Constructing a low cost and low powered cluster with Parallella boards

Michael Kruger
Computer Science department
Hamilton Building
Prince Alfred Street
Grahamstown, South Africa
g12k5549@campus.ru.ac.za

ABSTRACT

Currently, high performance computing is largely restricted to well-funded research groups. This project aimed to create a high performance cluster using the cheap and energy-efficient 18-core Parallella boards. Four boards were connected over a network as a cluster and basic test programs were run using MPI. Experimental results show that the Epiphany chip performs very well compared with other energy-efficient chips such as the Cortex A9 ARM with a 11× speedup. Similar performance is achieved by the cluster of four Parallella boards against an Intel i5 3570 running a single thread. The Epiphany however, sees a drop in speed when attempting complex arithmetic operations compared with the other processors owing to the lack of hardware support. It is possible to achieve high performance using low-powered Parallella boards as long as the user is aware of the Epiphany chip's weaknesses and avoids these.

CCS Concepts

•**Computer systems organization** → **Multicore architectures; Heterogeneous (hybrid) systems;** *Interconnection architectures;* •**Networks** → *Network on chip;* •**Hardware** → Impact on the environment;

1. INTRODUCTION

As processor clock speeds get faster they reach a soft cap, from either excessive power consumption or heat output, which makes it infeasible to increase the speed any higher. To get around this problem chips have started to incorporate multiple processors that run slower yet achieve the same or better performance. Using multiple individual computers over a network, which is known as cluster computing, it is possible for them to work together as one system to solve problems [6]. This has made access to a "supercomputer" simpler for the ordinary user. Obviously, there is still the need for specialised super-computers, but this is largely restricted to well-funded research groups.

© 2015 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Cluster computing has become more and more popular as it is a low-cost way of creating high performance computers. Using ordinary personal computers (PCs) is one way of setting up a cluster, but the advent of even lower cost "computing" devices, has provided more opportunities than ever before of creating such clusters.

The Parallella is one such device that provides 18 cores per board at a comparatively low cost [2].

In this research, we investigate whether it is possible to build a high performance computer using a cluster of Parallella boards, thereby providing a very low-cost alternative to an ordinary PC.

2. RELATED WORK

The Iridis-pi[8] is a cluster constructed from 64 Raspberry Pi model B nodes and held together with Lego. The Iridis-Pi was created by a team at the University of Southampton. Each Raspberry Pi has a 700 MHz ARM processor, 256 MB of RAM, a 16GB SD card, and a 100Mb/s network interface. The benefits of the cluster are its cheap price, compact size, and low power consumption. Using the HPL benchmark and all 64 nodes the Iridis-pi achieves a throughput of 1.14 gigaFLOPS[8]. The price of a Raspberry Pi model B at time of writing is 450 South African Rand[1] bringing the cost of the 64 Iridis-pi nodes to 28800 South African Rand. This price excludes the cost of the switch, Lego and cableing.

3. HIGH PERFORMANCE COMPUTING

HPC is the term for very fast systems aimed at processing large volumes of information quickly. High performance computers are made up of multiple processors as the speed of a single processor has reached its limits due to physics [11]. HPC is most cost effectively obtained using cluster computing, as most places needing large amounts of processing power have multiple computers readily available [6]. HPC is used to run high cost simulations that would be too expensive or difficult to do physically; these require large amounts of computational power to be completed in a timely manner [21; 20], and therefore, more powerful machines are continuously being built. HPC has evolved over time with the number of cores in a single computer approaching the millions and performance reaching multiple petaFLOPS (10^{15} floating-point operations per second) [3].

Owing to current size and speed constraints on single core

[1]http://pifactory.dedicated.co.za/product/raspberry-pi-1-model-b/

processors, it has been found that running multiple slower cores is both more efficient and faster. Algorithms need to take into account ways in which to split the workload evenly between multiple processors if they want to obtain faster execution speeds using this type of architecture [11]. Many compilers have special flags so that they can optimise programs for parallel computation [17]; this, however, only achieves a minor boost in speed when compared with an efficient algorithm that splits the work into multiple pieces that can be distributed among multiple processors. According to David Geer [11], to take advantage of multiple cores, programs need to be rewritten so that they can run on multiple threads, with each thread assigned to a separate processor.

Some common HPC concepts are detailed below.

Throughput The rate at which data can be successfully transferred over a channel.

Shared Memory Memory, over which multiple processes have control and which is shared between them.

Distributed memory This is the term used in a multicore system when a processor has its own private memory that it can access and use; however, when it needs information from another process, it has to communicate with the other process and request the particular data.

Bottleneck A bottleneck occurs when the effectiveness of a system is restricted by a single or small number of resources.

Latency This refers to the amount of time required for an instruction to travel from its source to its location and be acted upon. A large amount of latency is detrimental as the time to pass information around a system will become a bottleneck and the system will not be able to make use of all its computational power.

FLOPS Floating Point Operations Per Second is the usual measurement of a high performance computer. It refers to the number of instructions using floats that a system can compute per second. It is usually referred to using a prefix such as giga for 10^9 or peta for 10^{15}.

4. CLUSTERS HARDWARE AND SOFTWARE

With the demand for large amounts of processing power, various ways of creating supercomputers cheaply have appeared. Clusters of computers connected on a network can be purposed to work together as a supercomputer. With the increased speed and decreased latency of the Internet, it is possible to create a cluster using computers from all over the world; this has led to programs and applications that allow a computer to connect to a pool of other computers and add its processing power to the computation. There are, however, some factors limiting the effectiveness of cluster computing. These include building a switch to keep up with the speed of a single core processor and creating compilers that make good use of multiple processors.

4.1 Architectures

There are two generally used methods for controlling communication within a cluster:

MPI The individual nodes of the cluster can communicate with each other using a message passing interface (MPI), which provides a thread safe application programming interface (API) that allows the work to be effectively delegated to multiple nodes on the network [19; 14] and information passed between each node so that it can be worked on. More information on MPI is provided in Section 4.2 with the overview of MPICH.

Parallel Virtual Machine uses a parallel virtual machine (PVM) approach, which combines all the nodes and allows them to appear as a single PVM. This PVM handles all the message passing, task scheduling, and data conversions. To set this up, each node of the cluster needs the same PVM image installed and must be marked as a PVM node. Parallel virtual machines are popular due to the ease with which the cluster can be managed[12]. Some of the available PVMs are reviewed in Section 4.2.

4.2 Software

Some of the software useful for being deployed on clusters is reviewed below:

MPICH MPICH is a high performance, portable and widely used implementation of the Message Passing Interface (MPI) standard. MPICH was created for distributed memory systems, with the idea of portability and high performance in mind. Excellent results have been achieved with MPICH, which is the most used implementation of MPI in the world, and its derivatives. MPICH is able to work in many different environments and take advantage of what is available to increase performance while maintaining portability, for example, using shared memory to pass messages between processors faster. MPICH is distributed as source code and has an open-source freely available licence [5; 14; 13].

Open-MPI Open-MPI is an open-source implementation of the Message Passing Interface [1]. It has multiple partners from across the HPC community, maintaining its libraries[10]. These partners include ARM, which provided the Zynq-chip for use on the Parallella board [1]. Open-MPI conforms fully to MPI-3.1 standards, supports multiple operating systems[10], and is provided by default on the Parallella board Ubuntu distribution.

OpenMP OpenMP is "an industry standard API for shared-memory programming" [9]. A shared-memory parallel system describes a multi-processor system where individual processors share one memory location [7]. Each processor can still have its own personal cache memory to work with as the speed difference between main memory and processor memory would cripple the speed if the processor needed to pick up everything from the shared memory space. OpenMP was introduced to fix the inability of compilers to make good decisions on how to split up a program to take advantage of multiple processors; although this is possible for simpler programs, a user would need to cast a more discerning eye for more complex problems [7]. OpenMP provides an alternative to message passing

in parallel programming. OpenMP is a set of routines and compiler directives to manage shared-memory parallelism. The OpenMP standard is made up of four parts, namely, control structure, data environment, synchronisation, and runtime library [9], which can be added to a sequential program written in C, C++ or Fortran [7].

5. PARALLELLA CLUSTER

5.1 Parallella Specifications

The Parallella board is an "affordable, energy efficient, high performance, credit card sized computer"[2] that aims to provide a platform for developing and implementing high performance parallel processing. The 66-core version (64-Epiphany cores and two ARM cores) of the Parallella board achieves over 90 gigaFLOPS (10^9 floating point operations per second), while the 18-core (16-Epiphany and 2 ARM cores) version can reach 32 gigaFLOPS using only about 5 Watts. The Parallella has a 1-Gbps Ethernet port allowing a large amount of information to be passed quickly over the network. This increases its ability to work in a cluster as it can pass information to its peers rapidly, provided that the switch is capable of handling the 1Gbps bandwidth.

The aim of creating the Parallella board was to make parallel computing more accessible by creating an affordable, open-source, and open-access platform.

The price of a Parallella board starts at $99 (at the time of writing) for the 16-core board and uses a customised ARM implementation of Linux (Ubuntu 14.04). The Parallella is three years old and software that takes advantage of this is still being developed[2][18].

Programming for the Epiphany chip (the Parallella boards co-processor) is done in C and the Parallella team have provided some basic primitives with the SDK (Software Development Kit). Memory addressing, barriers, and communication between eCores are a few examples of what is provided by the SDK.

To run programs on the Epiphany chip, a workgroup of cores needs to be set up. This can be done using the provided SDK to give a starting node and the number of columns and rows in the matrix of cores [4; 16; 22; 19].

5.2 Physical Layout of Cluster

Figure 1 shows an overview of how each entity is connected to the cluster. For further information see [15].

The four parallella boards are connected to a gigabit switch with each having the hostname "parallella" followed by a unique number, for example, "parallella1". Parallella1 was chosen to launch programs to the cluster and can be referred to as the head node. The parallella boards are stacked on top of each other using spacers to separate them and two fans on either side. The fans are arranged to force air through the stack, with one pushing and the other pulling air through to cool them. This setup increases airflow allowing for heat to be carried away faster. The stack of parallella boards are placed on a non conductive surface and power is transfered through the spacers connecting each board. Access to the head node is done via ssh from a non Parallella host which will be referred to as the Controller, which is also connected to the gigabit switch. Giving commands to the cluster is typically done by connecting to the head node of the parallella stack via the Controller. Connected to the switch other than the cluster and external machine are a NFS server and a DHCP server. These two entities do not need to be on separate machines but in this case they are, They may also be setup to be on the Controller but when connecting a new computer to control and interact with the cluster it will have to be setup to fulfil the NFS and DHCP roles making the cluster less portable from machine to machine.

[2]https://www.kickstarter.com/projects/adapteva/parallella-a-supercomputer-for-everyone

5.3 Software

In this section, we give a break down of the software and required configuration needed to allow programs to be executed using MPI on the Parallella cluster.

5.3.1 Linux

The Parallella boards were supplied with a raw disk image of Ubuntu 14.04, downloaded from the Parallella website[3]. There are three different versions of the Parallella board for which Ubuntu images are provided: the z7010 image applies to the P1600 and P1501, while the z7020 image is for the P1602 [18]. The z7010 is specific to Parallella boards with the Xilinx Zynq Dual-core ARM A9 XC7Z010 host processor, which matches those used in the cluster. The other image is for the Xilinx Zynq Dual-core ARM A9 XC7Z020, which is used on the embedded version of the board.

There are also two different images per host processor: the first is a headless one, which is what we have used as it is the most stripped down of the images, while the other includes a desktop environment and HDMI drivers. The latter version is useful for displaying generated graphics and writing code in a GUI environment. The latest version of Ubuntu 14.04 headless for the z7010 was downloaded[4] and extracted.

Each Parallella board has a microSD card that acts as its main hard drive and into which the board is booted. All software that will ultimately be executed on a Parallella board must be copied to the microSD card. The capacity of the cards used in the cluster is 16 GB and the speed class is UHS mark 1 which makes the Minimum Serial Data Writing Speed 10 MB/s[5]. Using a Windows 7 machine, Win32 Disk Imager[6] and an SD to USB adaptor, the downloaded image was copied to each of the microSD cards.

5.3.2 SSH

SSH or Secure Shell is a network protocol that allows one computer to securely access another over a network. In the case of the Parallella stack the Open-SSH implementation will be used to facilitate the passing of messages between nodes by OpenMPI. Secure shell by standard uses port 22 and TCP to create connections between nodes so it is vital that this port is available and open.

For convenience, passwordless SSH was set up between the nodes so that OpenMPI is able to run programs on the nodes without needing a password to be provided for each run. To do this RSA public/private key pairs are used to

[3]ftp://ftp.parallella.org/ubuntu/dists/trusty/image/

[4]At the time of writing, the appropriate file was Ubuntu-14.04-headless-z7010-20150130.1.img.gz

[5]https://www.sdcard.org/developers/overview/speed_class/

[6]Downloaded from http://sourceforge.net/projects/win32diskimager/

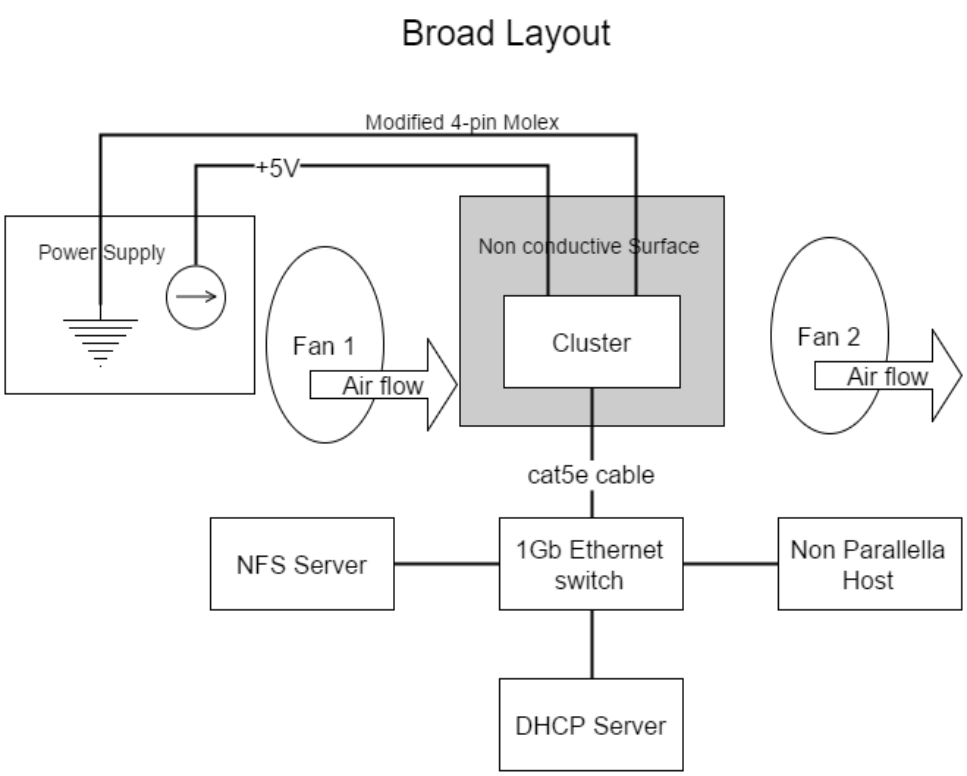

Figure 1: Basic overview of the cluster

authenticate the master with the slave nodes.

5.3.3 *Open-MPI*

OpenMPI was preinstalled and the environment correctly set up on the Ubuntu image provided. To set up OpenMPI, a user account was created on each parallella with the same directories and file locations for consistency when running anything through mpiexe or mpirun. To synchronise the files so that each instance of a program on each node had access to its local directory, NFS was used, the configuration of which is described in Section 5.3.4.

If SSH has been configured correctly, the password of each node being used does not need to be entered when running an MPI program.

5.3.4 *Network File Share*

For OpenMPI to work correctly, the file structure and locations of used files must be the same on each node. For this to happen, one of the nodes or a different machine was chosen to host all the program files and have all the nodes connect to that shared drive so that each node has the exact same version of the files as every other node. For our cluster, a different machine was used as problems occurred when running nfs-kernel-server on the master node, which was originally set as one of the Parallella boards; for details, refer to Section **??**. To remedy this, a Raspberry Pi was set up with an installation of nfs-kernel-server and its dependences through a repository. The upside of using a Raspberry Pi is it means that the cluster continues to use very little power, the downside is that the Ethernet adaptor used by the Raspberry Pi is only 100Mb/s and will not take advantage of the 1Gb/s Ethernet that is used by everything else that is part of the cluster. So the shared storage is accessed slower and if the cluster tries to access and copy large amounts of data there will be a large performance drop. To counter this when using large files and data they should be copied to each Parallella board's local hard drive for faster access.

6. BENCHMARKING THE CLUSTER

As the Parallella stack is a heterogeneous cluster, to be able to utilise fully the computational power of each board, MPI is used in conjunction with the Epiphany libraries.

Creating a program for both MPI and the Epiphany co-processor requires splitting of the work on two occasions: once at the MPI level and again for distribution to the co-processor. Figure 2 illustrates the process of running a program that uses MPI to spilt work between the four Parallella boards in the cluster, which then load the required srec files onto their respective co-processors. Figure 2 displays the way that MPI is run over SSH using parallella1 as the master node; this may slow down parallella1 owing to the overhead of setting up the MPI program on each Parallella board. If the cluster has a large number of nodes, the delay to parallella1 may be large, causing an unbalanced workload and extended execution time.

Using MPI two benchmarks were executed. The first benchmark determines the Parallella stack's ability to do a large number of floating point multiplications while the second tests the cluster's communication speed.

6.1 Floating Point Multiplication

This benchmark, which is embarrassingly parallel, tries to

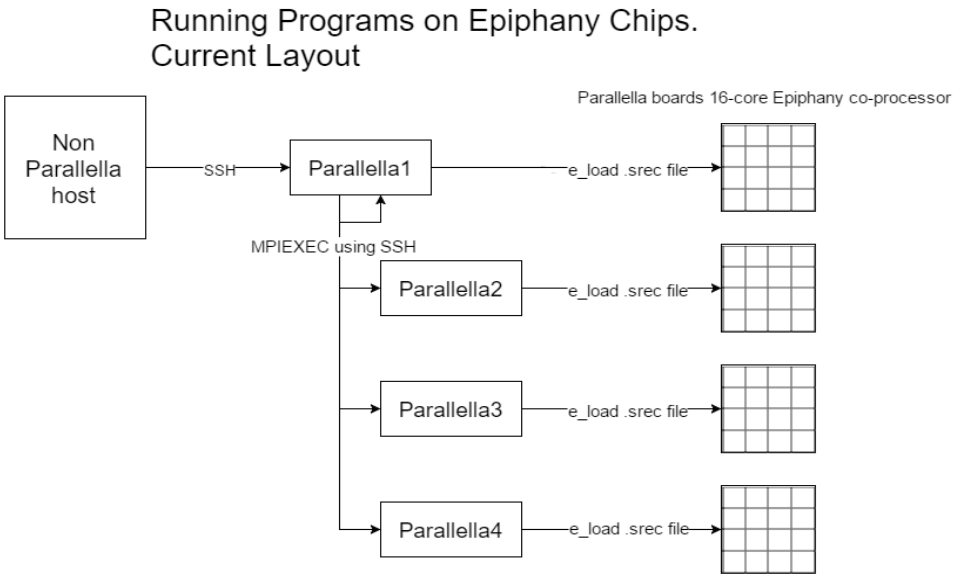

Figure 2: Starting a program using MPI for execution on Epiphany cores

emulate ideal conditions for the Parallella stack by creating and splitting up work for each core to do. This work requires no synchronisation with any other processes. This benchmark merely multiplies two floating point numbers for the desired number of iterations. However, the two floats are modified on every iteration to prevent the processor from possibly using a cached answer. This benchmark has a sequential version, which was executed on the i5 CPU and on a single ARM processor, a Parallella version for running on a single Parallella, and a modified version of the Parallella implementation with added MPI compatibility for execution on the cluster.

Table 1 shows that the time difference between one Parallella board and four Parallella boards is closer to a 2.5× speedup instead of the expected 4× increase. Before adding the MPI_Barriers and having each process started by MPI being completely separate after launch, the speedup was between 3.9× and 4.1×. This situation we felt was unrealistic as even the most parallel application would need to amalgamate its results at the end of the computation. This adds a constant time to the total computation, which, if the computation time were sufficiently large, would not effect the time in a significant way as it is not reliant apon the amount of iterations.

The difference in the timings of the Intel i5 3570 and the Parallella are very close and while the i5 is faster, it is worth noting that at 100% load, the Parallella stack consumes approximately 20 Watts of power, which is three times less than that used by the i5 when idle[7].

6.2 Ethernet Bandwidth

This benchmark was created by Blaise Barney[8] to measure point-to-point communications between MPI processors.

The benchmark pairs processes together and sends incrementally larger and larger messages between them and notes the speeds. As seen in Table 2, the bandwidth appears to get faster the larger the messages get, excluding some outliers. The larger message size increases the time in which a message is in transmission while the time to set up the connection between nodes stays the same as the number of messages sent is constant. The larger message size causes the MPI overhead per byte of data to be less.

7. LIMITATIONS

This section will cover the limitations and weaknesses of the Parallella board that were discovered over the course of this research.

7.1 Complex Arithmetic

When running the benchmark in Section 6.1 instead of multiplying we attempted to divide the two floating numbers. The performance difference in this case was huge, with the ARM 667MHz core 16.74× faster than the Parallella stack just using the Epiphany co-processor and 66.17× faster than a single 16-core Epiphany chip. This is due to the eCores having no hardware support for higher-complexity arithmetic. This translates to doing long division and requiring more CPU cycles per instruction. The slow division has far reaching consequences as many useful functions rely upon it such as modula and rand(). To try mitigate the performance drop if a large amount of division or complex arithmetic needs to be done it is better to pass the work to the ARM processor which can do the calculations and pass the data back to the co-processor.

7.2 Hardware Optimisations

All optimisations for Epiphany programs must be done by the compiler instead of clever hardware. The simple RISC cores provided on the Epiphany chip do not attempt to predict or change the ordering of instructions being executed[9].

[7]http://www.tomshardware.com/reviews/core-i5-3570-low-power,3204-13.html

[8]available from https://computing.llnl.gov/tutorials/mpi/samples/C/mpi_bandwidth.c

[9]http://www.bdti.com/InsideDSP/2012/09/05/Adapteva

Table 1: Time and speedup per machine for 100,000,000 iterations of floating point arithmetic

Processor type	Time(s)	Speedup with respect to one ARM core
One ARM core	16.3	1.0
One Epiphany chip	1.38	11.8
Four Epiphany chips	0.54	30.0
Four Parallella boards using ARM and Epiphany	0.34	48.1
Intel i5 3570	0.30	54.2

Table 2: Average point to point bandwidth using MPI

Message size(bytes)	Average bandwidth (MB/sec)	Average bandwidth with two processors per Ethernet adapter (MB/sec)
100000	44.48	37.52
200000	47.97	30.04
300000	45.71	27.06
400000	46.22	41.22
500000	47.2	45.81
600000	48.4	45.36
700000	49.76	46.93
800000	49.31	47.67
900000	49.31	47.49
1000000	51.15	48.25

8. CONCLUSION

Our objectives for this project were to build a cluster using four Parallella boards and to benchmark the cluster against similar low-cost systems.

In this paper, we discussed how a cluster of four Parallella boards was constructed and powered using a single power supply. The Parallella stack is actively cooled by two fans and connected to a 1Gb/s switch to facilitate communication between boards.

Software for programming and running distributed programs on the cluster was set up and configured and an NFS server for storing files and a DHCP server with nat to provide Internet and IP addresses were set up and added to the network. This means that the only requirement for interaction with the Parallella stack is a working SSH-client, making the system easily accessible with minimum effort and easy to set up on multiple machines. This satisfies the first two main objectives.

For benchmarking, some simple programs were created and executed sequentially and in parallel on the Parallella stack. The Parallella stack performed slightly worse when compared with an Intel i5 3.40GHz processor, but used at least three times less power[10]. In terms of cost, both the four Parallella boards and the Intel i5-3570 cost approximately $400[11], but to make use of both devices more components are needed adding to the total cost. Depending what hardware is bought to use with these devices the costs could be similar.

We found that the lack of hardware support for complex arithmetic such as square roots and division can be costly to performance if the instruction is frequent enough and steps are not taken to delegate complex arithmetic to the local ARM core, which does have the hardware support to process quickly. It is was also discovered and noted that the RISC cores on the Epiphany do not do any runtime optimisations and the only automated optimisations are provided by the compiler.

As this research only set out to create a basic cluster, there is plenty of scope for further development and optimisation; for example more benchmarks and programs should be executed to further test the Parallella stack. Additionally, research into using the Brown Deer COPRTHR (co-processing threads) library[12] to help create homogeneous Epiphany MPI programs is needed. The Parallella's Zynq processor has a field-programmable gate array that may be used to re-program the boards to better suit different purposes.

To facilitate use of the cluster, a web front end could be created to receive and run programs or to monitor the cluster's current work load. This could be extended to process requests and schedule work so that the cluster can be used simultaneously by multiple people.

Other extensions could be creating a library for languages other than C, C++ and OpenCL to be able to use the Parallella stack.

[10]http://www.tomshardware.com/reviews/core-i5-3570-low-power,3204-13.html

[11]http://www.amazon.com/Intel-i5-3570-Quad-Core-Processor-Cache/dp/B0083U94D8/

Acknowledgments

I would like to acknowledge the financial and technical support of Telkom SA, Tellabs, Easttel, Bright Ideas 39, THRIP and NRF SA (UID 75107) through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

References

[1] Open MPI: Open Source High Performance Computing. Online. Accessed: 2015.11.17. Available from: http://www.open-mpi.org/.

[12]http://www.browndeertechnology.com/coprthr.htm

[2] The Parallella Board. Online. Accessed: 2015-03-01. Available from: https://www.parallella.org/.

[3] Top 500 super computers. Online. Accessed: 2015.02.25. Available from: http://www.top500.org/.

[4] Adapteva. Epiphany Datasheet. Online. Accessed: 2015.05.05. Available from: http://adapteva.com/docs/e16g301_datasheet.pdf.

[5] Bridges, P., Doss, N., Gropp, W., Karrels, E., Lusk, E., and Skjellum, A. User's guide to MPICH, a portable implementation of MPI. *Argonne National Laboratory 9700* (1995), 60439–4801.

[6] Buyya, R. High performance cluster computing. *New Jersey: Prentice Hall* (1999).

[7] Chapman, B., Jost, G., and Van Der Pas, R. *Using OpenMP: portable shared memory parallel programming*, vol. 10. MIT press, 2008.

[8] Cox, S., Cox, J., Boardman, R., Johnston, S., Scott, M., and O'Brien, N. Iridis-pi: a low-cost, compact demonstration cluster. *Cluster Computing 17*, 2 (2014), 349–358.

[9] Dagum, L., and Menon, R. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE 5*, 1 (1998), 46–55.

[10] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2004, pp. 97–104.

[11] Geer, D. Chip makers turn to multi-core processors. *Computer 38*, 5 (2005), 11–13.

[12] Geist, A. *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT press, 1994.

[13] Gropp, W., and Lusk, E. Installation guide for mpich, a portable implementation of MPI. Tech. rep., Technical Report ANL-96/5, Argonne National Laboratory, 1996.

[14] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel computing 22*, 6 (1996), 789–828.

[15] Kruger, M. J. Building a Parallella board cluster, 2015. Honours thesis at Rhodes University.

[16] Olofsson, A., Nordström, T., and Ul-Abdin, Z. Kickstarting high-performance energy-efficient manycore architectures with Epiphany. *arXiv preprint arXiv:1412.5538* (2014).

[17] Padua, D. A., and Wolfe, M. J. Advanced compiler optimizations for supercomputers. *Communications of the ACM 29*, 12 (1986), 1184–1201.

[18] Parallella. Parallella Reference Manual. Online. Accessed: 2015.05.05. Available from: http://www.parallella.org/docs/parallella_manual.pdf.

[19] Richie, D., Ross, J., Park, S., and Shires, D. Threaded MPI programming model for the Epiphany RISC array processor. *Journal of Computational Science* (2015).

[20] Sanbonmatsu, K., and Tung, C.-S. High performance computing in biology: multimillion atom simulations of nanoscale systems. *Journal of structural biology 157*, 3 (2007), 470–480.

[21] Tezduyar, T., Aliabadi, S., Behr, M., Johnson, A., Kalro, V., and Litke, M. Flow simulation and high performance computing. *Computational Mechanics 18*, 6 (1996), 397–412. test.

[22] Varghese, A., Edwards, B., Mitra, G., and Rendell, A. P. Programming the Adapteva Epiphany 64-core network-on-chip coprocessor. In *Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International* (2014), IEEE, pp. 984–992.