was as low as 0.9 since the Intel Core 2 Duo has a 6 MB L2 cache when compared to a 1 MB cache on the ARM Cortex-A9. However for large problem sizes on a large cluster size, the ARM outperformed the Intel processor in performance per watt measurements.

The study concluded that as future SoCs begin using the ARM Cortex-A15 processor and we start seeing processor implementations of the 64-bit ARMv8 architecture, ARM will definitely make an impact in HPC and cloud computing.

5 Cluster Design

In this chapter we will look at the various steps undertaken and issues encountered while building the 6 node Pandaboard cluster. Two Raspberry Pi boards were also configured to run different Linux OS versions and compared for performance.

5.1 Hardware Setup

The hardware setup for the Pandaboard cluster consisted of a dual-core x68 master node, 6 Pandaboard ES boards, 2 Raspberry Pi boards, a power meter and a gigabit switch. Chapter 3 provides the details regarding the Pandaboard ES and Raspberry Pi boards.

An Intel dual-core machine with a clock frequency of 1.86 Ghz for each core was selected as the head node. The memory and disk space on the head node is 4GB and 235 GB respectively. The head node in the cluster was responsible for submitting and scheduling jobs on the slave nodes in the cluster. The head node acted as a central point from where commands meant to be run on the slave nodes in the cluster could be submitted. The head node also collected the power measurement information from the power meter via the USB port using a program that monitors the USB port for incoming data.

The power measurement was done by placing the power meter between the power source and the cluster nodes. The power meter was configured to output the power consumed by the cluster in watts every second. This information was then fed to the head node using a USB interface. A more detailed explanation of the procedure to measure power is given in chapter 6.

The switch used to network the nodes in the cluster was a 8 port Netgear GS608 gigabit Ethernet switch. The power consumed by the switch was not captured and measured as we were mainly interested in the power consumption of the Pandaboard and Raspberry Pi nodes.

The summary of the hardware specification for the cluster is as given in Table 5.1.

Processor	Clock	Memory	Storage	NIC	Hostname
Intel Core 2 Duo	1.86 Ghz	4 GB	HDD 235 GB	1 Gbps	master
ARM Cortex-A9	1.2 Ghz	1 GB	SDC 16 GB	10/100 Mbps	panda1
ARM Cortex-A9	1.2 Ghz	1 GB	SDC 16 GB	10/100 Mbps	panda2
ARM Cortex-A9	1.2 Ghz	1 GB	SDC 16 GB	10/100 Mbps	panda3
ARM Cortex-A9	1.2 Ghz	1 GB	SDC 16 GB	10/100 Mbps	panda4
ARM Cortex-A9	1.2 Ghz	1 GB	SDC 16 GB	10/100 Mbps	panda5
ARM Cortex-A9	1.2 Ghz	1 GB	SDC 16 GB	10/100 Mbps	panda6
ARM1176JZF-S	700 Mhz	256 MB	SDC 8GB	10/100 Mbps	rpi1
ARM1176JZF-S	700 Mhz	256 MB	SDC 16GB	10/100 Mbps	rpi2

Table 5.1: Hardware specification for the cluster

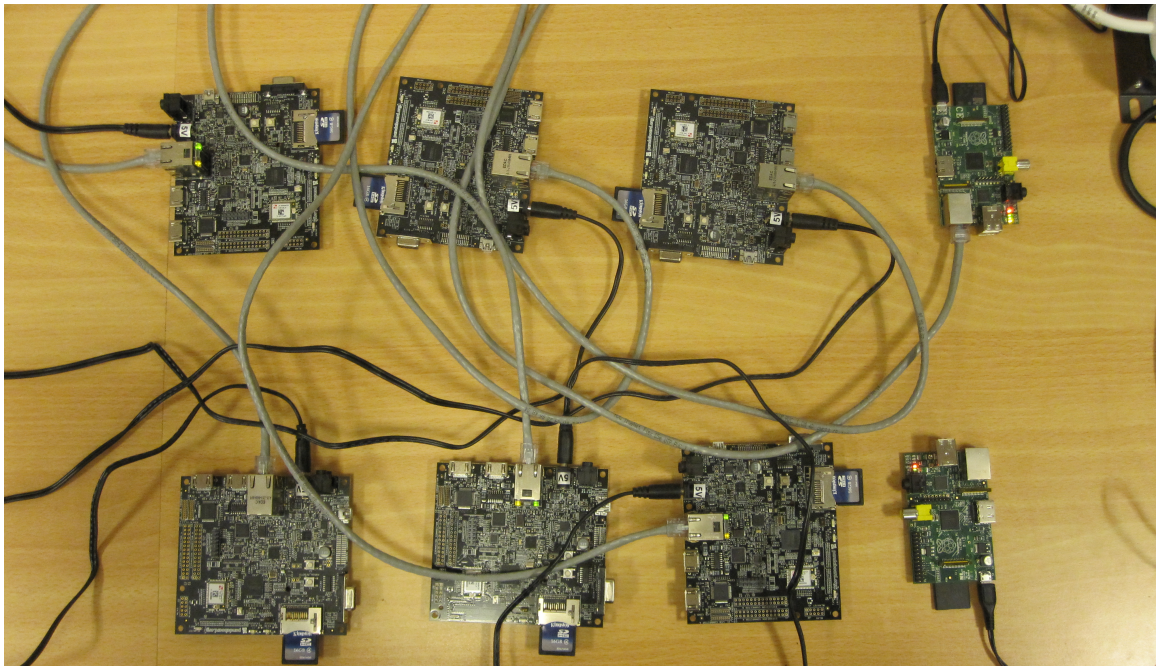

Figure 5.1: Project cluster setup

5.2 Operating System Setup

5.2.1 Panandboard ES

The ARM architecture is supported by a wide range of operating systems. The Pandaboard ES has a ARM Cortex-A9 CPU which implements the ARMv7 architecture. The Pandboard community offers support for Linux distributions such as Ubuntu and Linaro. The community also provides an Android port for the OMAP4 platform [33].

For this project, we decided to install the latest Ubuntu 12.10 “Quantal Quetzal” server edition on the Pandaboards. The Ubuntu 12.10 has a newer Linux kernel (3.4) and also supports the latest GCC 4.7 version. The other option was to use the older Ubuntu 12.04 “Precise Pangolin” release. The 12.04 release uses an older Linux 3.2 along with GCC 4.6. The newer kernel and GCC compiler on the 12.10 could offer improved performance over the 12.04.

The pre-installed compressed Ubuntu 12.10 image for OMAP4 can be downloaded from the web and copied to the SD card using the following command,

sudo dd if=/path/to/image/linux_image.img of=/dev/mmcblk0 ; sync

This Pandaboard ES has an HDMI and a DVI port for display. In order to boot the image and setup the initial configuration on the Pandaboard, a serial ouput cable was connected to the serial port on the Pandaboard. This was then connected to a host machine that supported serial input or a serial to USB converter could be used to plug the cable into the USB port on the host machine. On the host machine a program such as Minicom or Screen was used to read from */dev/ttyUSB0*.

One of the issues we faced during this step was that although the output from the serial port was being displayed on the host machines terminal, there was no input command being sent back from the host machine to the Pandaboard's serial port. This caused the initial setup to stall on the language selection screen waiting for input. After trying out various options, a decision was taken to disable the Pandaboard's initial configuration program called *oem-config* by manually

modifying the *oem-config* setup program. By doing this we were skipping all the initial configuration steps such as choice of language, date & location configuration and user account setup. In order to login and configure the system manually, we needed root access. Hence a dummy password was manually setup for the root user by modifying the */etc/password* file on the SD card. These modifications ensured that the initial *oem-config* steps were bypassed and a login prompt was obtained on the monitor connected to the Pandaboard through the HDMI port.

5.2.2 Raspberry Pi

On the Raspberry Pi, 2 different versions of Debian Linux were installed. The first version is the Debian armel which uses the “EABI baseline” C ABI. This version of Debian uses software in order to perform single and double precision floating point arithmetic. The Debian armel version does not make full use of the VFP (Vector Floating point) unit on the Raspberry Pi. We can build code compatible with Debian armel which use hardware floating point instructions by passing the flags *-mfpu=vfp* and *-mfloat-abi=softfp* to the GCC compiler. However since the system libraries and binaries still use software floating point and not the VFP to carry out its floating point calculations, there is very little performance gain.

The newer Debian armhf port for the Raspberry Pi is called Raspbian. This port for the Raspberry Pi has a Linux kernel that has been completely recompiled with the “EABI vfp hardfloat” C ABI. It is expected that for floating point computation intensive applications, this Linux port will provide a huge increase in performance. Also the ARM CPU running the Raspbian Linux OS utilizes the 128 KB L2 cache which by default is used by the GPU in the armel version. Hence for memory intensive operations, the Raspberry Pi running Raspbian could potentially give better performance.

The 2 Debian Linux ports were copied and installed on separate SD cards and the initial setup on the Raspberry Pi was without any issues.

5.3 Networking

The head node in the cluster had 2 network interfaces *eth0* and *eth1*. The first interface *eth0* was used for communication within the private network (cluster) and had a private IP address. The second interface *eth1* had a public IP address and was used to access the external network and the internet. Each of the slave nodes in the cluster also had an IP address and could communicate with the head node and each other through a switch. The switch used in this setup was an 8 port gigabit switch. In order to install software packages easily on the slave nodes, it was important for the slave nodes to have access to the internet. To achieve this, the gateway for all the slave nodes in the cluster was configured as the master/head node. The master node was setup to forward the internet requests from the other nodes in the cluster. Hence all network traffic to the public network were routed through the master. The network configuration for the entire cluster is as shown in Table 5.2.

Hostname	IP Address	Status
GX745MSCP1	129.215.62.197	Gateway
master	192.168.1.6	Head node
rpi1	192.168.1.7	Compute node
rpi2	192.168.1.8	Compute node
panda1	192.168.1.11	Compute node
panda2	192.168.1.12	Compute node
panda3	192.168.1.13	Compute node
panda4	192.168.1.14	Compute node
panda5	192.168.1.15	Compute node
panda6	192.168.1.16	Compute node

Table 5.2: Summary of network configuration

The network connectivity between the master node and the slave nodes in the cluster is illustrated in Figure 5.2 below.

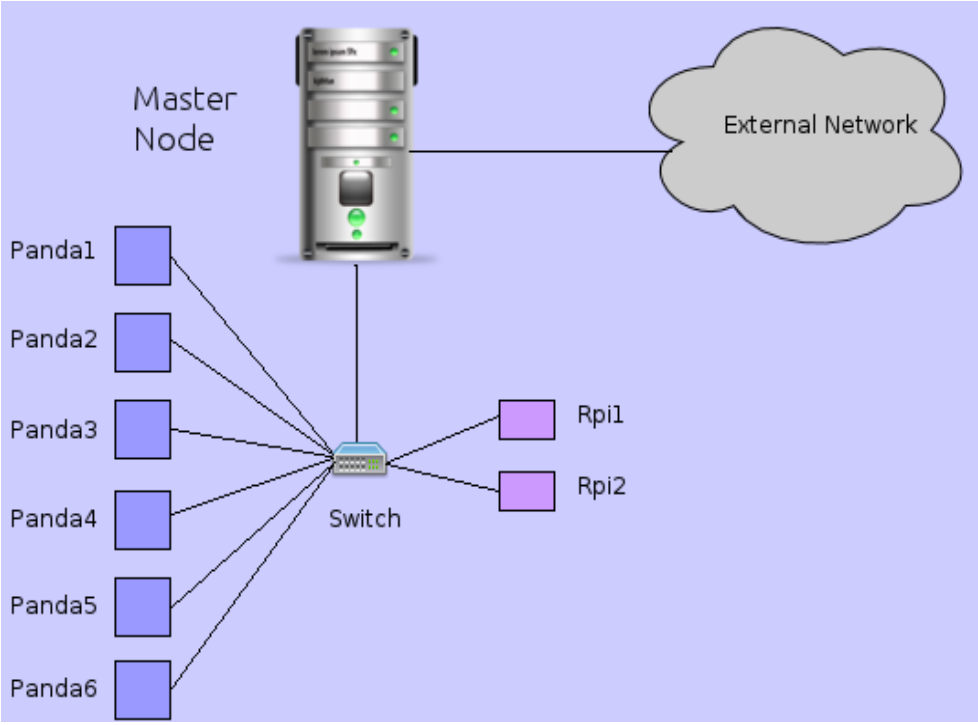

Figure 5.2: Network connectivity

5.4 Software setup

Ubuntu comes packaged with the *apt-get* tool which is an advanced packaging tool that is used to install and remove libraries and packages on Debian based systems. We made use of the *apt-get* utility in order to find and install various software tool chains if available on the Pandboards. This is one of the advantages of choosing Ubuntu as the OS because there is a big community of developers constantly updating various Ubuntu package repositories with tools and utilities.

5.4.1 C/C++

There are a wide variety of C/C++ compilers both commercial and open source that are available for the ARM architecture. However for this project we decided to use the GCC/G++ compilers because they are freely downloadable and also provides a wide variety of optimization options for the ARMv7 architecture. The 4.7.1 GCC/G++ version were installed on the Pandaboards and supports shared memory APIs for Posix threads and OpenMP.

5.4.2 Fortran

Libraries like BLAS, LAPACK and benchmarking codes like FFT and EB which are part of the NAS parallel benchmark suite require a Fortran compiler. If there is no available Fortran compiler, it would have been necessary to port the code from Fortran to C using a utility such as Netlib's *f2c* [36]. This was indeed the case in one of the MSc. dissertations from 2011 where scripts were created in order to port a lot of the Fortran code to C [37]. Linux distributions such as Debian and Ubuntu provide GNU Fortran 95 compiler for armel and armhf architectures. The GNU Fortran version 4.7.1 was installed on the Pandaboards.

5.4.3 Message Passing Library

The library used for inter-node communication is MPICH2 release 1.4.1. MPICH2 was developed by Argonne National Laboraty and is an implementation of the MPI-2.2 standard. The library is portably across a wide variety of platforms and architectures [38]. The MPICH 1.4.1 supports multi-threaded code that uses hybrid MPI-OpenMP programming model. Inter-process communication within the node is achieved using shared memory rather than using TCP/IP or other such protocols. The default protocol used for communication on an Ethernet based system is TCP/IP. However this can be changed during the compilation of MPICH2 to use a protocol such as OpenMX. OpenMX is a high performance implementation of the Myrinet Express stack over Ethernet [39]. The advantage of using OpenMX instead of TCP/IP is the reduction in latency during inter-node message passing. In this project we compiled the MPICH2 library with the default TCP/IP over Ethernet option. We could not explore the possibility of using OpenMX due to lack of time.

5.4.4 Queue Management and Job Scheduling

A queue management utility was necessary in order to submit and manage jobs on the cluster. The queue management utility identifies the resources on the cluster and enables the client nodes on the cluster to communicate with the master node. The TORQUE resource manager version 2.4.16 was installed on the cluster. TORQUE

stands for Terascale Open-source Resource and Queue Manager. TORQUE is based on the original PBS project and is highly scalable, reliable and fault tolerant [40].

TORQUE has an inbuilt scheduler program that is a very rudimentary scheduling tool. It follows a round robin schedule to assign jobs to nodes and also does not have advanced features like assigning queues to a subset of resources. For this project, we setup separate queues for the Raspberry Pi nodes and the Pandaboard resources. Hence there was a need for a more advanced job scheduler. We used the open source Maui cluster scheduler version 3.3.1 to enable more advanced scheduling. Maui was initially developed by Cluster Resources, Inc. and was designed to be used on clusters and supercomputers [41].

5.4.5 Network File System

In a homogeneous HPC cluster, the application code to be executed on the cluster is compiled on the front-end or head node as the architecture of the head node is the same as that of the slave nodes in the cluster. In this project we use an Intel x86 machine as the head node and the slave nodes are either ARMv7 or ARMv6 architecture. Hence compiling the application code on the head node is not an option due to binary incompatibility. The solution to this is to compile the code on one of the back-end nodes depending on the architecture and generate the correct binary. However in order for the MPICH2 Hydra process manager to run parallel jobs on the cluster, the generated binary should be available on all the resources in the cluster. This requires manually copying the binary to all the nodes. This is a laborious and time consuming process.

To ease this process, we installed an NFS server on the front-end node. An entry was added to the */etc/fstab* on all the slave nodes on the cluster to mount a directory present on the front-end node over the network. Now any program that is compiled on this NFS mounted path is visible to all nodes on the cluster. Hence the MPI runtime can execute the binary using the NFS mount path on the requested resources. Since the code is compiled on the correct architecture there is no binary incompatibility.

6 Benchmarking & Power Measurement

In this chapter we will discuss about the various benchmarks that were run on the Pandaboard ES boards and the Raspberry Pi boards. The benchmarks were chosen based on various performance metrics of the system and cluster that is of interest to HPC applications.

6.1 CoreMark

CoreMark is a system independent benchmark developed by the Embedded Microprocessor Benchmark Consortium (EEMBC) [42]. It is seen as a replacement for the Dhrystone benchmark developed in the 80s which is no longer applicable to current systems. The benchmark can be run in single thread as well as multi-threaded mode.

The CoreMark benchmark implements algorithms and data structures used commonly in most applications [43]. Some of the tests performed are as follows,

1. List processing such as sorting, searching and reversing a list. This tests the non-serial access of memory with the use of pointers. It also tests the memory hierarchy and cache efficiency of a system for lists that do not fit into the systems cache [43].

2. A 16-bit Cyclic Redundancy Check (CRC) is also performed on the data elements contained in the list. CRC is one of the most commonly used algorithms in embedded applications and hence this test is included in the timing of the benchmark [43].

3. Matrix operations such as multiplication with another matrix, a vector and a constant. CoreMark extracts bits from a part of the data in the matrix and performs operations [43]. A CRC is performed in the end to validate the operations on the matrix.

4. State machine processing where branch operations such as if, else and switch are used to test how the CPU handles control statements [43].

The final output of the CoreMark benchmark is the number of iterations executed per second. The number of iterations can be changed during compilation depending on the attributes of the machine such as clock speed, memory capacity, number of levels of cache etc.

6.2 STREAM

The STREAM benchmark was developed by John McCalpin and is the industry standard for measuring the memory bandwidth of a system and the rate of computation for simple vector kernels [44]. STREAM counts the number of bytes to be read plus the number of bytes to be written. There are 4 operations performed by STREAM implemented as 4 separate kernels and the output gives the memory bandwidth obtained for each operations in MB/s. The table below shows the flops and bytes counted per iteration of the STREAM loop.

Name	Kernel	Bytes/Iter	FLOPS/Iter
COPY	a(i) = b(i)	16	0
SCALE	a(i) = q * b(i)	16	1
SUM	a(i) = b(i) + c(i)	24	1
TRIAD	a(i) = b(i) + q * c(i)	24	2

Table 6.1: Shows number of bytes and FLOPS counted in each iteration (Source: www.cs.virginia.edu/stream)

6.3 Linpack

Linpack is a program implemented either in Fortran or C to perform numerical linear algebra computations on computers. It was initially developed as a benchmark to measure the number of FLOPS on supercomputers. Linpack solves a dense *NxN* system of linear equations $Ax=b$ as it is a common problem in engineering and scientific computing. Linpack makes use of the Basic Linear Algebra Library

(BLAS) to perform matrix and vector operations. Linpack has now been superseded by LAPACK which is a more suitable for modern architectures [45]. The Linpack benchmark used in this project measures the double precision floating point performance on a 200x200 array. The performance output is calculated as the average rolled and unrolled performance.

6.4 HPL

High Performance Linpack (HPL) is a parallel implementation of the Linpack benchmark and is portable on a wide number of machines. HPL uses double precision 64-bit arithmetic to solve a linear system of equations of order N [46]. It is usually run on distributed memory computers to determine the double precision floating point performance of the system. The HPL benchmark uses LU decomposition with partial row pivoting. It uses MPI for inter-node communication and relies on various routines from BLAS and LAPACK libraries.

The algorithm can be described as having the characteristics, "*Two-dimensional block-cyclic data distribution - Right-looking variant of the LU factorization with row partial pivoting featuring multiple look-ahead depths - Recursive panel factorization with pivot search and column broadcast combined - Various virtual panel broadcast topologies - bandwidth reducing swap-broadcast algorithm - backward substitution with look-ahead of depth 1*" [46].

The input file, *HPL.dat* for the HPL benchmark provides information regarding the problem size, the block size, the grid dimension etc. This input file can be tweaked for performance according to the system on which the HPL benchmark is being run on and the network topology used for interconnecting the nodes.

6.5 Ping Pong

The ping pong benchmark measures the latency and bandwidth of network communication between 2 nodes on the cluster. Latency is defined as the hardware and software overhead involved in transmitting a zero byte message between 2 MPI

processes. The bandwidth is the rate of transmission of data between 2 MPI processes. The ping pong benchmark allocates a message of specified size and sends it from one process to another and the same message is received back. This communication pattern is performed over multiple iterations and the time taken to execute these iterations is calculated. The latency and bandwidth of the communication is given by the formulae,

$$Latency = 0.5 * (T2 - T1) / Total\ Number\ of\ Iterations$$

$$Bandwidth = Total\ Message\ Size / (T2 - T1)$$

Where (T2 – T1) gives the total time to execute all iterations in seconds. Latency is given in milliseconds or seconds and Bandwidth is in *MB/s.*

6.6 NAS Parallel Benchmarks

The NAS parallel benchmarks developed by NASA consist of a set of programs that implement various algorithms used in Computational Fluid Dymamics (CFD) [47]. There are 5 kernels and 3 pseudo applications that are part of this benchmark suite. For this project we are interested in 2 kernels namely, Embarrassingly Parallel (EP) and 3D Fast Fourier Transform (FFT). We use the MPI version of these benchmarks and scale the problem size as the number of processes grows.

6.6.1 Embarrassingly Parallel

This benchmark measures the upper limit of floating point performance of the cluster without having significant inter-process communication. Embarrassingly parallel problems scale well on clusters where the inter-node communication could be a bottleneck since there is very little communication taking place. Since the Pandaboard uses a 10/100 Mbps network card and uses TCP/IP over Ethernet for inter-node message passing, there is a significant performance loss when scaling benchmarks that use inter-node communication frequently. An embarrassingly parallel benchmark should scale almost linearly on this cluster and give us a good

measure of raw floating point performance. This type of benchmark is typically used in simulation applications that use the Monte Carlo method.

6.6.2 3D Fast Fourier Transform

3D Fast Fourier Transforms kernel is used widely in spectral codes. This kernel uses forward and inverse FFTs to numerically solve a partial differential equation. This benchmark tests the network performance of the cluster rigorously as the array transpose operations being performed require the use the all to all communications [48]. The MPICH2 library version used in this project uses shared memory for on node communication. For inter-node communication it uses TCP/IP over Ethernet. We should see a performance degradation when inter-node communications happen and should be a good test to measure the network performance and power usage of the cluster when there is significant network activity.

6.7 Power Measurement

Measuring power accurately is an important aspect of this project. There are many techniques for measuring the power consumption of the cluster. A software approach would involve using a tool such as Intel Power Gadget to measure the processor power in watts using energy counters [49]. The other approach is to use a power meter between the power supply and the cluster nodes. Due to lack of free power measurement tools available for ARM architectures we used the second approach.

The device used to measure the power consumption of the cluster is a *Watts Up? PRO* power meter. The meter captured the power consumption of the components connected via its output socket. The power meter comes equipped with a USB interface which can be plugged into the front-end node to log the power utilization of the node(s) being monitored. In order to read the power measurements from the USB device on the front-end node a software was needed to interpret and output the incoming data. This was achieved by using a free user developed C program that reads the input from */dev/ttyUSB0* and outputs the power in watts to the screen every second.

The power measurement is typically started just before a benchmark is submitted to run on the cluster. Once the benchmark completes, the power capture program on the front-end is terminated. The power consumption of the cluster can have variations during the run time of the benchmarks depending on the type of computation or communication happening on the nodes in the cluster. Hence we take the average power consumed by the cluster over the total run time of the benchmark. A Perl script was used to parse the power information and calculate the average power consumption of the cluster.

6.7.1 Metrics

The Top500 list ranks supercomputers according to the raw floating point performance. Power consumption although captured is not a criteria in ranking these big machines. The Green500 list on the other hand ranks the most energy efficient supercomputers in the world. The HPL benchmark is used to calculate the performance of the cluster in GFLOPS. The power consumed by the cluster during the running of HPL is also captured. Then the performance per watt is determined using the following formulae [50],

$$\textit{Performance Per Watt (PPW)} = \textit{Performance} / \textit{Power}$$

For the Green500 list this is given as MFLOPS/watt. The most energy efficient supercomputer today is the IBM Blue Gene/Q which has a PPW of 2100.88 [4].

The *performance* value given in the formulae above can be any other performance measure of the system using other benchmarks. Some of the PPW values for the other benchmarks in this project are given in Table 6.2,

Benchmark	PPW
CoreMark	Iterations / Watt
STREAM	MB / Watt
HPL	MFLOPS / Watt
NAS EB	Mops / Watt
NAS FT	Mops / Watt

Table 6.2: Performance Per Watt calculation for various benchmarks

7 Results and Analysis

In this chapter we analyse the results for the various serial and parallel benchmarks that were run on the Pandaboard and Raspberry Pi boards. We focus on the raw performance figures for these benchmarks and also take into account the power consumption during the running of the benchmarks. The performance per watt (PPW) data is also presented and discussed.

7.1 CPU Frequency Scaling

CPU Frequency scaling refers to the technique by which the CPU can scale the clock frequency of its cores up or down depending on the load of the system [51]. This feature is useful in scenarios where power saving becomes critical. For example when mobile devices run on battery power, running the CPU cores at full clock frequency is a waste of energy. Instead if the CPU is able to scale down the frequency when there is little or no load, there could be a significant amount of savings made in terms of energy consumed.

This functionality is achieved with the help of CPU frequency governors which are part of some Linux kernels. There are 5 types of governors in the Linux kernel,

Performance: In this mode the CPU frequency is set statically to the highest frequency value as specified in *scaling_max_freq.* This governor is desirable on machines that are used to run applications that expect the maximum performance from the CPU.

Ondemand: This governor is set by default after the installation of Ubuntu. With this governor the CPU frequency is scaled up or down depending on the current usage of the system. The CPU must perform this change is frequency very quickly, usually in nano seconds in order to ensure that applications running on the machine are able to get extract good performance from the CPU.

Conservative: The *conservative* governor performs the same functionality as the *ondemand* governor where the CPU clock frequency is scaled up or down depending

on the current usage. However unlike the *ondemand* governor the scaling is done more gracefully wherein the increase in clock speed is gradual.

Userspace: This governor allows any program or user with root privilege to set the frequency of the CPU manually. This could be useful in cases where we need to over clock the CPU.

Powersave: With this governor, the CPU frequency is set statically to the lowest frequency value as given in *scaling_min_frequency*. This mode would be suitable for systems where performance is not a criteria and saving power is the highest priority.

A user can set a specific governor by modifying the following file,

/sys/devices/system/cpu/cpu/cpufreq/scaling_governor*

This scaling governor can also be changed using the *cpufreq-set* tool which can be downloaded and installed on the Ubuntu. Once installed, the governor can be changed using the following command,

cpufreq-set -g <governor name>

We use the *performance* governor for all the machines in the project since we want to obtain maximum performance from the system.

7.1.1 Idle Power Reading

Machine	Governor	Average Power (Watts)
Pandaboard ES	Ondemand	2.124
Pandaboard ES	Performance	2.607
Raspberry Pi	NA	1.993

Table 7.1: Average idle power usage

We measured the average idle power consumption of the Pandaboard ES board using both the *ondemand* as well as the *performance* governors. The average idle

power consumption was also measured for the Raspberry Pi running the Raspbian Linux OS with a clock frequency of 700 Mhz. These readings are presented in Table 7.1 and the graph in Figure 7.1 shows the variation in power consumption for a period of 1 hour.

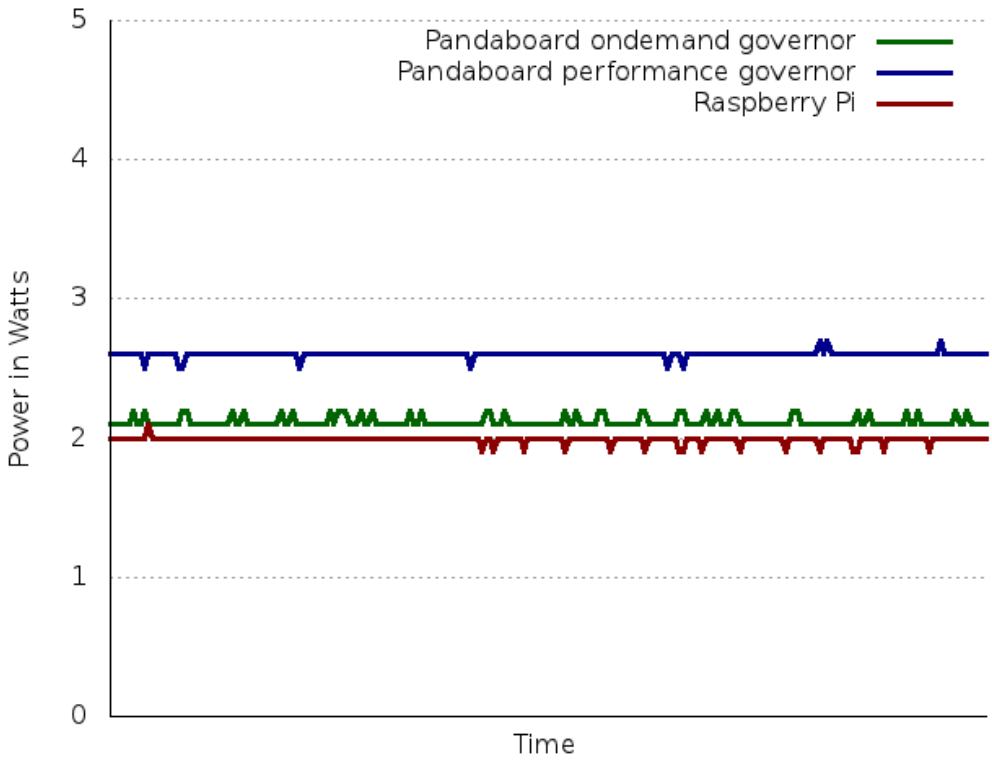

Figure 7.1: Idle power usage over 1 hour

7.2 CoreMark

As discussed in Chapter 6, the CoreMark benchmark measures the number of iterations per second by executing a variety of algorithms. The CoreMark benchmark was run on the Pandaboard ES board both in single threaded and multi-threaded mode. The single threaded version was run on the Raspberry Pi board using both the Debian armel and Raspbian versions of Linux.

The CoreMark source code was compiled with various levels of optimization in order to compare the improvement in performance as well as power usage. The number of iterations was chosen as 1 million to ensure that the benchmark runs for at least for 10 minutes. The CoreMark benchmark has 2 phases. Phase 1 is the performance run and phase 2 is the validation run. Both these phases are timed and

the number of iterations chosen is the same for both phases. The readings are presented only for the first phase as the results in both phases are similar.

Table 7.2 shows the iterations and iterations per watt along with the various compiler optimization levels for both the Pandaboard ES and Raspberry Pi boards.

Machine	Threads	Iterations/sec	Avg. Power (Watts)	Optimization	PPW
Pandaboard	1	3237.199	4.080	-O2	793.431
Pandaboard	1	3364.262	4.200	-O3	801.014
Pandaboard	1	3398.332	4.200	-O3 unroll	809.126
Raspberrry Pi (armel)	1	1309.066	2.410	-O2	543.180
Raspberrry Pi (armel)	1	1364.904	2.404	-O3	567.763
Raspberrry Pi (armel)	1	1405.797	2.405	-O3 unroll	584.530
Raspberrry Pi (armhf)	1	1282.0036	2.240	-O2	572.323
Raspberrry Pi (armhf)	1	1321.0039	2.253	-O3	568.331
Raspberrry Pi (armhf)	1	1381.1963	2.240	-O3 unroll	616.605
Pandaboard	2	6456.882	5.320	-O3 unroll	1213.699

Table 7.2: CoreMark performance

On the Pandaboard ES board we see that in the single thread mode there is a marginal increase in the performance per watt as we compile the code with the optimization level 3 and also explicitly unroll loops. The average power usage of the Pandaboard ES also goes up marginally. The level 3 optimization with gcc involves all the flags of level 2 plus additional flags such as inlining functions and tree vectorizing [52].

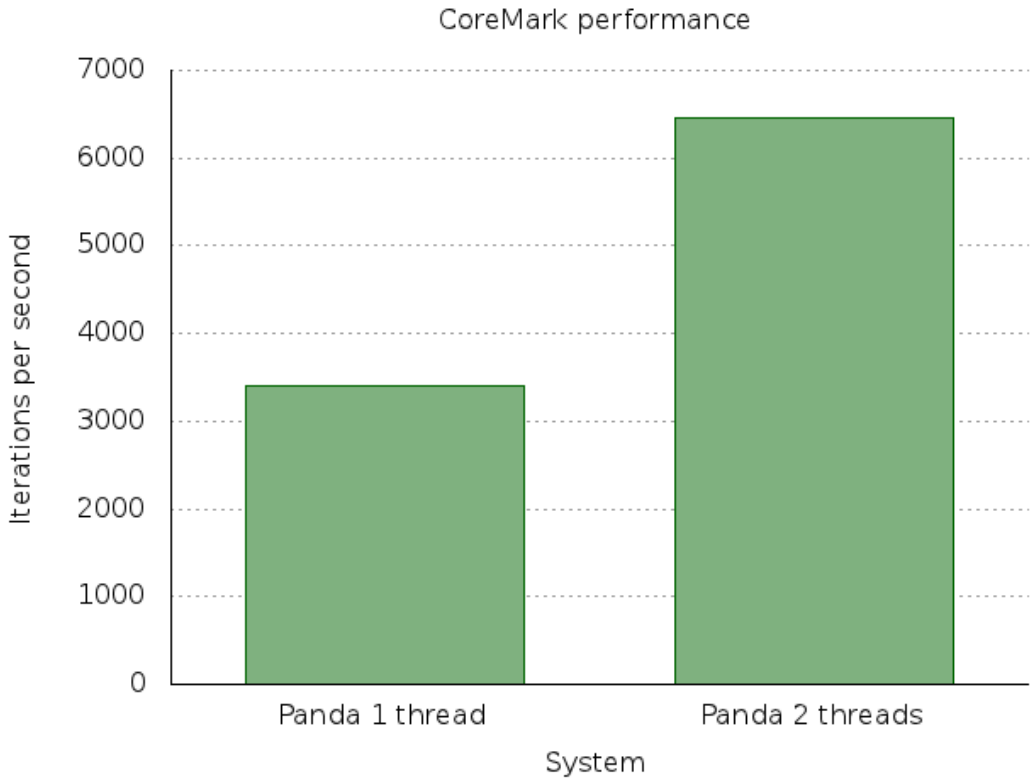

Figure 7.2: Single and multi-threaded performance on Pandaboard ES

From Figure 7.2 we see that when CoreMark is run using 2 threads on the Pandaboard, we get almost linear speedup. This is the expected behaviour as the code runs the same number of iterations on both the threads and there is no communication overhead. However we see in Figure 7.3 that the iterations per watt does not remain the same when both the cores are used on the Pandaboard. When CoreMark is run on a single core using one thread, the other free core still uses some amount of power (idle power) and that power is part of the average power calculated while determining the iterations per watt value. Ideally we should only consider the power consumed by the core on which the benchmark runs in order to get the true performance per watt measure (PPW). In this case both the cores are running with the scaling governor set to performance. Also the idle power used by the Pandaboard (2.6 W) is constant for both the single and multi threaded power measurements. If the PPW is to remain constant, the idle power used should also double. For multi-core machines, the PPW is best measured by loading all the cores with work. We

must also consider the jitter caused by certain OS services such as interrupt handler and scheduler.

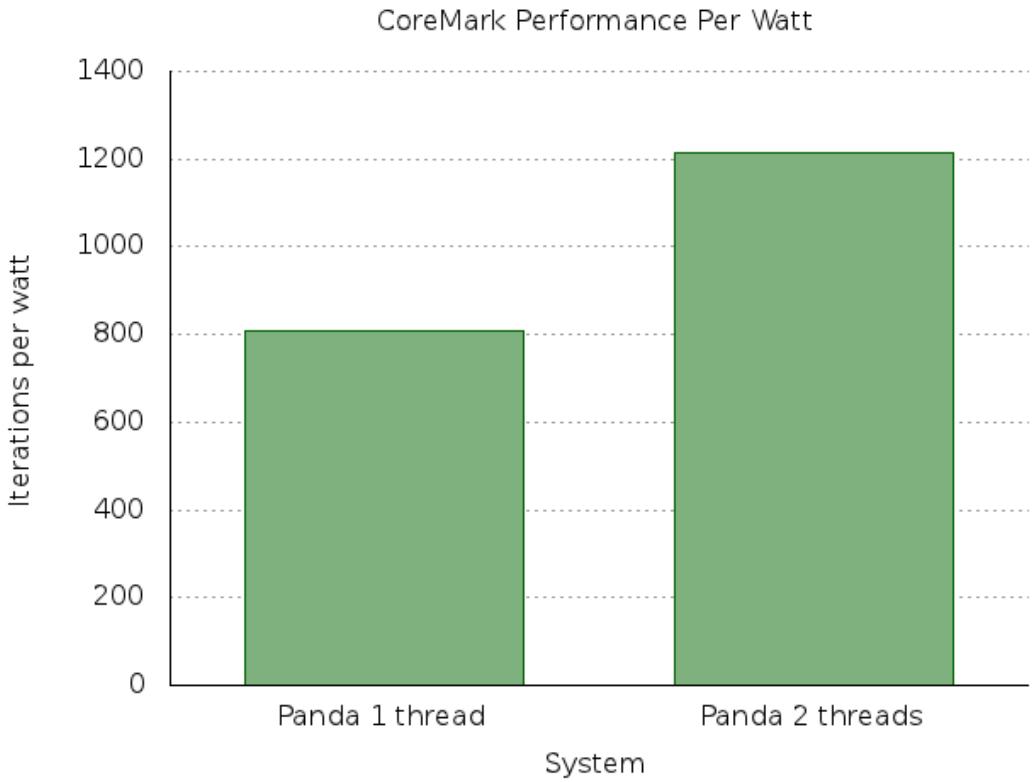

Figure 7.3: PPW for single and multiple threads

From Table 7.2 we see that the Raspberry Pi boards running both the Debian armel and Raspbian versions of Linux running the single thread version of CoreMark have approximately 40% of the raw performance of the Pandaboard. The Raspberry Pi clock frequency is 700 Mhz and has 25% of the memory capacity of the Pandaboard ES. The L2 cache size on the Raspberry Pi is also much smaller than that of the Pandaboard ES. This explains the lower performance of the Raspberry Pi when compared to the Pandaboard ES. However the Raspberry Pi (running Raspbian) consumes only 2.25W which is just over 50% of the power consumed by the Pandaboard ES to run the single thread benchmark.

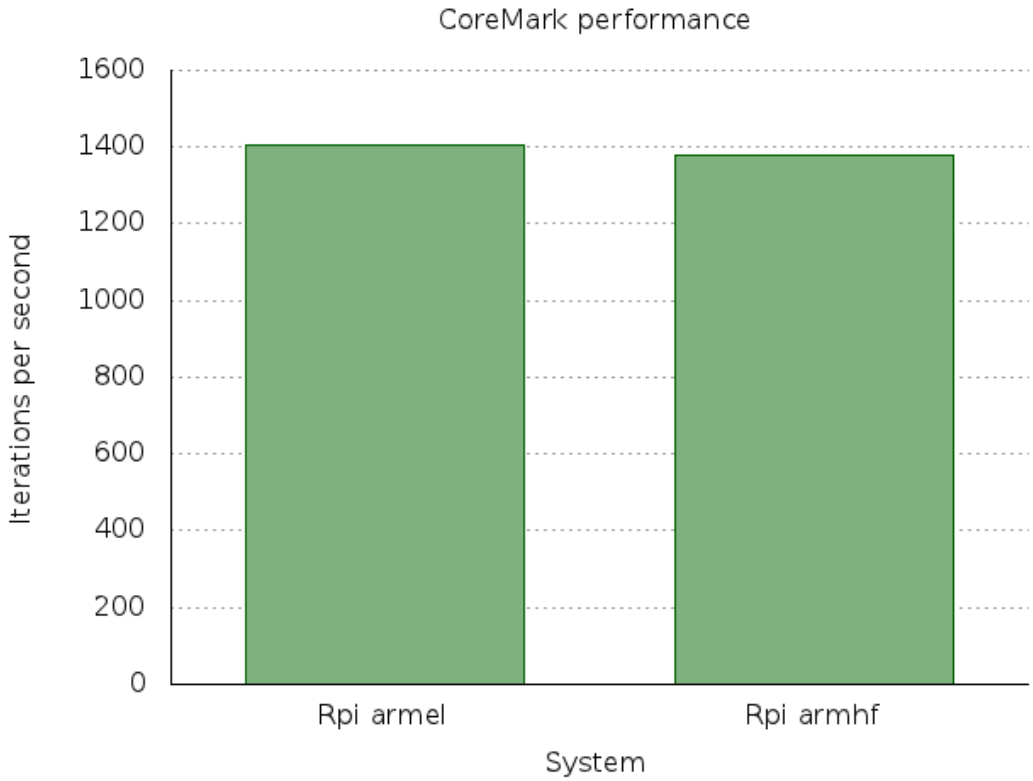

Figure 7.4: CoreMark performance on Raspberry Pi

The Raspberry Pi has an idle power usage of 2W. When we run CoreMark, the average power used only increases by 0.25W. The Pandaboard ES on the other hand uses 1.6W more power than its idle power of 2.6W. This makes the iterations per watt measurement of the Raspberry Pi much closer to that of the Pandaboard ES. As discussed earlier, the PPW measurement on the Pandaboard ES should be considered with both cores being used to run the benchmark to eliminate the idle power used by one of the cores when its not being used. If the multi-threaded performance of the Pandaboard is considered, we get approximately twice the performance per watt compared to the Raspberry Pi.

From Figure 7.5 we see that the the Raspberry Pi running Raspbian has a marginally better iterations per watt measure when compared to that of Debian armel. This is because the power consumption of the Raspberry Pi running Raspbian is less than that of the Raspberry Pi running the Debian armel version.

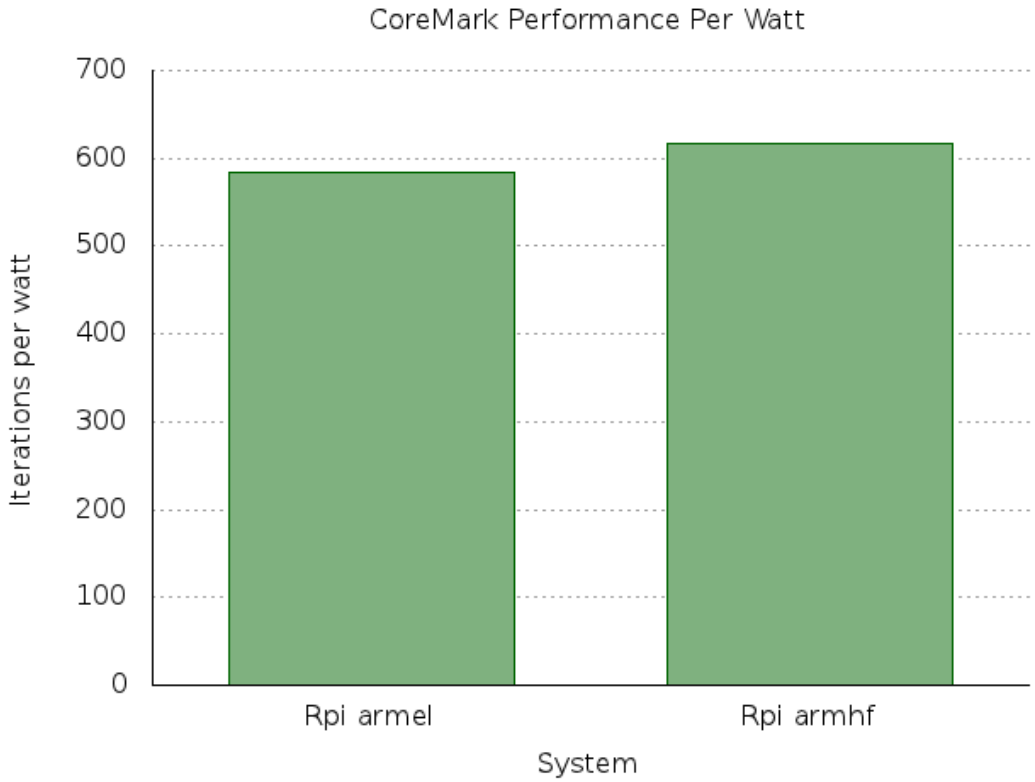

Figure 7.5: CoreMark PPW on the Raspberry Pi

7.2.1 PPW comparison to Intel Atom and Intel Xeon

From a previous MSc. project on low power HPC [37], we find that the CoreMark PPW result obtained on the Intel Atom was 65.9 iterations per watt and the Intel Xeon at 55.76 iterations per watt. This was the single threaded result for for 1 million iterations using CoreMark. As shown in Table 7.2, the single threaded PPW ratio of the ARM Cortex-A9 processor on the Pandaboard ES is 12 times greater than that of the Intel Atom and 14.5 times greater than that of the Intel Xeon. The ARM11 processor on the Raspberry Pi (running Raspbian) has a iterations per watt ratio approximately 9 times greater than that of the Intel Atom and 11 times greater than that of the Intel Xeon.

The Intel Atom processor benchmarked has 2 processors, each 2-way hyper-threaded giving a total of 4 virtual processors. The Intel Xeon has 8 cores. When the multi-threaded version of CoreMark was run on these machines with 4 threads on the Atom and 8 threads on the Xeon, the iterations per watt obtained was 201.7 and 432.90 respectively. The multi-threaded (2 threads) version on the ARM Cortex-A9

processor delivers a iterations per watt ratio approximately 3 times greater than that of the Intel Xeon and 6 times greater than that of the Intel Atom.

7.3 STREAM

Table 7.3 show the STREAM benchmark results for a memory size of 750 MB. The benchmark was run both in single and multi threaded modes. The code for the STREAM benchmark was also compiled with various GCC optimization levels along with flags that can assist the compiler to extract good performance from the Pandaboard.

Machine	Num. Threads	Function	Rate (MB/s)	Avg. Time (secs)	Avg. Power (Watts)	PPW	Compiler Flags
Pandaboard ES	1	Copy	741.566	0.7086	3.77	196.680	O2
		Scale	689.926	0.7613		182.984	
		Add	812.799	0.9693		215.573	
		Triad	782.349	1.0063		207.497	
Pandaboard ES	1	Copy	743.427	0.7086	3.76	197.719	O3
		Scale	691.064	0.7603		183.793	
		Add	813.235	0.9681		216.285	
		Triad	784.277	1.0044		208.584	
Pandaboard ES	1	Copy	998.482	0.5254	3.81	262.068	-O3 -fprefetch-loop-arrays -march=armv7-a -mtune=cortex-a9
		Scale	1008.503	0.5209		264.698	
		Add	808.591	0.9733		212.228	
		Triad	934.027	0.8423		245.151	
Pandaboard ES	2	Copy	1209.681	0.4346	4.33	279.372	-O3 -fprefetch-loop-arrays -march=armv7-a -mtune=cortex-a9
		Scale	1227.748	0.4284		283.544	
		Add	949.268	0.8305		219.230	
		Triad	1251.933	0.6300		289.130	

Table 7.3: Results for STREAM on the Pandaboard ES

As explained in Chapter 3, the RAM on the Pandaboard ES board is a 1 GB dual channel LPDDR2 memory. The peak transfer rate possible with the LPDDR2

technology is given as 32Gbps or 4 GB/s [53]. However we can lose memory bandwidth due to various reasons and therefore do not expect to attain full peak bandwidth while running the benchmark.

We see that between the O2 and O3 optimization levels there is almost no improvement in memory bandwidth obtained for the 4 kernels. When the option to prefetch memory is added, a drastic increase in memory bandwidth is obtained. The *-fprefetch-loop-arrays* flag tells the compiler to generate instructions to prefetch memory on platforms that support this feature. With the multi-threaded version of STREAM we get further improvement for all 4 kernels.

Figures 7.6 shows the memory bandwidth per second results obtained for both the single and multi threaded versions of STREAM.

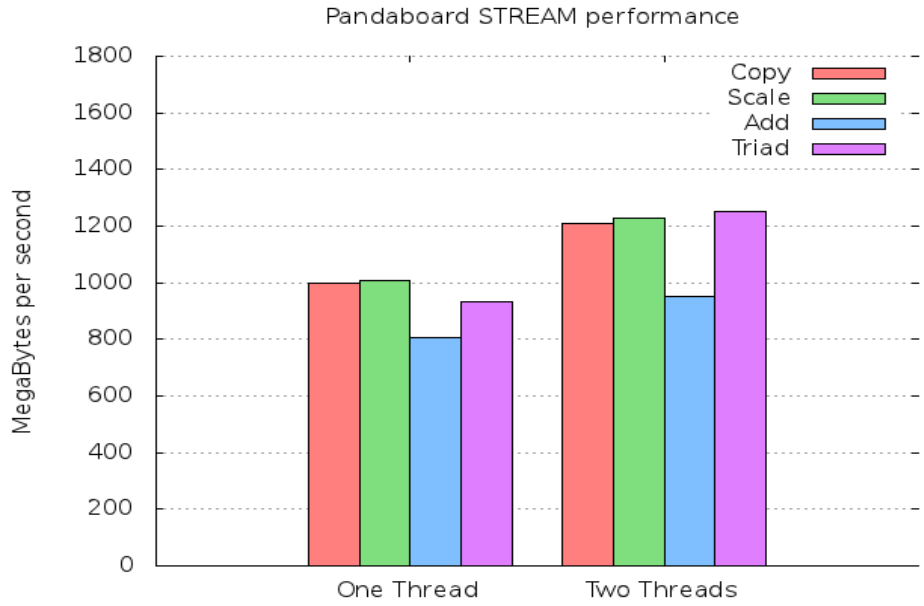

Figure 7.6: STREAM performance results on Pandaboard using O3 and prefetch flags

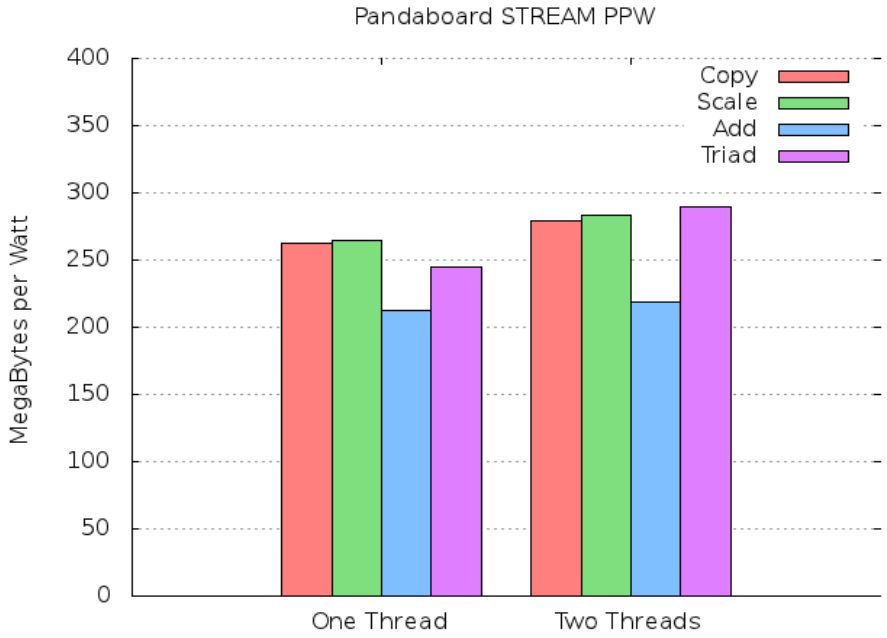

Figure 7.7: STREAM PPW results on Pandaboard using O3 and prefetch flags

When the STREAM benchmark was run with 2 threads on the Pandaboard, we did not see linear increase in performance. However the second core on the Pandaboard used more than half a Watt of power. Hence from Figure 7.7 we see that the MB/Watt metrics when running 2 threads is only marginally better than the single threaded version.

Machine	Num. Threads	Function	Rate (MB/s)	Avg. Time (secs)	Avg. Power (Watts)	PPW	Compiler Flags
Raspberry Pi (Debian armel)	1	Copy	193.976	0.5440	2.511	77.250	O3
		Scale	109.252	0.9784		43.509	
		Add	119.608	1.3554		47.633	
		Triad	78.073	2.0425		31.092	
Raspberry Pi (Debian armel)	1	Copy	194.126	0.5436	2.593	74.865	-O3 -fprefetch-loop-arrays
		Scale	166.799	0.6486		64.326	
		Add	158.872	1.0367		61.269	
		Triad	98.915	1.6502		38.146	

Raspberry Pi (Raspbian)	1	Copy	190.442	0.5516	2.35	81.039	O3
		Scale	181.604	0.5787		77.278	
		Add	175.529	0.8974		74.693	
		Triad	173.916	0.9077		74.006	
Raspberry Pi (Raspbian)	1	Copy	375.559	0.2801	2.369	158.530	-O3 -fprefetch-loop-arrays
		Scale	229.766	0.4598		96.988	
		Add	193.428	0.8147		81.649	
		Triad	239.299	0.6587		101.012	

Table 7.3: Results for STREAM on the Raspberry Pi

The STREAM benchmark with the various compiler optimization levels and the prefetch option was run on the Raspberry Pi running both the Debian armel and the Raspbian versions of Linux. The Raspberry Pi has a 256 MB RAM however we have configured the CPU/GPU split as 224/32 MB. The size of the data used was 150 MB. We see that adding the prefetch flag on the Debian armel version, we get a slight improvement in memory bandwidth for the *scale*, *add* and *triad* functions.

The Raspberry Pi running Raspbian on the other hand has the L2 cache available for use by the CPU and hence we expect to see better improvement in performance. As shown in Table 7.3 the Raspbian version is able to attain better memory bandwidth for the *scale*, *add* and *triad* operations than the Debian armel when just the O3 flags is used. When the *-fprefetch-loop-arrays* flag is used in conjunction with the O3 flag, we see that the Raspbian version is able to achieve much higher bandwidth for all 4 operations. This is mainly due to the availability of the 128 KB L2 cache on the Raspbian.

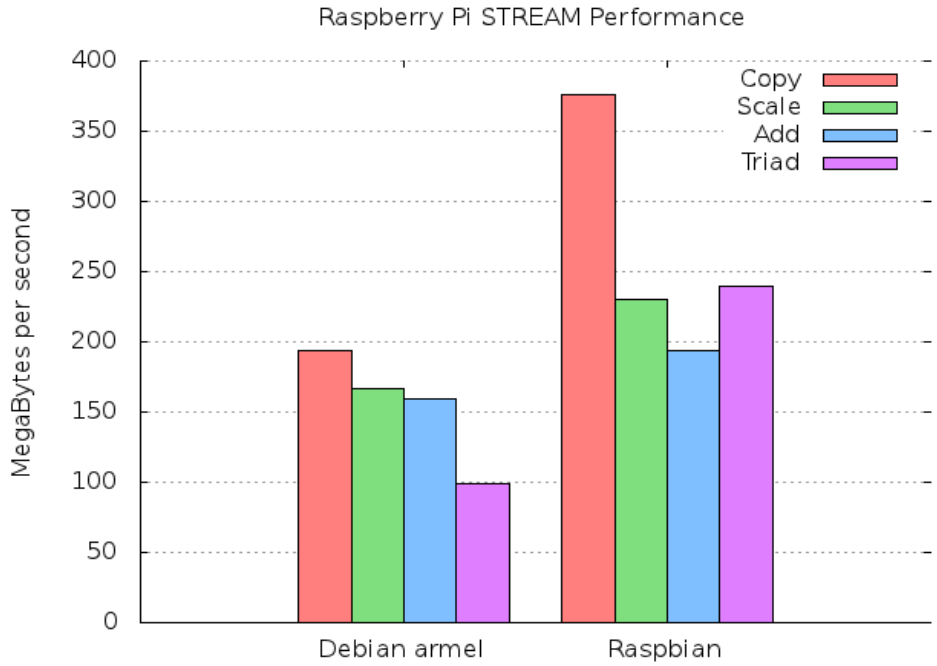

Figure 7.8: STREAM performance results on the Raspberry Pi using O3 & prefetch flags

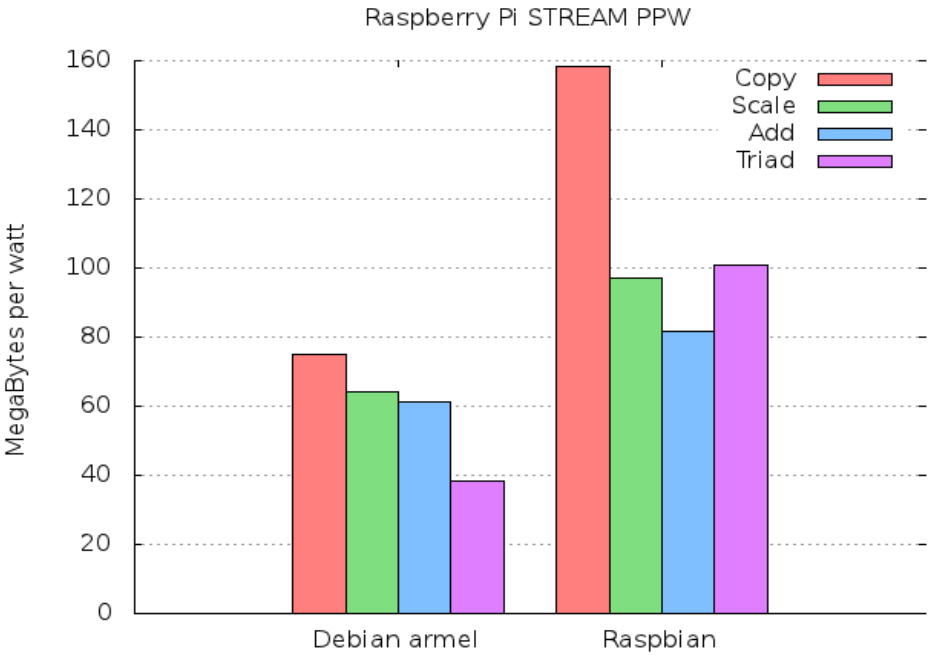

Figure 7.9: STREAM PPW results on the Raspberry Pi using O3 & prefetch flags

The performance per watt (MB/watt) measurement on the Raspberry Pi running both Debian armel and Raspbian is shown in Figure 7.9. Here we see that the Raspbian version is able to achieve better memory bandwidth using lesser power than the

Debian armel version. Thereby the performance per watt on the Raspbian is superior to that of the Debian armel version.

7.3.1 PPW comparison with Intel Atom and Intel Xeon

In Table 7.8 we see the bandwidth per watt comparison of the Pandaboard and Raspberry Pi versus that of the Intel Xeon and Intel Atom. The results for the Xeon and Atom were taken from a previous MSc. project on low power HPC.

Number of Threads	PPW ratio type	Function	PPW ratio
1	Pandaboard / Intel Xeon	Copy Scale Add Triad	8.495 8.575 6.322 7.214
1	Pandaboard / Intel Atom	Copy Scale Add Triad	4.013 4.574 3.077 4.820
2	Pandaboard / Intel Xeon	Copy Scale Add Triad	9.125 9.185 6.531 8.509
2	Pandaboard / Intel Atom	Copy Scale Add Triad	4.311 4.900 6.531 5.684
1	Raspberry Pi / Intel Xeon	Copy Scale Add Triad	5.178 3.168 2.432 2.972
1	Raspberry Pi / Intel Atom	Copy Scale Add Triad	2.44 1.67 1.18 1.98

Table 7.8: STREAM PPW comparison with Intel Xeon and Intel Atom

For 4 functions in the STREAM benchmark, the Pandaboard ES outperforms the Intel Xeon by approximately 6 to 9 times in bandwidth per watt measurements. When compared to the the PPW of the Intel Atom, the Pandaboard ES has approximately 3 to 5 times the bandwidth per watt metrics for the 4 kernels used in the STREAM benchmark.

The Raspberry Pi running Raspbian has approximately 2.5 to 5 times the bandwidth per watt metrics when compared to that of the Intel Xeon. The bandwidth per watt of the Raspberry Pi is only 1.2 to 2.4 times greater than that of the Intel Atom.

7.4 HPL

The HPL benchmark was only run on the Pandaboard ES cluster. HPL depends on the BLAS library for its basic vector and matrix operations. HPL also uses the LAPACK library which is a linear algebra package. We initially downloaded the source code for these libraries and compiled them manually. Then HPL was compiled using these libraries. However the performance obtained was a small fraction of the expected performance on the Pandaboard. The reason for the lack of performance was because the BLAS and LAPACK routines were not tuned and optimized for the Pandaboard ES. Machine specific BLAS libraries compiled by vendors such as IBM, Intel, Sun etc., are available for their respective hardware platforms. For the ARM architecture, we can use an automatic code generator such as ATLAS (Automatically Tuned Linear Algebra Software). In order to get the best performance for our HPL benchmark, we used the ATLAS 3.10 release. The 3.10 release has code to detect and optimize for ARM hardware. Specifically the code has GEMM routines implemented in assembly code using NEON SIMD instructions contributed by Vesperix Corporation. The code also has the option of using GEMM routines implemented in assembly code that uses VFP instructions. The library also has ARM Cortex-A9 specific defaults and supports Linux distributions that use the hard-float ABI [54].

The HPL benchmark was run on 2, 4, 6, 8, 10 and 12 cores on the Pandaboard cluster with the problem size scaled to 80% of the amount of memory available. Various block sizes were experimented with and the optimal block size was determined as 128.

Number of cores	Problem size / Block size	Performance (GFLOPS)	Avg. Power (Watts)	PPW (GFLOP/Watt)
2	9984 / 128	1.601	5.496	0.2913
4	14592 / 128	2.618	11.06	0.2367
6	17920 / 128	3.641	17.304	0.2104
8	20608 / 128	4.748	23.18	0.2048
10	23168 / 128	5.833	29.256	0.1993
12	25344 / 128	6.484	34.535	0.1877

Table 7.3: HPL results on Pandaboard cluster

HPL	Number of cores	Problem size / Block size	Performance (GFLOPS)
Unoptimized	2	9984 / 128	1.590e-01
Optimized	2	9984 / 128	1.601e+00
Unoptimized	12	25344 / 128	8.770e-01
Optimized	12	25344 / 128	6.484e+00

Table 7.4: Comparison of HPL performance using unoptimized and optimized BLAS and LAPACK

We see from Table 7.4 that there is approximately a 10 times increase in performance between the unoptimized and optimized BLAS and LAPACK versions of the library for the HPL benchmark using 2 cores on a single node. Within a node the MPICH2 1.4.1 library uses shared memory for inter process communication and therefore there is very little overhead. Even when scaled to 12 cores (6 nodes), we see that there is approximately a 700% increase in performance between the optimized and unoptimized versions.

Figure 7.10: Scaling HPL on the Pandaboard cluster

As shown in Figure 7.10 we see that HPL scales quite well as we increase the number of cores running the benchmark. We should also note here that the problem size is scaled along with the number of cores so that at least 80% of the available memory is utilized. The problem size can be input using the HPL.dat file which is read by the program during start up. The HPL.dat file also contains information such as block size, grid size and process grid ratio. The block size is used for granularity and data distribution. Hence a small block size would imply that for data distribution there is good load balance. However from a computation perspective, by choosing a smaller block size there is a risk of data not being reused by the systems cache hierarchy. Also there will be an increase in the number of messages communicated and for a cluster such as the Pandaboard cluster which uses 10/100 Mbps ethernet, this can be expensive. Hence by trial and error we can arrive at an optimal value for the block size and this is dependent on the computation/communication ratio of the system. The process grid ratio PxQ is chosen such that both P and Q values are as close as possible with Q always being the larger of the two.

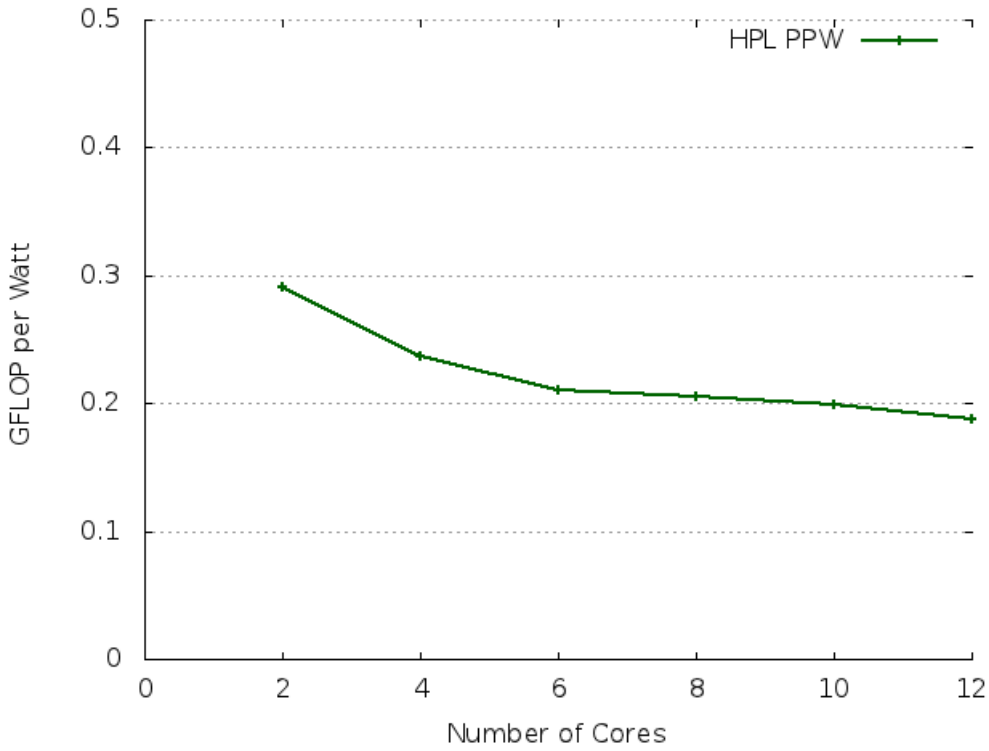

Figure 7.11: HPL Performance per watt

The performance per watt measurement taken on the Pandaboard cluster shows that as the number of nodes running HPL increases, the raw HPL performance obtained drops and the power consumption of the cluster increases. Thereby causing a decrease in the performance per watt metrics. We should note that when there is inter-node communication, the network controller is also drawing extra current in order to communicate the data though the Ethernet port. This is not the case when there is interprocess communication happening within the node as the MPICH2 library uses shared memory thereby bypassing the network controller. Hence future ARM CPUs should have higher core density to decrease the glue power of each node.

We tried running HPL with only one process per node so that only 1 core in a node is utilized. This is not a good option to get a reasonable performance per watt measurement since the other idle core (possibly running OS services) utilizes some of the total power and hence we get a decreased performance per watt ratio.

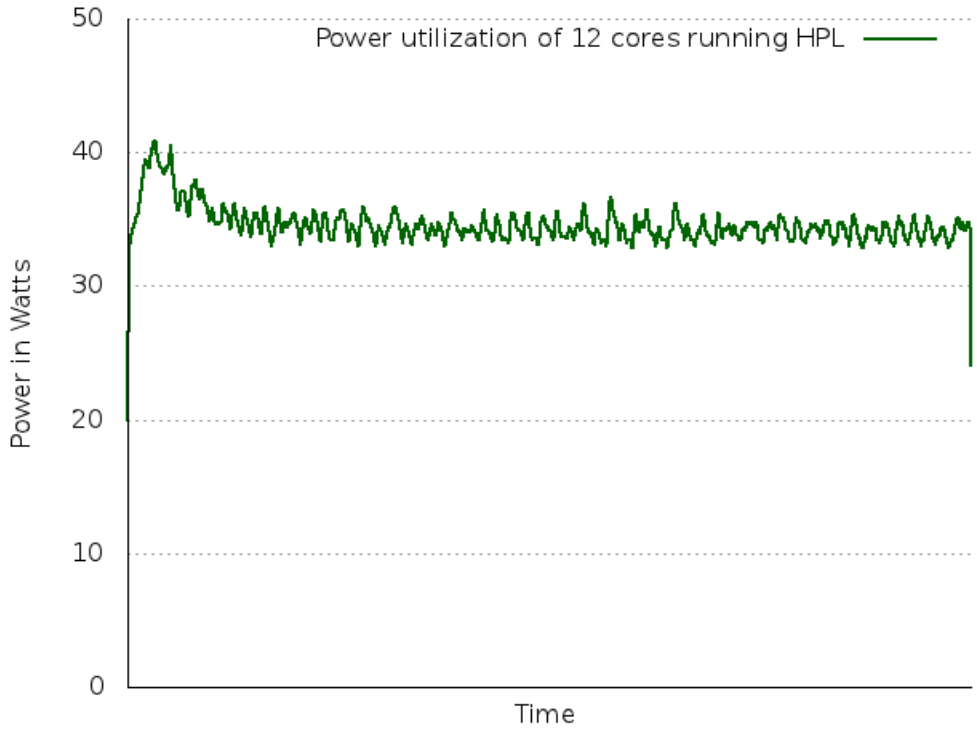

Figure 7.12: Power utilization of the 12 core Pandaboard cluster while running HPL

From the power consumption graph shown in Figure 7.12, we see that there is a surge is utilization of power at the beginning of the HPL benchmark. During the start up of HPL, there is initialization and setup activity taking place. Also there are communication patterns using broadcast messages to all processes within the cluster. Eventually the power consumption settles at around 34 Watts. The peaks in the power consumption graph could represent phases in the benchmark where there is communication activity going on between the processes.

7.5 Linpack

The Linpack benchmark was run on the Raspberry Pi boards running the Debian armel and Raspbian Linux versions. The array size used was 200x200. The benchmark calculates the average rolled and unrolled double precision floating point performance. This benchmark is of interest to us mainly to compare the performance of the Raspberry Pi when it uses the soft-float abi (Debian armel) versus the hard-float abi (Raspbian).

Operating System	KFLOPS	Avg. Power (Watts)	KFLOP/Watt	Compiler Flags
Debian armel	8199.005	2.165	3787.069	-O3
Debian armel	8719.577	2.17	4018.238	-O3 -fprefetch-loop-arrays
Raspbian	42256.41	2.18	19383.674	-O3
Raspbian	43084.967	2.18	19763.746	-O3 -mfloat-abi=hard -mfpu=vfp
Raspbian	53593.496	2.16	24811.803	-O3 -fprefetch-loop-arrays -mfloat-abi=hard -mfpu=vfp

Table 7.5: Linpack results on the Raspberry Pi

We see from Table 7.5 that the Raspberry Pi running Debian armel is only able to achieve a mere 8.7 MFLOPS when compiled with the optimization level 3 and the prefecth flag. In our analysis of the STREAM benchmark we observed that since the Debian armel does not use the L2 cache, there is very little performance gain when data is prefetched from memory. Also the Debian armel uses the soft-float ABI where the compiler emulates floating point arithmetic by means of library calls in software. This way of performing floating-point arithmetic can be slow and hence there is a loss of performance especially since the ARM11 CPU does have a VFP unit.

In the Raspbian version, the compiler is able to generate VFP hardware floating-point instructions using the VFP ABI. With this ABI, the VFP registers are used to pass function arguments and return values, thereby resulting in faster floating-point code. The Raspbian OS also utilizes the 128 KB L2 cache and hence adding the prefetch flag during compilation, the compiler is able to prefetch data from memory and store it in the cache for future access. From Figure 7.13 we see that the Raspbian version delivers approximately 6 times better floating-point performance than the Debian armel version.

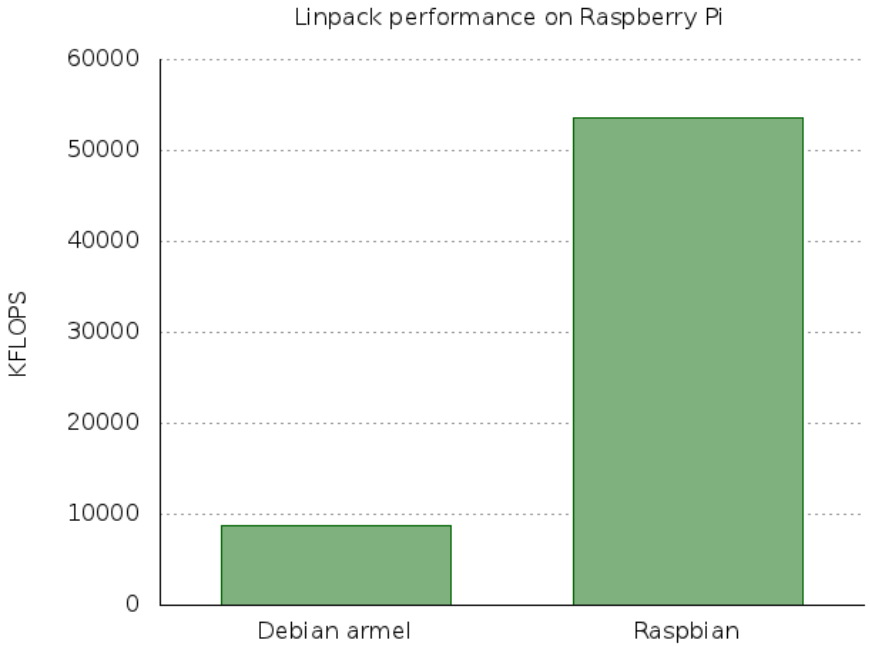

Figure 7.13: Linpack performance on Raspberry Pi

It is also interesting to find that the Raspberry Pi running Raspbian uses the same amount of power as the Raspberry Pi running Debian armel while giving 6 times the performance. Hence the performance per watt obtained on the Raspbian is 6 times the performance per watt obtained on the Debian armel. This clearly shows the importance of the hardware floating-point unit and the L2 cache.

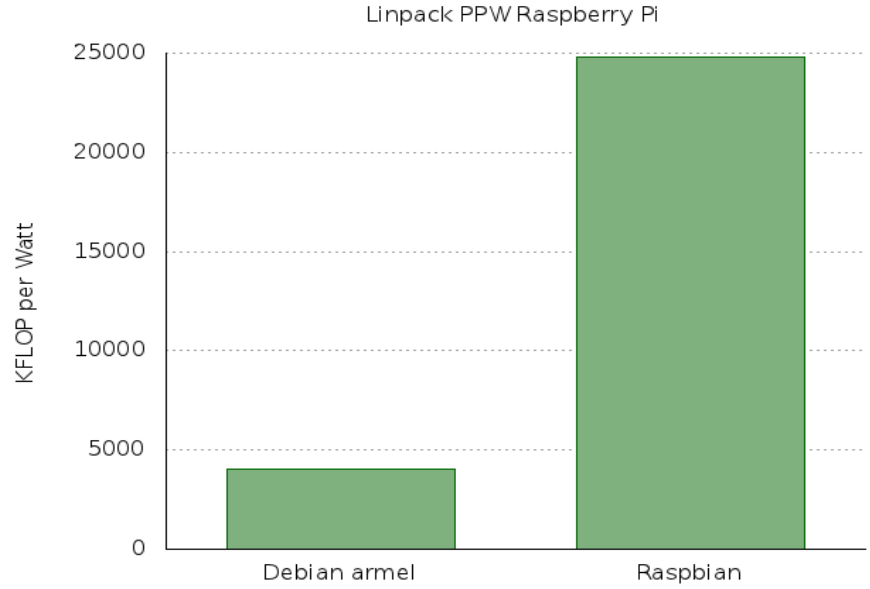

Figure 7.13: Linpack performance per watt on Raspberry Pi

7.6 Ping Pong

The ping pong benchmark was implemented in C using MPI to exchange a message between 2 processes for a certain number of iterations. The benchmark measures the latency and bandwidth of MPI communication. We executed the benchmark for both inter-node as well and intra-node MPI communication. The MPICH2 1.4.1 library uses the Nemesis channel which supports multiple network protocols and uses shared memory communication within the node.

From Figure 7.14 we see that the benchmark begins by exchanging an 8 byte message and with each step the message size doubles. There is a gradual increase in the latency until the message size is 64 KB. After this point, there is a steep increase in the communication latency. Similarly the bandwidth keeps increasing till the 64 KB mark. However when the message size reaches 128 KB, there is a drop in the bandwidth before resuming increasing again. The reason for this behaviour is due to the switch in messaging protocol at 128KB message size.

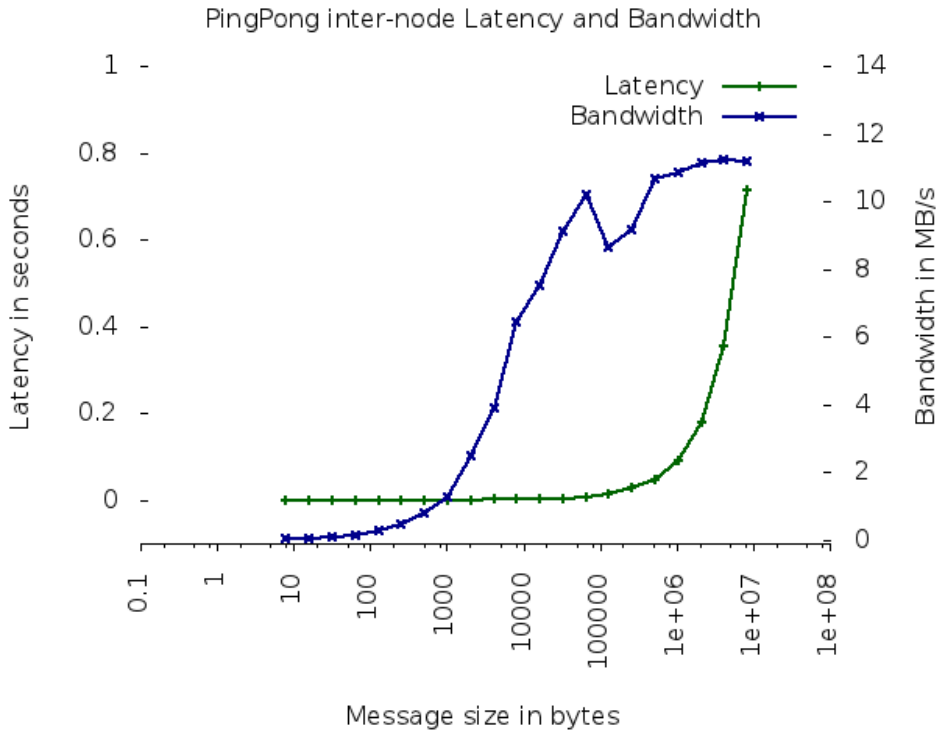

Figure 7.14: Ping Pong inter-node latency and bandwidth

MPICH2 uses 2 protocols Eager and Rendezvous which are internal to MPI in order to accomplish message delivery. The Eager protocol is used for small messages up to a certain size. It assumes that a message sent from one process can be stored on the receiving process and hence does not wait for an acknowledgement from a matching receive. The receiving process buffers the message upon arrival even if a receive has not been posted. The Rendezvous protocol requires an acknowledgement from the receiving process in order for the send operation to complete. This protocol is generally used for large messages since there cannot be any assumptions made regarding the buffer space available on the receiving process [55].

The advantage of the Eager protocol is that there is no acknowledgement required from the receiving process and therefore there is no synchronization. The disadvantage is that it is not scalable if there are many senders as there is significant buffering required. Also buffering can take up CPU cycles on the receivers side. Thee Rendezvous protocol on the other hand is scalable as there is no buffer space needed for the data payload on the receivers side. However, since there is handshaking taking place between the sender and receiver we get higher latency. The Eager limit message size in the MPICH2 1.4.1 has been set to 128 KB [56]. When the message size is greater than or equal to 128 KB, the MPI runtime automatically switches to using the Rendezvous protocol.

In Figure 7.15 we see the ping pong results for the intra-node latency and bandwidth. Since the MPI library uses shared memory for intra-node communication, the latency is extremely low. Also we are able to reach a maximum bandwidth of almost 1.2 GB/s. If future versions of the Pandaboard support gigabit Ethernet, this difference in bandwidth between the inter and intra node communication will be greatly reduced. MPICH2 uses the TCP/IP for inter-node communication and can add to the communication latency. Using a protocol such as Open-MX could reduce this latency for inter-node communication further [39].

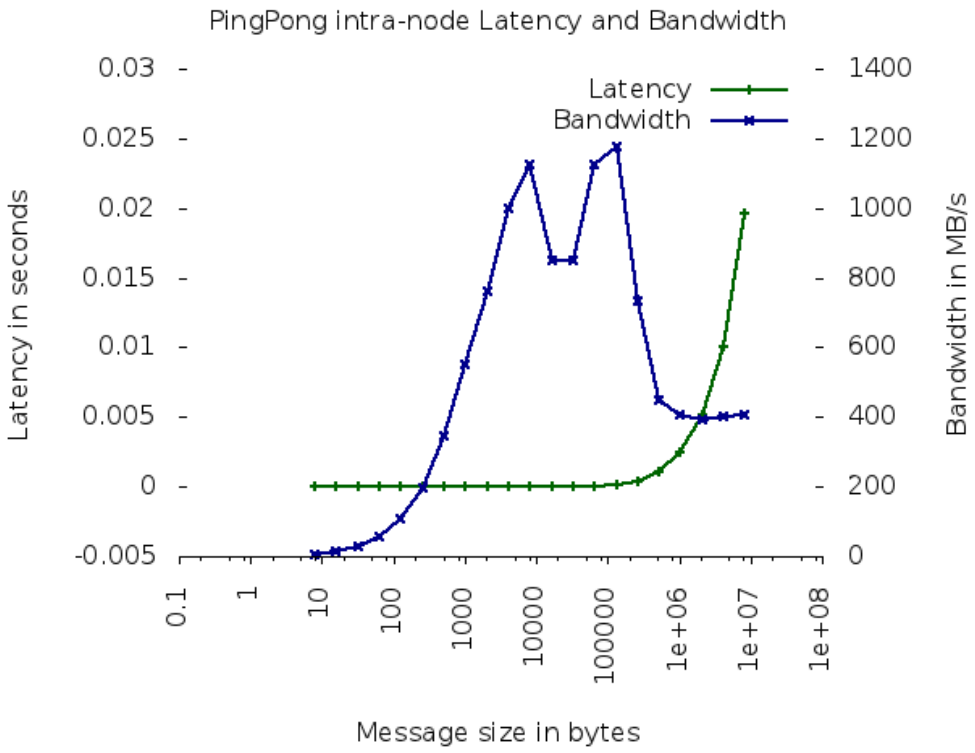

Figure 7.15: Ping Pong intra-node latency and bandwidth

7.7 NAS Parallel Benchmarks

Two benchmarks from the NAS parallel benchmark suite were run on the Pandaboard cluster. In the sections below, we will look at the results produced by these benchmarks.

7.7.1 Embarrassingly Parallel (EP)

Number of Processes	Problem Size	Mop/s	Avg. Power (Watts)	PPW (Mop/Watt)
2	536870912	10.24	4.99	2.052
4	2147483648	20.38	11.13	1.831
6	2147483648	30.42	16.378	1.857
8	2147483648	40.49	21.304	1.900
10	8589934592	50.77	28.893	1.757
12	8589934592	60.61	35.422	1.711

Table 7.6: Embarrassingly Parallel benchmark results on the Pandaboard cluster

The EP benchmark was run on the Pandaboard cluster to measure the performance per watt for problems where there is very little communication between the MPI processes in the cluster. As shown in Table 7.6 the problem was scaled up to 12 cores with an increase in problem size at a certain stage. This was done to ensure that the benchmark ran for at least 1 minute.

From Figure 7.16 we see that the problem scales quite well on the Pandaboard cluster. For 2 cores, we get are able to perform approximately 10 million operations per second and when scaled up to 12 cores we get just over 60 million operations per second . Hence the performance obtained is directly proportional to the number of cores used. This is mainly because there is good load balance across the various cores in the cluster and also very little communication between them. Hence

increasing the number of cores while increasing the problem size gives us almost linear scaling.

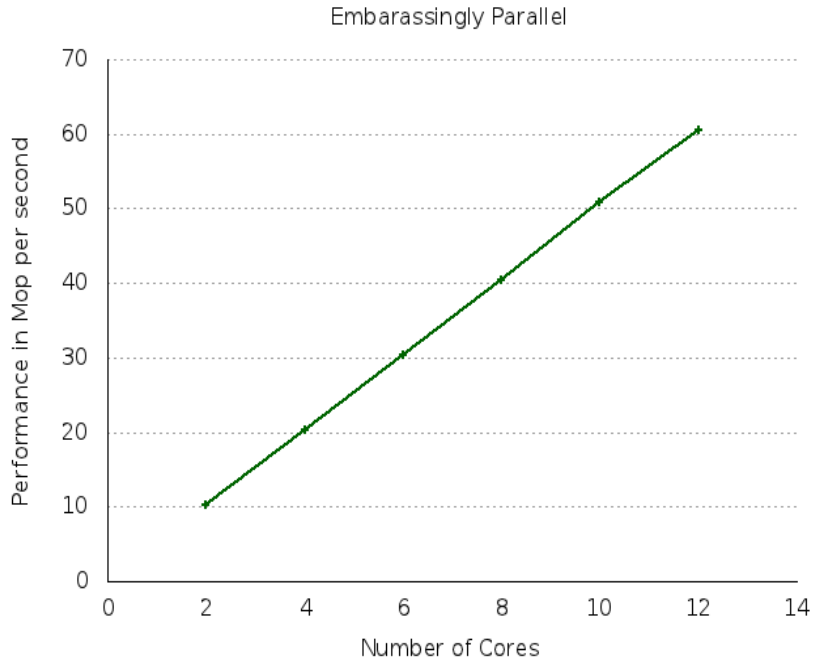

Figure 7.16: Embarrassingly Parallel benchmark

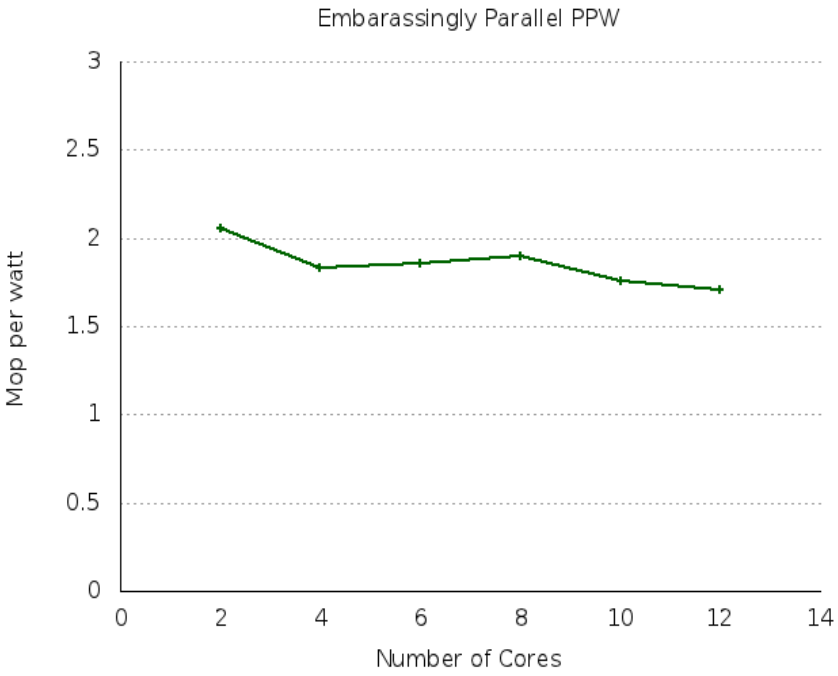

Figure 7.17: Embarrassingly Parallel performance per watt

We see from Figure 7.17 that scaling the EP benchmark to more cores leads to a decrease in the performance per watt measured. This is mainly because even though the performance scales linearly, the power used does not. For each node (2 cores) being added to the benchmark measurement, the average power used does not increase by 4.99 watts (average power used for 2 cores). Instead there is some extra usage of power. If the power used also scaled linearly, we could have a linearly scaling PPW reading which is the ideal scenario.

7.7.2 3D Fast Fourier Transform (FT)

The FFT benchmark was run on the cluster to mainly test the raw performance and performance per watt when there is a high amount of network activity. The FFT benchmark uses MPI all to all communication pattern.

Number of processes	Problem Size	Mop/s	Avg. Power (Watts)	PPW (Mop/Watt)
1	256x256x128	140.47	3.654	38.442
2	256x256x128	209.46	4.730	44.283
4	512x256x256	102.35	10.84	9.441
8	512x256x256	243.11	23.89	10.172

Table 7.7: 3D FFT benchmark results on the Pandaboard cluster

The number of processes used to run the 3D FFT benchmark should be a power of 2. Hence as shown in Table 7.7, we see that this benchmark was executed on up to 8 cores in the cluster. The problem size was also scaled along with the number of cores in order to ensure that benchmark runs for a reasonably long period of time.

We see from Figure 7.18 that at 4 cores, there is a drop in performance of the FFT benchmark. The number of operations per second obtained is less than that obtained when the benchmark is run on 1 and 2 cores. This is mainly because for the 1 and 2 core runs, the interprocess communication uses shared memory for all MPI all to all messages. Communication has much higher bandwidth and very low latency when

compared to inter-node TCP/IP based communication. The performance per watt measurement shown in Figure 7.19 illustrates the point that network communication on the Pandaboard cluster is expensive. For a cluster that does not have a high speed inter-connect, a problem involving little or no communication would be suitable.

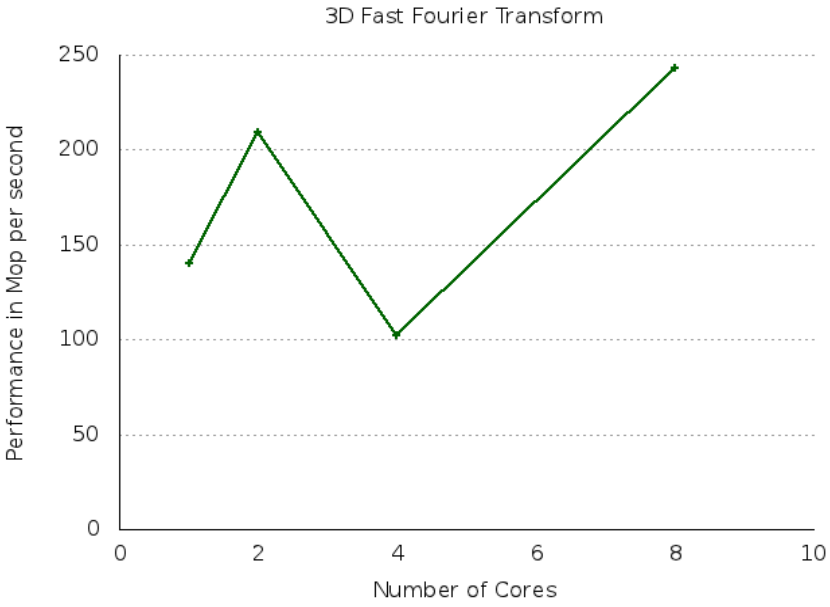

Figure 7.18: 3D Fast Fourier Transform performance

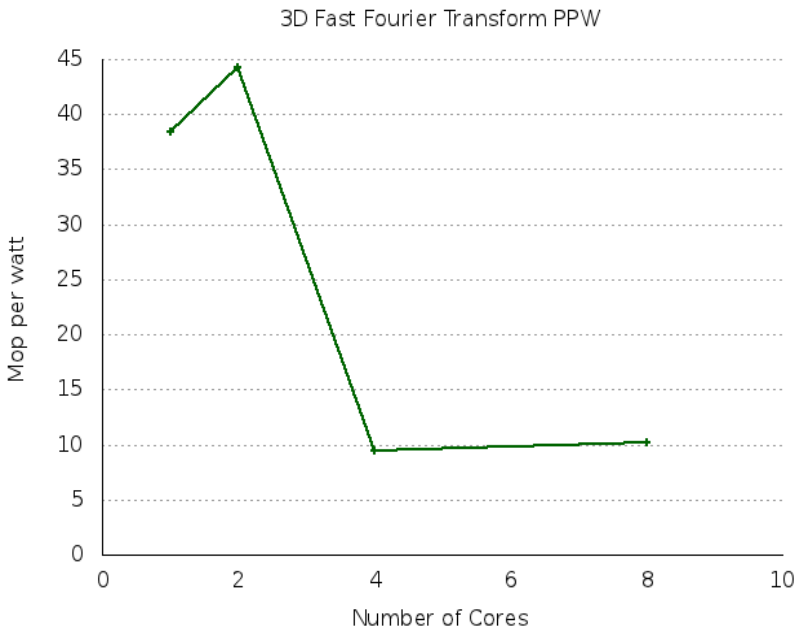

Figure 7.19: 3D Fast Fourier Transform performance per watt

8 Future Work

There is a lot of scope for further research and experimentation in the area of low power HPC. The ARM processors used in this project were 32-bit, had a much lower clock speed and memory capacity when compared to some of the commonly used processors in HPC such as Intel and AMD. Future ARM processors such as the Cortex-A15 and the newly released ARMv8 architecture are definite candidates to compete against the big players in HPC.

Future projects could use an ARM SoC that is based on the ARMv8 architecture. One possible candidate is Nvidia's Project Denver processors which are on track to hit the mobile market in 2013 [58]. Nvidia is also planning to release its Tegra 4 SoCs in 2013 which has a quad-core ARM Cortex-A15 processor. Future Pandaboard designs could use the OMAP5 SoCs from Texas Instruments which also use the ARM Cortex-A15 CPU.

ARM processor designs in the future will have higher core densities. As we move to 4, 8 and 16 cores on the chip, the glue power used by memory, network controller and disk/SD card become a small fraction of the total power used by the node. This scenario is ideally suited for projects like this one where the CPUs performance per watt metrics is of importance.

There is also scope for exploring the possibility of using the hybrid MPI-OpenMP programming model and compare its performance per watt metrics against a pure MPI implementation when the core count goes up.

The Ubuntu 12.10 release installed on the cluster was a development branch image. As a result there were no kernel headers available for this specific kernel version (3.4.0-201-omap4). Hence we were unable to compile and configure OpenMX on the Pandaboard ES. Once Ubuntu 12.10 is officially released, we could recompile MPICH2 with OpenMX support and benchmark the cluster. We should see some improvement in inter-node communication latency.

In this project we have mainly used industry standard benchmarks to test various performance aspects of the cluster. We can run real scientific applications such as Molecular Dynamics, simulations used in biology and chemistry on the Pandaboard cluster. This will give us a realistic scenario of using ARM processors in HPC.

9 Conclusions

ARM processors have created a lot of interest among HPC community due to its low power usage. This project attempts to use ARM based single board computers to build a cluster. The focus of the project is to measure the performance per watt of the ARM boards by running a variety of benchmarks. We were able to successfully build the cluster using open-source software tools and run benchmarks on the system.

The Pandaboard ES which uses the ARM Cortex-A9 processor turns out to be very energy-efficient for the performance it is able to deliver. This is evident from the performance per watt results obtained on the Pandaboard. The Pandaboard consistently outperforms machines like the Intel Atom and Intel Xeon when the metrics used is performance per watt. From a previous comparison study on low power HPC [37], we saw that for most benchmarks the ARM (Marvell 88F6281) processor gave better performance per watt results that the Intel Atom and Intel Xeon processors. The results obtained in this project show that processors with modern ARM architecture such as ARMv7 not only provide very good raw performance, they also use lesser energy than older ARM architectures, thereby widening the performance per watt gap.

We executed the benchmarks CoreMark, STREAM and HPL by compiling the source code using various flags and optimization levels. There was a significant improvement in performance when flags to unroll loops, vectorize code and prefetch data from memory were added, depending on the operations performed in the benchmarks. The ARM Cortex-A9 processor provides NEON SIMD instructions. Although the NEON instructions supports only 32-bit floating point SIMD operations, we saw a significant improvement in performance between the ATLAS implementation that used NEON SIMD to implement GEMM routines versus the unoptimized ATLAS version. This reinforces the point that tuning the code for the architecture and usage of compiler optimization flags is crucial to obtain high performance.

The Raspberry Pi boards were benchmarked using the Debian armel and Raspbian versions of Linux. The Raspbian uses the hard-float ABI and delivers much higher floating point performance when compared to the Debian armel which uses the soft-float ABI. The Raspbian also outperformed the Debian armel when the STREAM benchmark was executed on the Raspberry Pi. The reason for this is because the Raspbian utilizes the 128 KB L2 cache while the Debian armel does not. These results are indicative of the role the operating system plays in obtain performance from the hardware.

When the single threaded CoreMark performance per watt metrics is compared between the Raspberry Pi and the Pandaboard ES, we find that the Raspberry Pi is not too far behind the Pandaboard even though the Pandaboard performs 2.5 times better in terms of raw performance. This is mainly due to the lower power consumption of the Raspberry Pi during peak load. The Raspberry Pi also uses much lower power when running the STREAM benchmark compared to the Pandaboard ES.

The ARM architecture roadmap clearly shows the path that is being taken in the coming years. ARM currently dominates the mobile market and is poised to enter the high performance server market. In order to find a solution to the high power consumption of HPC machines, we need to make changes in several layers of the hardware and software stack. ARM based processor technology could lead the way by starting at the bottom most level which is the hardware.

Appendix A Benchmark Sample Results

A-1 Sample output of CoreMark benchmark on the Pandaboard ES

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 309747
Total time (secs): 309.747000
Iterations/Sec   : 6456.882553
Iterations       : 2000000
Compiler version : GCC4.7.1
Compiler flags   : -O3 -funroll-loops -g -DMULTITHREAD=2 -DUSE_PTHREAD
-DPERFORMANCE_RUN=1  -lrt
Parallel PThreads : 2
Memory location  : Please put data memory location here
(e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[1]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
[1]crcmatrix     : 0x1fd7
[0]crcstate      : 0x8e3a
[1]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
[1]crcfinal      : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 6456.882553 / GCC4.7.1 -O3 -funroll-loops -g
-DMULTITHREAD=2 -DUSE_PTHREAD -DPERFORMANCE_RUN=1 -lrt /
Heap / 2:PThreads
```

A-2 Sample output of CoreMark benchmark on the Raspberry Pi

```
2K performance run parameters for coremark.
CoreMark Size    : 666
Total ticks      : 711340
Total time (secs): 711.340000
Iterations/Sec   : 1405.797509
Iterations       : 1000000
Compiler version : GCC4.4.5
Compiler flags   : -O3 -funroll-loops -DPERFORMANCE_RUN=1  -lrt
Memory location  : Please put data memory location here
(e.g. code in flash, data on heap etc)
seedcrc          : 0xe9f5
[0]crclist       : 0xe714
[0]crcmatrix     : 0x1fd7
```

```
[0]crcstate      : 0x8e3a
[0]crcfinal      : 0x988c
Correct operation validated. See readme.txt for run and reporting rules.
CoreMark 1.0 : 1405.797509 / GCC4.4.5 -O3 -funroll-loops
-DPERFORMANCE_RUN=1  -lrt / Heap
```

A-3 Sample output of STREAM benchmark on the Pandaboard ES

```
-------------------------------------------------------------
STREAM version $Revision: 5.9 $
-------------------------------------------------------------
This system uses 8 bytes per DOUBLE PRECISION word.
-------------------------------------------------------------
Array size = 32768000, Offset = 0
Total memory required = 750.0 MB.
Each test is run 10 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Number of Threads requested = 2
-------------------------------------------------------------
Printing one line per active thread....
Printing one line per active thread....
-------------------------------------------------------------
Your clock granularity/precision appears to be 29 microseconds.
Each test below will take on the order of 529326 microseconds.
   (= 18252 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:       1209.6816       0.4346       0.4334       0.4357
Scale:      1227.7488       0.4284       0.4270       0.4295
Add:         949.2687       0.8305       0.8285       0.8340
Triad:      1251.9332       0.6300       0.6282       0.6372
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
```

A-4 Sample output of STREAM benchmark on the Raspberry Pi

```
-------------------------------------------------------------
STREAM version $Revision: 5.9 $
-------------------------------------------------------------
This system uses 8 bytes per DOUBLE PRECISION word.
-------------------------------------------------------------
Array size = 6553600, Offset = 0
Total memory required = 150.0 MB.
Each test is run 10 times, but only
the *best* time for each is used.
-------------------------------------------------------------
Printing one line per active thread....
-------------------------------------------------------------
Your clock granularity/precision appears to be 2 microseconds.
Each test below will take on the order of 257426 microseconds.
   (= 128713 clock ticks)
Increase the size of the arrays if this shows that
you are not getting at least 20 clock ticks per test.
-------------------------------------------------------------
WARNING -- The above is only a rough guideline.
For best results, please be sure you know the
precision of your system timer.
-------------------------------------------------------------
Function      Rate (MB/s)   Avg time     Min time     Max time
Copy:         375.5592       0.2801       0.2792       0.2814
Scale:        229.7664       0.4598       0.4564       0.4780
Add:          193.4280       0.8147       0.8132       0.8153
Triad:        239.2993       0.6587       0.6573       0.6596
-------------------------------------------------------------
Solution Validates
-------------------------------------------------------------
```

A-5 Sample output of HPL benchmark on the Pandaboard ES for 12 cores using unoptimized BLAS and LAPACK

```
================================================================================
HPLinpack 2.0  --  High-Performance Linpack benchmark  --  September 10, 2008
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================
An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:
N      :   25344
NB     :     128
PMAP   : Row-major process mapping
P      :       3
Q      :       4
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
--------------------------------------------------------------------------------
- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be               1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0
================================================================================
```

```
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4       25344   128     3     4          12376.00              8.770e-01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0039910 ..... PASSED
================================================================================
Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------
End of Tests.
================================================================================
```

A-6 Sample output of HPL benchmark on the Pandaboard ES for 2 cores using ATLAS 3.10 and compiled with flags for ARM

```
================================================================================
HPLinpack 2.0  --  High-Performance Linpack benchmark  --   September 10, 2008
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================
An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:
N      :    9984
NB     :     128
PMAP   : Row-major process mapping
P      :       1
Q      :       2
PFACT  :   Right
```

```
NBMIN :      4
NDIV  :      2
RFACT :   Crout
BCAST :  1ringM
DEPTH :      1
SWAP  : Mix (threshold = 64)
L1    : transposed form
U     : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
--------------------------------------------------------------------------------
- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
     ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be             1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4       9984   128     1     2             414.41             1.601e+00
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0013786 ..... PASSED
================================================================================
Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------
End of Tests.
================================================================================
```

A-7 Sample output of HPL benchmark on the Pandaboard ES for 12 cores using ATLAS 3.10 and compiled with flags for ARM

```
================================================================================
HPLinpack 2.0  --  High-Performance Linpack benchmark  --   September 10, 2008
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
```

```
================================================================================
An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:
N      :   25344
NB     :     128
PMAP   : Row-major process mapping
P      :       3
Q      :       4
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
--------------------------------------------------------------------------------
- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be            1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4       25344   128     3     4            1673.99              6.484e+00
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0009402 ..... PASSED
================================================================================
Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
```

```
----------------------------------------------------------------------------
End of Tests.
============================================================================
```

A-8 Sample output of Ping Pong benchmark on the Pandaboard ES using 2 nodes.

Message Size (Bytes)	Latency (s)	Bandwidth (MB/s)
8	0.000392	0.019458
16	0.000389	0.039207
32	0.000396	0.077099
64	0.000420	0.145339
128	0.000463	0.263774
256	0.000545	0.447762
512	0.000618	0.790108
1024	0.000771	1.265843
2048	0.000783	2.494470
4096	0.001004	3.891370
8192	0.001218	6.415783
16384	0.002074	7.532569
32768	0.003435	9.097896
65536	0.006141	10.177662
131072	0.014470	8.638434
262144	0.027330	9.147350
524288	0.046816	10.680017
1048576	0.092237	10.841607
2097152	0.179291	11.155026
4194304	0.356498	11.220265
8388608	0.714353	11.198949

A-8 Sample output of embarrassingly parallelism benchmark on the Pandaboard ES for 12 cores

```
NAS Parallel Benchmarks 3.2 -- EP Benchmark
Number of random numbers generated:    8589934592
Number of active processes:                  12
EP Benchmark Results:
CPU Time =  141.7163
N = 2^32
No. Gaussian Pairs =    3373275903.
Sums =     4.764367927993350D+04   -8.084072988045315D+04
Counts:
 0    1572172634.
 1    1501108549.
 2     281805648.
 3      17761221.
 4        424017.
 5          3821.
 6            13.
 7             0.
 8             0.
 9             0.

EP Benchmark Completed.
Class           =                    C
Size            =           8589934592
Iterations      =                    0
Time in seconds =               141.72
Total processes =                   12
Compiled procs  =                   12
Mop/s total     =                60.61
Mop/s/process   =                 5.05
Operation type  = Random numbers generated
Verification    =           SUCCESSFUL
Version         =                  3.2
Compile date    =           07 Aug 2012

Compile options:
   MPIF77       = mpif77
   FLINK        = $(MPIF77)
   FMPI_LIB     = -L/usr/local/mpi/lib -lfmpich
   FMPI_INC     = -I/usr/local/mpi/include
   FFLAGS       = -O3
   FLINKFLAGS   = -O3
   RAND         = randi8
```

```
Please send the results of this run to:
NPB Development Team
Internet: npb@nas.nasa.gov
If email is not available, send this to:
MS T27A-1
NASA Ames Research Center
Moffett Field, CA  94035-1000
Fax: 650-604-3957
```

A-8 Sample output of 3D FFT benchmark on the Pandaboard ES for 8 cores

```
NAS Parallel Benchmarks 3.2 -- FT Benchmark
No input file inputft.data. Using compiled defaults
Size                :  512x 256x 256
Iterations          :          20
Number of processes :           8
Processor array     :       1x  8
Layout type         :          1D

T =    1     Checksum =    5.177643571579D+02    5.077803458597D+02
T =    2     Checksum =    5.154521291263D+02    5.088249431599D+02
T =    3     Checksum =    5.146409228650D+02    5.096208912659D+02
T =    4     Checksum =    5.142378756213D+02    5.101023387619D+02
T =    5     Checksum =    5.139626667737D+02    5.103976610618D+02
T =    6     Checksum =    5.137423460082D+02    5.105948019802D+02
T =    7     Checksum =    5.135547056878D+02    5.107404165783D+02
T =    8     Checksum =    5.133910925467D+02    5.108576573661D+02
T =    9     Checksum =    5.132470705390D+02    5.109577278523D+02
T =   10     Checksum =    5.131197729984D+02    5.110460304483D+02
T =   11     Checksum =    5.130070319283D+02    5.111252433800D+02
T =   12     Checksum =    5.129070537032D+02    5.111968077719D+02
T =   13     Checksum =    5.128182883503D+02    5.112616233064D+02
T =   14     Checksum =    5.127393733383D+02    5.113203605551D+02
T =   15     Checksum =    5.126691062021D+02    5.113735928093D+02
T =   16     Checksum =    5.126064276005D+02    5.114218460548D+02
T =   17     Checksum =    5.125504076570D+02    5.114656139760D+02
T =   18     Checksum =    5.125002331721D+02    5.115053595966D+0
T =   19     Checksum =    5.124551951846D+02    5.115415130407D+02
T =   20     Checksum =    5.124146770029D+02    5.115744692211D+02

Result verification successful
class = B
FT Benchmark Completed.
Class           =                        B
Size            =           512x 256x 256
```

```
Iterations      =                20
Time in seconds =            378.65
Total processes =                 8
Compiled procs  =                 8
Mop/s total     =            243.11
Mop/s/process   =             30.39
Operation type  =       floating point
Verification    =          SUCCESSFUL
Version         =               3.2
Compile date    =        07 Aug 2012

Compile options:
   MPIF77       = mpif77
   FLINK        = $(MPIF77)
   FMPI_LIB     = -L/usr/local/mpi/lib -lfmpich
   FMPI_INC     = -I/usr/local/mpi/include
   FFLAGS       = -O3
   FLINKFLAGS   = -O3
   RAND         = randi8

Please send the results of this run to:
NPB Development Team
Internet: npb@nas.nasa.gov
If email is not available, send this to:
MS T27A-1
NASA Ames Research Center
Moffett Field, CA  94035-1000
Fax: 650-604-3957
```

Appendix B Scripts

B-1 execAll.pl

```
# Author: Nikilesh Balakrishnan
# Description: Script to execute commands from
#                   the master on all Pandaboard ES boards
#!/usr/bin/perl -w
use strict;

if($#ARGV != 0)
{
        print "Usage: ./execAll <Command>\n";
        exit(1);
}
my $command = $ARGV[0];

my @machineList = ();
push(@machineList, "panda1");
push(@machineList, "panda2");
push(@machineList, "panda3");
push(@machineList, "panda4");
push(@machineList, "panda5");
push(@machineList, "panda6");

&execCmd();

sub execCmd
{
        foreach my $dest (@machineList)
        {
                print "Executing command $command on $dest\n";
                my $cmd = "ssh $dest '$command'";
                my $ret = `$cmd`;
                die "Error: $?" if($? != 0);

                print "Finished executing command $command on $dest\n";
                $ret = &trim($ret);
                next if($ret eq "");

                print "Output: \n$ret\n";
        }
}

sub trim
{
```

```
	my $string = shift;
	$string =~ s/^\s+//;
	$string =~ s/\s+$//;
	return $string;
}
```

B-2 getAvg.pl

```
# Author: Nikilesh Balakrishnan
# Description: Script that calculates the average power used
#!/usr/local/bin/perl -w

use strict;
my $count = 0;
my $totalWatts = 0;

while(<>)
{
	chomp;
	my ($first, $watt) = split(/ /,$_);

	++$count;
	$totalWatts += $watt;
}

my $avg = $totalWatts / $count;
print "total power: $totalWatts, count: $count, Avg power: $avg\n";
```

B-3 initialize.sh

```
# Author: Nikilesh Balakrishnan
# Description: Script to initialize the pandaboard when it boots up
sudo service idmapd stop
sudo service cron stop
sudo service statd stop
sudo service atd stop
sudo service upstart-socket-bridge stop
sudo service upstart-udev-bridge stop
sudo ntpdate -u -b uk.pool.ntp.org
sudo pbs_mom
sudo cpufreq-set -g performance
```

10 Bibliography

[1] HPCWire: Arm Yourselves for Exascale, Part 1. Online at http://www.hpcwire.com/hpcwire/2011-11-09/arm_yourselves_for_exascale,_part_1.html (referenced 07/18/2012).

[2] Performance per watt - Wikipedia. Online at http://en.wikipedia.org/wiki/Performance_per_watt (referenced 07/18/2012).

[3] Top500 Supercomputing Sites. Online at http://www.top500.org/list/2012/06/100 (referenced 07/21/2012).

[4] The Green500 List :: Environmentally Responsible Supercomputing :: The Green500 List June 2012. Online at http://www.green500.org/lists/2012/06/top/list.php?from=1&to=100 (referenced on 07/21/2012).

[5] Moore's law – Wikipedia, the free encyclopedia. Online at http://en.wikipedia.org/wiki/Moores_law (referenced on 07/26/2012).

[6] Exascale Computing Study: Technology Challenges in Achieving Exascale Systems. (DARPA)

[7] Why CPU Frequency Stalled – IEEE Spectrum. Online at http://spectrum.ieee.org/computing/hardware/why-cpu-frequency-stalled (reference on 07/27/2012).

[8] Computational efficiency for CPUs and GPUs in 2012. Online at http://www.realworldtech.com/compute-efficiency-2012/ (referenced on 07/30/2012).

[9] HPCWire: The 2012 Performance per watt wars. Online at http://www.hpcwire.com/hpcwire/2012-07-26/the_2012_performance_per_watt_wars.html (referenced on 07/30/2012)

[10] The future of AMD's Fusion APUs: Kaveri will fully share memory between CPU and GPU. Online at http://www.extremetech.com/computing/130939-the-future-of-amds-fusion-

apus-kaveri-will-fully-share-memory-between-cpu-and-gpu (referenced on 08/01/2012)

[11] Intel MIC – Wikipedia, the free encyclopedia. Online at http://en.wikipedia.org/wiki/Intel_MIC (referenced on 08/01/2012).

[12] Tim Mattson, Intel Labs. *Many core processors at Intel: lessons learned and a view of the future.*

[13] AnandTech – Intel Announces Xeon Phi Family of Co-Processors – MIC Goes Retail. Online at http://www.anandtech.com/show/6017/intel-announces-xeon-phi-family-of-coprocessors-mic-goes-retail/ (referenced on 08/01/2012).

[14] ARM, Windows 8 to Power Future Notebooks, says IHS. Online at http://it.tmcnet.com/topics/it/articles/201819-arm-windows-8-power-future-notebooks-says-ihs.htm (referenced on 08/02/2012).

[15] Richard Grisenthwaite, Lead Architect and Fello, ARM. *ARMv8 Technology Preview*

[16] Calxeda EnergyCore. Online at http://www.calxeda.com/wp-content/uploads/2012/06/ECX1000-Product-Brief-612.pdf (referenced on 08/02/2012)

[17] "Project Denver" Processor to Usher in New Era of Computing – Nvidia. Online at http://blogs.nvidia.com/2011/01/project-denver-processor-to-usher-in-new-era-of-computing/ (referenced on 08/03/2012).

[18] TMS320C66x multicore DSPs for high-performance computing. Online at http://www.ti.com/lit/ml/sprt619/sprt619.pdf (referenced on 08/03/2012).

[19] OMAP4460 Pandaboard ES System Reference Manual. Online at http://pandaboard.org/sites/default/files/board_reference/pandaboard-es-b/panda-es-b-manual.pdf (referenced on 08/04/2012).

[20] Cortex-A9 Processor – ARM. Online at http://arm.com/products/processors/cortex-a/cortex-a9.php (referenced on

08/04/2012).

[21] ARM Information Center. Online at http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0409f/Chdceejc.html (referenced on 08/04/2012).

[22] ARM Architecture – Wikipedia. Online at http://en.wikipedia.org/wiki/ARM_architecture#Floating-point_.28VFP.29 (Referenced on 08/04/2012).

[23] Rpi Hardware – eLinux.org. Online at http://elinux.org/RPi_Hardware (Referenced on 08/04/2012).

[24] ARM1176JZF-S, Revision: r0p7. Technical Reference Manual, ARM. Findings pp-18-2.

[25] The Graph 500 List. Online at http://www.graph500.org/ (Referenced on 08/06/2012).

[26] *The Blue Gene/Q Compute Chip*. Rudd Haring / IBM Blue Gene Team.

[27] IBM plants transactional memory in CPU. Online at http://www.eetimes.com/electronics-news/4218914/IBM-plants-transactional-memory-in-CPU (Referenced on 08/06/2012).

[28] IBM Blue Gene/Q supercomputer delivers petascale computing for high-performance computing applications. Online at http://www-01.ibm.com/common/ssi/rep_ca/8/897/ENUS112-028/ENUS112-028.PDF (Referenced on 08/06/2012).

[29] *Mont-Blanc. European scalable and power efficient HPC platform based on low-power embedded technology.* Alex Ramirez, Barcelona Supercomputing Center, Technical Coordinator.

[30] *The Mont-Blanc architecture, BoF session ISC 2012*. Alex Ramirez.

[31] The OmpSs programming model | Programming models @ BSC. Online at http://pm.bsc.es/ompss (Referenced on 08/08/2012).

[32] [Phoronix] 12-Core ARM Cluster Benchmarked Against Interl Atom, Ivy

Bridge and AMD Fusion. Online at http://www.phoronix.com/scan.php?page=article&item=phoronix_effimass_cluster&num=1 (Referenced on 08/08/2012). pp 1-16.

[33] Software Pandaboard. Online at http://pandaboard.org/content/resources/software (Referenced on 08/09/2012).

[34] [Phoronix] Building A 96-Core Ubuntu ARM Solar-Powered Cluster. Online at http://www.phoronix.com/scan.php?page=article&item=mit_cluster_build&num=1 (Referenced on 08/08/2012).

[35] *Towards Fault-Tolerant Energy-Efficient High Performance Computing in the Cloud*. Kurt L.Keville, Rohan Garg, David J.Yates, Kapil Arya, Gene Cooperman.

[36] F2c. Online at http://www.netlib.org/f2c/ (Referenced on 08/08/2012).

[37] *Low-Power High Performance Computing*. Panagiotis Kritikakos, EPCC.

[38] MPICH2: High Performance and Widely Portable MPI. Online at http://www.mcs.anl.gov/research/projects/mpich2/ (Referenced on 08/10/2012).

[39] Open-MX: Myrinet Express over Generic Ethernet Hardware. Online at http://open-mx.gforge.inria.fr/ (Referenced on 08/10/2012).

[40] TORQUE Resource Manager. Online at http://www.adaptivecomputing.com/products/open-source/torque/ (Referenced on 08/10/2012).

[41] Cluster resources :: Products - Maui Cluster Scheduler. Online at http://www.clusterresources.com/products/maui-cluster-scheduler.php (Referenced on 08/10/2012).

[42] CoreMark an EEMBC Benchmark. Online at http://coremark.org/ (Referenced on 08/11/2012).

[43] CoreMark: A realistic way to benchmark CPU performance. Online at

http://www.eetimes.com/design/embedded/4212735/CoreMark--A-realistic-way-to-benchmark-CPU-performance?pageNumber=0 (Referenced on 08/11/2012). pp 1-3.

[44] STREAM Benchmark Reference Information. Online at https://www.cs.virginia.edu/stream/ref.html (Referenced on 08/11/2012).

[45] LINPACK. Online at http://www.netlib.org/linpack/ (Referenced on 08/11/2012).

[46] HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. Online at http://www.netlib.org/benchmark/hpl/ (Referenced on 08/11/2012).

[47] NAS Parallel Benchmarks. Online at http://www.nas.nasa.gov/publications/npb.html (Referenced on 08/11/2012).

[48] *THE NAS PARALLEL BENCHMARKS*. D Bailey, E Barszcz, J Barton, D Browning, R Carter, L Dagum, R Fatoohi, S Fineberg, P Frederickson, T Lasinski, R Schreiber, H Simon, V Venkatakrishnan and S Weeratunga. RNR Technical Report RNR-94-007, March 1994.

[49] Intel Power Gadget 2.0. Online at http://software.intel.com/en-us/articles/intel-power-gadget/ (Referenced on 08/12/2012).

[50] *Power Measurement Tutorial for the Green500 List*. R. Ge, X. Feng, H. Pyla, K. Cameron, W. Feng.

[51] CPU frequency and voltage scaling code in th**e** Linux(TM**)** kernel. Online at http://kernel.org/doc/Documentation/cpu-freq/governors.txt (Referenced on 08/13/2012).

[52] Optimize Options - Using the GNU Compiler Collection (GCC). Online at http://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc/Optimize-Options.html (Referenced on 08/14/2012).

[53] *LP-DDR2, The Next Generation Memory Technology*. Jaime Borras, Wireless Silicon Group, Inc.

[54] ATLAS-ARM FAQ. Online at http://www.vesperix.com/arm/atlas-arm/faq/index.html (Referenced on 08/16/2012).

[55] MPI Performance Topics. Online at https://computing.llnl.gov/tutorials/mpi_performance/ (Referenced on 08/17/2012).

[56] MPICH2 1.4.1 source code. Filename: *./src/mpid/ch3/channels/nemesis/include/mpidi_ch3_post.h*, *MPIDI_CH3_EAGER_MAX_MSG_SIZE*.

[57] Nvidia: Project Denver Is On-Track, Does Not Interfere with ARM's Own 64-Bit Tech - X-bit labs. Online at http://www.xbitlabs.com/news/cpu/display/20111110230832_Nvidia_Project_Denver_Is_On_Track_Does_Not_Interfere_with_ARM_s_Own_64_Bit_Tech.html (Referenced on 08/20/2012).

[58] F.Bloggs. 1993 *Latex Users do it in Environments* Int. Journal of Silly Findings. pp 23-29.