# DISTRIBUTED COMPUTING WITH THE RASPBERRY PI

by

BRIAN DYE

B.A., Kansas State University, 2012

---

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2014

Approved by:

Major Professor
Mitchell Neilsen

# Abstract

The Raspberry Pi is a versatile computer for its size and cost. The research done in this project will explore how well the Raspberry Pi performs in a clustered environment. Using the Pi as the components of a Beowulf cluster will produce an inexpensive and small cluster. The research includes constructing the cluster as well as running a computationally intensive program called OpenFOAM. The Pi cluster's performance will be measured using the High Performance Linpack benchmark. The Raspberry Pi is already used for basic computer science education and in a cluster can also be used to promote more advanced concepts such as parallel programming and high performance computing. The inexpensive cost of the cluster combined with its compact sizing would make a viable alternative for educational facilities that don't own, or can't spare, their own production clusters for educational use. This also could see use with researchers running computationally intensive programs locally on a personal cluster. The cluster produced was an eight node Pi cluster that generates up to 2.365 GFLOPS.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This project will be investigating the Raspberry Pi computer and its ability to perform within a computer cluster environment. Computing has found uses in many research areas. Researchers utilize computers to run simulations of natural phenomena or analyse data pertaining to their work. As these programs grow in complexity and accuracy so does the computer resources required to process them. These programs can quickly outscale the available resources of a single computer and can create processing times of days to weeks. To develop systems that can run these programs effectively researchers create computing systems consisting of multiple processors and machines working together. These systems are called distributed systems. Given a task these processors within the system will each take a part of the problem and process their sections simultaneously.

The field involved with designing and improving these distributed computing systems is called High Performance Computing (HPC). Even with the computing power of the supercomputers produced by aggregating multiple processors and machines, the ambitions and program complexity will still be increasing throughout the scientific and business communities. A currently growing example of this trend is Big Data. Big Data is the name given to incredibly large amounts of stored information on databases and clouds[2]. Market

researchers utilize big data to discover trends or patterns to help guide their business decisions[3]. While other researchers utilize big data strategies such as analyzing genetics of thousands of humans to discover gene patterns that lead into generating hereditary disease development[4]. In 2012 calculations showed that 2.5 exabytes of data (or 2.5 billion gigabytes) were being generated each day[3]. As the demand for big data storage and analytics grows, the databases, supercomputers, and algorithms that are the backbone of this industry must also grow with it. A consequence of this is that there will also be a demand for the education of the engineers who can build and maintain these systems. In addition a requirement for scientists to research and implement improved techniques for the evolving field of supercomputing.

For the purpose of education and independent research, the cost of producing and maintaining a clustered computer can be prohibitive. The upfront cost of obtaining components and the space required to store the devices are just the initial concerns. Depending on the scale of the cluster to be built, ongoing concerns include the cost of electrical power for running and continuously cooling the computer cluster. Another scenario found in practice is a facility already owns a computer cluster but has issues in the way for using it in classroom usage. An example of this is a research first, pedagogy second policy[5] where the cluster prioritizes work related tasks higher than education related tasks. Leading to scenarios where lab or project based classes are not feasible to execute with the cluster potentially being unavailable for them during higher priority workloads.

Scenarios like these can be potentially resolved by the Raspberry Pi within a cluster. The Raspberry Pi is an inexpensive computer that rivals the size of a credit card[6]. Linking a series of Raspberry Pis into a distributed computing system would just as well create a very small sized cluster. Those performing research into the field of distributed computing could potentially use these small scale clusters as personal research tools. Just as college level classrooms could utilize these for hands on cluster education in place of an absent

or unavailable computer cluster. The experiments to be performed on the Raspberry Pi and the produced cluster will help investigate the performance of a Raspberry Pi computer cluster.

This paper below will continue with a more in depth description of computer clusters, the Raspberry Pi, and information on some of the programs Ill be using in this cluster. The next chapter will explain the experiments performed, followed by how the cluster was constructed. The paper will end with an analysis of the experimental results.

## 1.1 Distributed Computing

Mentioned previously, distributed computing aggregates computing power and resources from many computer sources to complete a common objective. A distributed system is called a heterogeneous system if there are various types of nodes with varying hardware within the system. Otherwise a system containing only a series of identical nodes is named a homogenous system. How the system is constructed depends on the workloads designers estimate the system will be working with. A system that will be serving millions of http requests will be largely focusing on their bandwidth, high availability, and databases, while a big data analysis will appreciate large amounts of ram and small latency between the data storage and the processors. Other systems include variations on how the machines are connected. Such as linking together workspace computers in a building, creating a computer cluster out of the borrowed power of the workers stations[7].

While there are many types of distributed computing systems, this thesis will be focusing on the computer cluster, more specifically a beowulf cluster. A computer cluster is formed by connecting the nodes over a Local Area Network (LAN) within the same room or building. Clusters that fall under the Beowulf cluster category have a few notable restrictions that define them[7]. Beowulf nodes are created from easily affordable, off the shelf parts. These

clusters contain a homogeneous set of nodes that all contain their own local storage, ram, processor, operating system, and network connection. Their nodes are dedicated to the beowulf cluster and arent simply a persons personal workstation that the cluster borrows spare computing power the user happens to not be using. One node is special, labeled as the master which will be in charge of initiating and distributing programs towards the slave nodes. This master node will also act as the access point to the cluster and the node that will accept commands from users. Other specifically named types of clusters will share one or more of these requirements, but this is what in general it means to be a beowulf cluster specifically. The restrictions for beowulf clusters create a cluster that is simple, aggregates power from cheaper components, and is consistent. Since each node and the network is dedicated to the cluster the cluster won't vary in computing power as in the workstation distributed computer scenario would.

In the aim of making a general efficient cluster, a balance between cpu speed, I/O speed, and available ram is preferred. I/O speed refers to the speed of the network or the latency between reading and writing data between the hard drive and available memory. The amount of ram available limits the size of problems the computer can work on efficiently. An example of this is using more memory than available ram leads to thrashing. Trashing is when the computer is forced to temporarily store parts of memory to the hard drive to make space for what the program needs to load into memory now. The worst case scenario comes when the computer must juggle memory between ram and the hard drive repeatedly. While this occurs the processor cant make much progress. The program cant continue until the needed data loads. Network speed should also be sufficiently fast to minimize the time it takes for nodes to communicate. Effectively the time a processor must wait for a message to be fully sent is time not spent contributing processing power to the completion of the program. Both of these scenarios where the processor is idle because of thrashing or communication bottlenecks is called CPU starvation, as the system just isnt feeding data fast enough for

continuous processing. These concerns are shared generally with all of computing however in a clustered environment one processor lagging behind can be the cause of slowdown for hundreds of others. And in this field of generating massive computing power, these are very important concerns.

### 1.1.1 How Parallelization Works

As Ive mentioned multiple times, clusters will be aggregating computing power. At this point Im going to explain how this is achieved, as it is only possible with certain types of programs. For a cluster to achieve a processing power of completely independent but connected machines the program must be divisible. This means that at some point in a program the remaining work can be divided into several subprograms and computed relatively independently of each other. By sending each division to a node within the cluster the processing can be done in parallel. This means that sequential programs will not be gaining any speedup from clustering, these programs only can be ran on one machine. Programs that are designed to be divisible are called parallel or distributable programs.

An example of a simple parallelizable algorithm is matrix addition. Two large matrices added together can observe a speedup in a parallel environment by dividing the problem into subproblems. The first step is to distribute the matrices to the cluster, where each node in this case can claim 1/Nth of the rows of matrix A and the same rows from matrix B. Where N is the number of nodes available for computing the answer. Each node simultaneously adds their two submatrices together and eventually returns the result back to the master node who compiles the resulting matrix from the clusters subproblem results. Matrix addition is a very simple parallel algorithm because each subprogram doesnt depend on any other subprogram to complete.

For this very simple example the resulting matrix would be calculated in a little more than 1/Nth the time taken for a single node to compute the same answer. The overhead of

distributing the program to the cluster and the time taken communicating the results back to the master will add time to the clusters results. However the parallel computation most cases, such as this one, will show a dramatic speedup to the computation time.

Another potential case in this scenario is if the three matrices are so large that they exceed the available ram of the node working alone. In this case the calculation can potentially take longer as page faults and thrashing are possible. Though with distribution, if 1/Nth of each array can fit into the memory of a single node then this problem will not experience the same potential thrashing scenario. This is one of the great advantages of a clustered environment that we can take problems that are too large for one machine and divide them into chunks that can fit comfortably on a single machine.

## 1.2 Brief Overview of the Raspberry Pi

The Raspberry Pi is a small computer produced by the Raspberry Pi foundation. The foundation aimed to make a small, low cost computers to encourage the practice and learning of basic computer science concepts[6]. Their goal was to create a device with just enough processing power while still being inexpensive, small, and available for a target audience of hobbyists, education, and inventors. To aid attracting these customers the Raspberry Pi was to have enough processing power for simple tasks such as web browsing, playing simple games, and capable of outputting 1080p from an hdmi output. The development process involved six years of designs and planning until in 2012 the first model B Raspberry Pi became available for purchase. The interest the community had for these tiny computers exceeded the expectations of the Raspberry Pi Foundation as they sold out within minutes[8]. The order backlog for the Pis took over 3 months to finally be overcome.

The current iteration of Raspberry Pis consist of a model A and model B. Both models contain many of the same components including a USB connector, HDMI slot, and a 3.5mm

audio jack. Each version also requires an SD card that the Pi will use to boot with and use for its local persistent storage. The processor is a 700 MHz arm6 chip[9]. The arm architecture has a history of energy efficiency, low heat production, and small size has made a popular choice for mobile devices such as cellphones and tablets. For these same reasons its also a good choice for the Raspberry Pi. The gpu found within the Pi is a 250 Mhz VideoCore IV.

The Raspberry Pi runs on a linux operating system, the most popular operating system of choice is Raspbian. Raspbian is debian operating system optimized for the Pi hardware. This operating system is produced and updated by the Raspberry Pi foundation and is freely available online or found preloaded on special Raspberry Pi SD cards. Since their initial release the Raspbian image has been updated with some important features. The initial release focused on python and c/c++ and in 2013 Java 7 JKD was added to the Raspbian repository[10].

An announcement that excited many hobbyists was the Raspberry Pi Foundations announcement of overclocking profiles being added to the Raspbian image[?] . Since these profiles were tested and supported by the foundation, they also have the added bonus of not voiding the Raspberry Pi warranty. The highest overclock profile, called Turbo Mode, scales the Raspberry Pis processor speed to 1 GHz and the gpu speed becomes 500 MHz. The Raspberry Pi Foundation also states that in their testing didnt detect any detriment to the lifespan of the Pi when overclocked to these profiles. Should the Pi begin to overheat however the overclock will automatically revert to normal speeds (700 Mhz, 250 Mhz) until the temperature no longer exceeds 85 degrees celsius. These are the main features that the model A and B have in common, the table below shows the hardware specifications of the two models and the few areas they differ.

Because the intention is to turn multiple Raspberry Pis into a computer cluster the ethernet port located only on the model B is critical. The model B is what will be used to

| | Model A | Model B | With Turbo |
|---|---|---|---|
| CPU | 700 MHz | 700 MHz | 1000MHz |
| GPU | 250 MHz | 250 MHz | 500 MHz |
| RAM | 256 MB | 512 MB | |
| Ethernet | None | 10/100 Mbit/s | |
| USB 2.0 | 1 Port | 2 Ports | |

**Table 1.1**: *Raspberry Pi Hardware Specs*

construct the cluster and for the rest of this thesis any mentioning of the Raspberry Pi will be intended toward the model B.

## 1.3 Programs

## 1.4 OpenFOAM

OpenFOAM is an open source collection of Computational Fluid Dynamics (CFD) solvers[11]. With it researchers can simulate and view how fluid or gaseous systems react to a given situation under a specific scientific model[12]. Since the solvers of OpenFOAM are open source researchers are able to modify the solvers at will. This allows the simulations to change to the needs of the researchers implementing it. The default package of OpenFOAM contains over 80 solver programs, each solver simulating a specific model in CFD[11].

OpenFOAM also has the ability to be ran in a parallel fashion. For a CFD solver parallelism quickly becomes more vital when increasing the resolution or length of the simulation. The runtime to solve such problems can lead to hours or days of computation time to finish the simulation on a single machine. Because of this importance, the parallelism is built into the OpenFOAM libraries and simulations created in OpenFOAM are just as easily ran in parallel across multiple machines as they are ran on a single core.

## 1.5 Benchmarking With High Performance Linpack

The act of benchmarking in computing is attempting to measure the peak performance of computer systems or its components. There are many methods of benchmarking and in general each benchmark focuses on a single aspect of the machine to be tested. Some examples of these aspects are FLoating Operations Per Second (FLOPS), memory bandwidth, network throughput, and network latency.

However the numbers coming from a benchmark dont entirely state how well a computer system runs any program. In the most pragmatic sense a benchmark simply tells you how the computer system performs at processing the benchmarks test. For example a benchmark focusing on network throughput will usually be optimized for running on that system in particular. The optimized algorithms then produce a much more attractive throughput result that general programs will find difficult to reach. However in a more realistic point of view one can view these results as the upper limit of the systems performance.

The value that comes from benchmarking is it allows system designers to create a baseline result for future comparisons. This base result allows cluster inventors and managers to modify their system configuration, either with new libraries, hardware, or network topologies and compare their new configuration with the old. Using benchmarks in this way will allow the cluster administrators to measure any performance increase or decrease within their newly modified architecture.

For this project I will be focusing on measuring the Raspberry Pi clusters ability to generate FLOPS with the High Performance Linpack (HPL)[13]. As many scientific simulations involve floating point operations how fast a computing system can generate these operations are a real concern for these areas. HPL calculates the rate these operations are completed by solving a dense, randomized, series of linear equations over a large coefficient matrix.

### 1.5.1 The Current Standard

The high performance linpack is currently considered a standard for measuring computing power in the HPC community, and has been for a couple of decades. But this may not always be the case. Bill Kramer of the National Center for Supercomputing Applications (NCSA) has summarized a few key drawbacks of measuring modern supercomputers with HPL[14]. One of biggest points to consider is that modern computing has changed over the last couple of decades. During the time period HPL was created the most common computing problem was solving large dense linear equations arrays, so it makes sense that HPL would measure these situations specifically. However modern day supercomputing programs have grown to also test the limits of the entire computer cluster which HPL doesnt factor into the equation. This leads to the problem that the general supercomputing standard has fallen to measuring only a single facet of the entire system and in effect penalizes a more well rounded supercomputer. Some of Kramers potential improvements to the benchmarking standard include simply combining multiple benchmarks that focus on other areas of the systems as well as system costs being a factor for being the best supercomputer[14]. Even though the current standard, HPL, is becoming somewhat dated I still believe that for this project it will still be useful. The amount of FLOPS generated by our Pi cluster will be relevant for computing programs such as the simulations generated by OpenFOAM. Measuring the FLOPS will also show how well the Pi cluster is performing in a cluster.

## 1.6 Related Work

### 1.6.1 LittleFe

The LittleFe project began in 2005 to assist institutions in high performance computing education[15]. LittleFe acknowledged the fact that not all institutions can afford systems such

as computer clusters for educational purposes such as hands-on labs work or demonstrations. For the goal of bringing clusters to fill this educational gap, the project developed critical design constraints such as costing less than $3000, weighing under 50 pounds, and fits in a travel case (in this instance a pelican case) that can be easily transported[16]. The current iteration of LittleFe is the Littlefe v4 which is a six node beowulf style cluster. Each node is powered by a 1.8Ghz dual core atom processor with 2GB of ram communicating over a gigabit switch. The total parts list of the LittleFe cluster with todays prices easily falls under $2,500, includes the casing the nodes will be mounted in, and was designed to be put together with simple tools[15]. As a strategy to reduce weight and cost the master node is the only node to contain a hard drive. The cluster itself netboots off of a linux image in the dvd drive making setting up and managing the nodes simple. The LittleFe group has also had an active part in high performance computing outreach by taking their LittleFe clusters to competitions, conferences, and LittleFe build events[16]. LittleFe is a solution for institutions lacking demonstration clusters and in a way the Raspberry Pi cluster can become an alternative solution as well.

### 1.6.2 Raspberry Pi in Bitcoin Mining

One group showing quite a bit of interest in Raspberry Pis is the Bitcoin mining enthusiasts/industry. Bitcoin is a digital currency, sometimes called cryptocurrency due to how it's processed[17]. With no central bank the transfer and holdings of bitcoins relies on a decentralized system to process the digital transactions between bitcoin wallets. This decentralized processing is completed by the bitcoin miners. A bitcoin miner is any processing device that is running the bitcoin mining program[17]. This program joins the computer with the bitcoin peer to peer network. When a bitcoin transaction is attempted a data block to validate the transaction is also created. Processing this data block requires the bitcoin miners to take cryptographic hashes distributed to them and begin computing the block. In effect the

bitcoin reward for computing these blocks is the bitcoin service paying the miners for their processing time. Which is paid by the processing fee of bitcoin transactions.

The bitcoin mining program itself is an interesting program in a distributed system sense because its a cpu bound program without an upper limit. If you devote more computing power to mining, the hashes you receive will become harder to compute. There is still a better reward for bringing more power to the bitcoin peer to peer network. Which leads to some truly impressive bitcoin mining operations, such as the one produced by Dave Carlson[18]. Carlson has set up 5,600 bitcoin mining modules. Each module consists of a Raspberry Pi, an extension board that has 16 slots for a bitfury board. Each of these boards contain 16 bitfury chips that specialize in hash computing relevant to bitcoin mining. Each module is connected to a few linux servers that check the bitcoin mining network for work. The target is delivered to each Raspberry Pi who then funnels the data to the 256 mining chips under its control. Once the chips send data back to the Pi, the Pi sends the computed solution to the linux server. The server then relays the answer back to the network. This monolithic set up represents about 7-10

# Chapter 2

# Planned Experiments

## 2.1 Raspberry Pi Hardware Tests

The hardware tests consists mostly of observing how well the Pis components act under pressure. The three areas that will be given extra attention are the Raspberry Pis temperature to detect any overheat tendencies, SD card quality and specifically its impact on the Pi operation, and the impact of the ethernet port use on the Raspberry Pi.

### 2.1.1 Heat Monitoring

While the Raspberry Pi operating under normal settings, in this case with no overclock, there is no need to worry about heat generation. This doesnt come as much surprise, as the arm chip architecture has a history of use in cell phones and other handhelds. All of which are low voltage devices that would be undesirable if they began to burn the hands of their operators. However the question this test is created to answer is if this quality still holds when we overclock to the highest supported setting of the Raspbian operating system. Information distributed by the Raspberry Pi Foundation includes an automatic disabling of overclocking should the chip reach overheating temperatures of 85 degrees celsius[1].

The first test will be with the Raspberry Pis on a turbo overclock setting while exposed to the open air without any casings. The second test will involve the Raspberry Pi standard clear plastic cases. If these two tests should fail to cause an overheating situation, there will be extra tests to attempt to force an overheating situation. These tests will be performed in a room temperature environment with no active fans for added cooling. A single Pi will have its heat levels monitored through the temperature sensor located on the Raspberry Pi. Cpu throttling will be disabled for the duration of the tests to measure potential heat production. Should the test results show that the Raspberry Pis doesnt approach the critical 85 degree celsius threshold, they will be repeated with the plastic casings attached. Adding the cases will create an environment which will limit the transfer of air around the Pis themselves. This effectively adds a level of insulation around the Raspberry Pis, and theoretically limiting their ability to passively cool.

Each test will involve running an HPL benchmark to keep the Pis processing with an intensive workload. Measuring the temperature will be done through the temperature sensors already existing in the Raspberry Pi hardware. A shell script will monitor the system temperature once every two minutes while the HPL benchmark processes. The task will involve recording the results and stopping the test should the temperature reach the aforementioned 85 celsius threshold.

### 2.1.2 SD Cards

SD cards are effectively the hard drive of the Raspberry Pi. SD cards main target audience however are camera devices. The main way cameras interface with their SD cards are send bulk data, in this case as images, to and from the card. The manufacturers advertise a read/write speed for their SD cards, and in this case its a sequential read/write speed. This can have a potential impact on how well a Raspberry Pi can use the SD card given to it. Operating systems tend to rely heavily on reading specific files anywhere in their

file systems. To find the random access read/write times of the SD card I will be using CrystalMarkDown[19] which allows us to benchmark these values. I will also be loading a Raspbian image on each of the SD cards. The Raspbian images will be observed in how they perform with these SD cards and how the various read and write speeds impact the operating system. The speeds returned by CrystalMarkDown will also be compared with the speeds obtainable in a live environment of Raspbian on the Raspberry Pi.

### 2.1.3 Networking

In a distributed environment networking is a critical component. Cluster builders will want to have the highest speed networks they can manage to minimize the time spent communicating rather than processing. The Raspberry Pi Model B comes with an ethernet port rated for 10/100MBit/s speeds. However this ethernet port is connected to the USB bus[6]. As a consequence, there is a lack of a network controller on the Pi board itself. The network tests will simply be an observation of how the lack of a controller impacts the Raspberry Pi when utilizing network protocols such as SCP and NFS. The workload a controllerless ethernet port can place on the Pi can potentially be a concern when deciding what programs a Raspberry Pi cluster owner will want to run. And thus is worth looking into.

## 2.2 OpenFOAM

OpenFOAM will be one of the first distributable programs to be run on the cluster. My goal with OpenFOAM is to be able to observe a noticeable speedup in computation time through distribution. OpenFOAM itself is a cpu bound program and thus when we distribute the program the cpu workload per chunk of the simulation will be lowered. To measure speedup I will be using a standard tutorial program with the OpenFOAM 2.3.0 distribution called

DamBreakFine[11]. DamBreakFine simulates a solid column of water initially, which during the simulation will collapse and fill the resulting container with introduced turbulence from an added wall at the bottom of the tank.

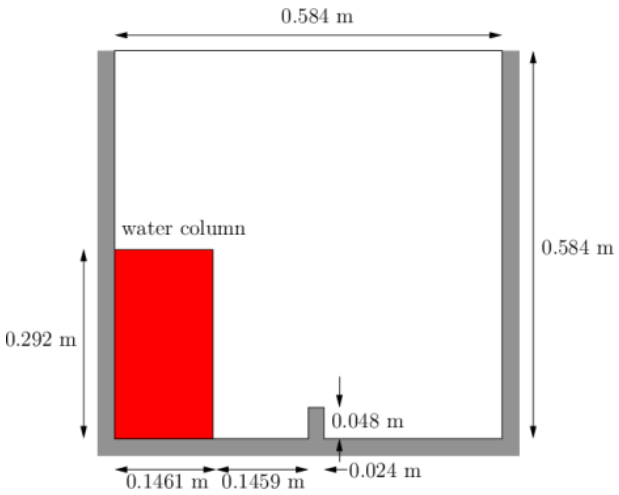


**Figure 2.1**: *damBreakFine Initial State*[20]

To measure the speedup of DamBreakFine I will be running the program multiple times. The first run will contain only one node as the base run time. Each subsequent run will add an additional node to the cluster until the cluster is utilizing all eight nodes simultaneously. The time to completion will be recorded for each run and compared with one another to show any measurable speedup in the simulation creation run times.

## 2.3 HPL

As stated previously the high performance linpack will be used to measure the computational power of the Pi cluster. Since this projects motivation is to generate an inexpensive computer cluster the results of this benchmark test can be helpful for other clusters designed for the same role. The results of this test also can prove useful for future Pi cluster experiments as a baseline to compare this simple eight node cluster with more complicated Pi clusters, or

to compare any improvements/decrements in performance as the Raspberry Pi software or hardware receives updates.

Since optimization plays a heavy role in benchmark results, and the Raspberry Pis arm6 architecture doesnt come with a heavily optimized BLAS library, the project will utilize ATLAS[?] to generate an automatically optimized and tuned BLAS library[21]. Sine ATLAS performs some generalized optimizations this will also lead to a benchmark configuration that can be easily recreated on similar clusters elsewhere.

The entire cluster will be used for the benchmark and the initial value of N will be calculated from the advice given on the HPL website[13]. For an initial N value the following formula will be used:

$N_0 = \sqrt{\frac{M_a * Nodes}{8}} * 0.8$

$N_0$ being the initial value of N to consider in HPL, $M_a$ is the average available memory to each node, and $Nodes$ is the amount of nodes available for the test. The formula is basically a rough estimate of how large the coefficient matrix will have to be to fill the largest amount of available ram to the cluster. For testing there will be multiple benchmarks performed spanning around the value $N_0$. The result with the highest ram utilization should also produce the highest amount of measured GFLOPS, or 1 billion FLOPS. This performance peak will then be compared against the theoretical computation limit for floating point operations of the Raspberry Pi's arm6 chip.

# Chapter 3

# Cluster Construction

This chapter will go over the decisions and methods used to construct the Raspberry Pi cluster. Starting with the hardware making up the physical components of the cluster and continuing on with the setup and installation of relevant programs for this project. The cluster to be produced in this project is an eight node beowulf style cluster. In beowulf style there will be a single master node that will be our interface with the rest of the cluster for job starting, resource distribution, and cluster monitoring.

## 3.1 Hardware

For this cluster eight Raspberry Pi model B was used, as the ethernet port is vital for the cluster. Another reason to favor the model B is it contains double the ram of the model A which will let the cluster run more memory heavy programs. Basic necessities for the Pi include an SD card and micro usb power cable. The SD card used for this project was the Transcend 16GB class 10 SD card. For larger clusters than what this project considers, getting a powered usb hub to distribute more power per outlet can be useful. Since theres only eight Raspberry Pis in this scenario though this project just uses two power strips for

all of the devices in the cluster. For this project the Raspberry Pis each have a transparent plastic case that will be used in the overheat tests but are otherwise completely optional. The tests carried out will be without the case. While the Pis won't overheat inside their cases running normally these will be overclocked to their highest overclock profile. Whether or not these will induce overheating will be determined by later tests when the cluster is complete.

The networking aspect is taken care of by a simple eight port gigabit switch. A 100Mbit switch would be sufficient for the Raspberry Pi cluster, however I used the gigabit switch because I had it on hand. With the Pis plugged into the switch, and the switch and last Pi then plugged into a router the basic hardware setup is complete. Overall the total cost of parts, not taking into account the router or optional plastic cases would come out to under $500.

## 3.2 Raspberry Pi Setup

The next step is to begin producing the master image for the SD cards to be inserted in the Pis. The first step is determining which operating system to use as a base for the cluster. The most popular choice for general Raspberry Pi setups is definitely Raspbian. Raspbian comes out of the box with many programs installed for convenience and has a large amount of community support available[22]. Another operating system looked into for this project was Arch Linux Arm[6]. Arch Linux takes the minimalist approach upon installation, meaning less programs will come installed by default allowing you to easily make a streamlined image for the cluster. An important quality both operating systems share and is important for higher performance with our experiments is the hard float support[6]. Hard float support is important because the Raspberry Pi does contain floating point registers. Hard floating point operation support in this context means that the operating system can utilize the

| Overclock Mode | CPU MHz | GPU MHz | SDRAM MHz | overvoltage |
|---|---|---|---|---|
| None | 700Mhz | 250MHz | 400Mhz | 0 |
| Modest | 800Mhz | 250MHz | 400Mhz | 0 |
| Medium | 900Mhz | 250MHz | 450Mhz | 2 |
| High | 950Mhz | 250MHz | 450Mhz | 6 |
| Turbo | 1000Mhz | 500MHz | 600Mhz | 6 |

**Table 3.1**: *Overclock Profiles*[1]

processors floating point registers. The alternative is soft floating point support. If a program or operating system utilizes soft floating point then, regardless of the chip, the floating point operations will be emulated by using multiple integer registers. Emulating floating point operations this way is much slower than utilizing existing hardware registers. Since the Arm architecture has a wide array of devices that dont contain floating point registers there is a definite trend of general Arm support using soft floating point.

With the Raspbian image loaded onto an SD Card the basic image was ready to be configured with settings and programs common to each node within the cluster. Later this image will be used to produce the master and worker node images which will contain only minor differences.

A Raspberry Pi was loaded with the SD card and booted to begin modifying settings. Raspbian comes with a configuration utility called raspi-config and is a convenient way to set up a single Pi. The settings used in this project where expanding the file system to encompass the entire SD card and enabling the turbo mode overclock profile. As of 2012 Raspbian comes with 5 profiles for overclocking, shown in the table below. These overclock profiles are stable and testing shows they do not impose a significant impact on the lifespan of the device. Utilizing any of these profiles has the added benefit of not voiding the warranty of the device and is supported by the Raspberry Pi Foundation. However end users are still free to attempt their own overclock settings.

Following basic settings the system needs to have an MPI library installed. This project

chose OpenMPI and MPICH2 for their popularity and more importantly their available source code for compiling to the arm6 architecture of the Raspberry Pi.

With this core image created the next step is to clone it onto another SD card, giving us two unique versions of the core for the master and worker. Its worth mentioning that cloning a 16GB sd card can take quite a bit of time. Reading the 16GB file took around 17 minutes, while writing the image to another SD card took 26 minutes. The Raspberry Pi doesnt support booting off of a network image, so each Pi will require an SD card to operate and join the cluster. Another disadvantage that comes from not using network booting is that updating the Raspberry Pi cluster becomes more troublesome. Updating general settings on the Raspberry Pi cluster that contains more than a few nodes can become very time consuming. The first option is to make changes to one Raspbian image and propagate the changes by recloning the new SD image to the rest of the cluster. Cloning the Raspbian images can take quite a large amount of man hours depending on the number of SD writers available, so this method discourages modifications. The second option involves utilizing the raspi-config interface or performing manual changes on each individual node. This method can be just as tedious as the first, however it has the advantage of being automatable through scripting. Scripting is very attractive and its worth knowing that any action that can be performed with the raspi-config tool can be performed on command line or by modifying certain text files. An example of this, when the raspi-config tool is used to modify the overclock profile it creates changes to the /boot/config.txt file which contains all of the settings for overclocking. Being able to manually set changes this way allows the cluster maintainer to automate all actions raspi-config can perform and quickly relay the update to the rest of the cluster.

### 3.2.1 Master and Worker Node Setup

The master node will be unique from the rest of the SD card images since this particular node will be the general access point to the rest of the cluster. To allow the master node to access an arbitrary node on the cluster there will need to be some form of passwordless login setup for the master node. Since some of the distributed programs to be run on the cluster utilize ssh, the cluster was set up for passwordless ssh logins. First the RSA key was generated for the master, and then sent the public key to the initial worker node images authorized_keys file. This allows the master to connect to the initial worker node. Since this worker nodes SD image will be cloned over the remaining worker nodes, the master will also be able to log onto any worker within the cluster. In this project the master node will also be computing, so the public key was also added to the masters authorized_keys file. For ease of distributing files to the entire cluster, the master node was also configured to export directories containing the experiments binaries and base files through an NFS server.

The worker nodes do not need much work compared to the master. Any programs to be tested locally on the machines can be installed before replicating the nodes image to the remaining workers. Since the worker nodes will primarily be gathering most of the binaries from the master nodes NFS shares those network directories are added to the file system table located at /etc/fstab. For this cluster they are set to not mount automatically and only will be mounted when needed with some scripts I wrote for the master node.

## 3.3 Installing and Configuring Testing Programs

The following sections contain information about how the testing programs were installed for testing. Including any installation issues, how they were resolved, and what configuration options were taken.

### 3.3.1 OpenFOAM

OpenFOAM hosts the source code on their main website, as well as releases debian software packages of precompiled binaries for the operating systems they support. Raspbian, and the arm6 architecture in general, isnt supported by the OpenFOAM team. I first attempted to compile the source code locally on the Raspberry Pi itself. This requires modification of more than a few makefiles throughout the build. When compilation began a few critical libraries of OpenFOAM-2.3.0 failed to compile, resulted in compilation errors a few hours into the build. While researching potential solutions to compile these libraries for the Raspbian OS, this was found to be a recurring problem as people tried to run OpenFOAM on their own Raspberry Pi. Though a CFD company, Rheologic[23], has solved these compilation issues and hosts the working binaries for Raspbian specifically on their website.

### 3.3.2 ATLAS, BLAS, and HPL

BLAS libraries can be found many places online, including package repositories used by operating systems used on the Raspberry Pi. For running the high performance linpack benchmark the best version of BLAS to use is a system specific optimized version. The optimization level of BLAS will have a significant impact on the results produced by HPL. Highly optimized binaries are machine specific and typically released by the hardware vendor. This level of optimization couldnt be found for the Raspberry Pi. Instead the project will use ATLAS to generate a tuned version of BLAS.

While ATLAS does support the arm architecture, its default is to generate soft floating point calculations[24]. This can be changed by downloading and utilizing a specific architectural defaults file for ATLAS[?] . Making ATLAS use the new architectural defaults involves editing a configuration file to reference hardfp options and adding a configuration flag when running ATLAS. Other than these changes, the rest of the BLAS generation follows the

standard ATLAS installation and configuration procedures. For the best results of the tuning stage of ATLAS cpu throttling must be disabled. The tuning stage relies on many timed tests. These tests measure different hardware components and their performance at various times during compilation. If cpu throttling is enabled these timed tests will produce incorrect results as the cpu speed changes. Without overriding through the use of runtime flags, the ATLAS configure stage will produce an error if throttling is enabled. Disabling cpu throttling involves adding force_turbo=1 within the /boot/config.txt file located on the Raspbian image citeRP:Rasp. For those interested in maintaining their warranties disabling throttling will void them. Once the configuration and tuning stage begins, expect the Raspberry Pi to be performing these tests and optimizations for at least 26 hours.

# Chapter 4

# Result Analysis

## 4.1 Hardware

### 4.1.1 Heat Tests

Before running the temperature tests the Pis were left idle so that I could assess their idle temperature state. The Pi cluster was oriented in such a way that each Raspberry Pi was around one inch from another Pi for all save the last test performed. The rooms temperature they were left in was noted to be within the 23-24 celsius range. After a few periods of rest and temperature checking the Pis showed an average idle temperature of 45.5 celsius. Each Pi appeared to run naturally hotter or cooler than one another, but stayed within $\pm 2$ degrees celsius of the average. After assessing their idle state the first temperature test began with the cluster as it was. To reiterate how the test was performed, a suitably long test run of HPL was performed to keep the Pis intensively computing. And a shell script in the background would check the Raspberry Pis heat sensor every two minutes.

The first heat test performed was with the Raspberry Pis without their cases attached. In this test run the Pis quickly heated to a running temperature of around 55 degrees celsius. However the heat build up plateaued relatively quickly and the cluster never approached

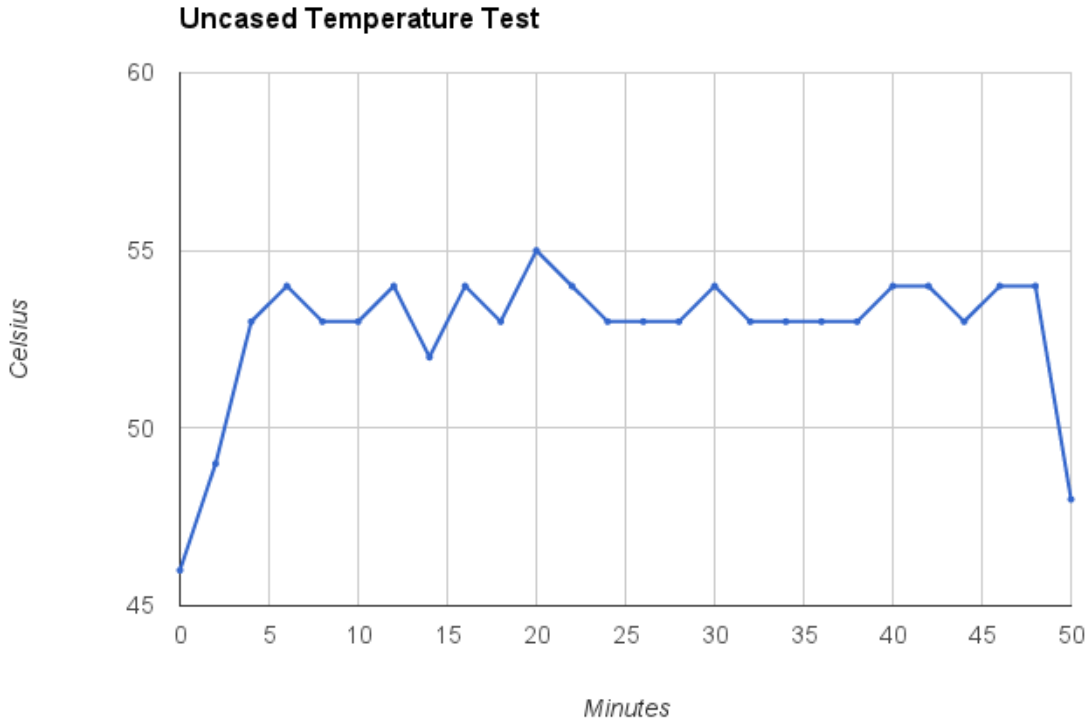


**Figure 4.1**: *Overheating Test Results Without Casings*

the dreaded 85 degrees celsius threshold. For a cluster of Raspberry Pis in a non insulating environment such as a case or closed frame the nodes do not require active cooling, or even extra passive cooling measures such as the heat sink accessories available for the Pi.

After the Pi cluster had time to cool down from the first experiment I placed the cluster within their plastic cases. These cases will restrict some airflow between the Pi and its environment, though it won't be a complete insulation as each case does have openings to the outside world. Most of the openings in the case are to allow access to the Pis connections but in addition to those are two vents that will be under the board as well as two holes where mounting screws can be placed.

The cased experiment began just as the uncased, with a measurement of the idle tem-

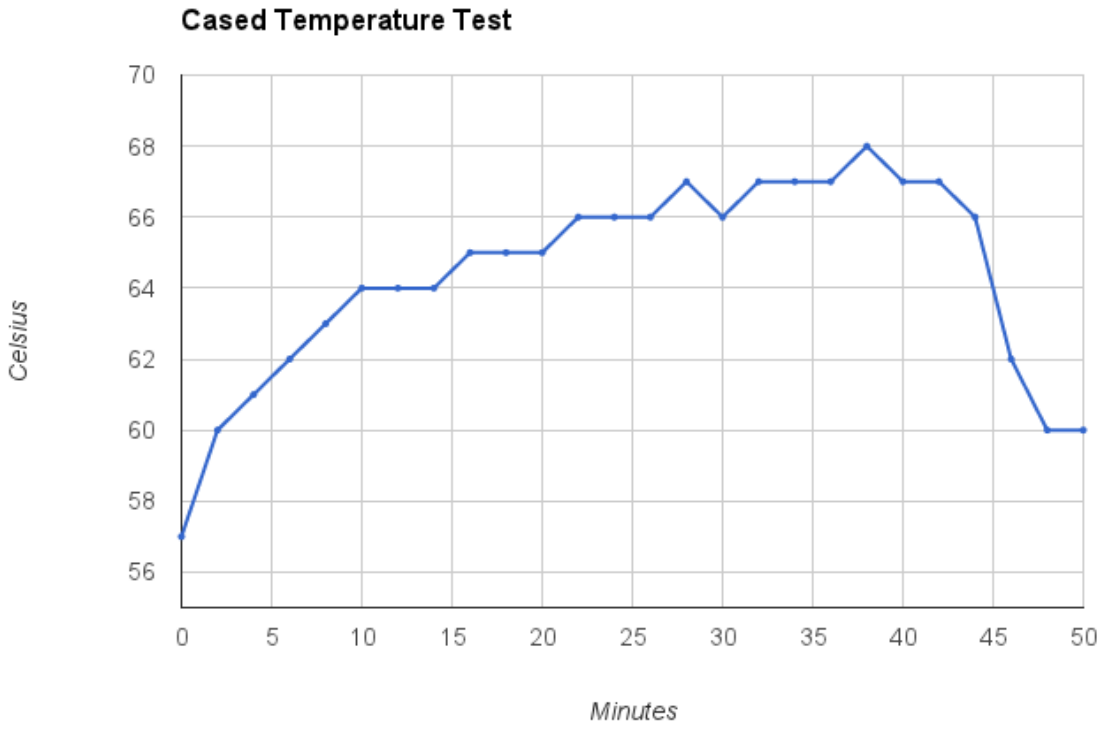


**Figure 4.2**: *Overheating Test Results With Casings*

perature of the Raspberry PI. This time, with their cases attached, the Pis had an idle temperature of 56 celsius. During the HPL benchmark run their temperatures eventually peaked at 68 degrees celsius and remained in the range of 66-68 degree range until the test finished. Once the test ended the nodes began to approach their idle temperatures but at a noticeably slower rate than when they were uncased. This shows that while the cases produce a more insulated environment they alone are not enough to bring the Raspberry Pi board to an overheating state.

These results show that in a room temperature environment the Raspberry Pi hardware will stay below their overheat threshold of 85 degrees celsius even with the Turbo mode overclock profile selected. These results are attributable to the arm chips design towards

energy efficiency with a history of low heat producing chips found in mobile devices that tend to not have any active cooling. This is encouraging for creating a Raspberry Pi node cluster since active cooling is largely situation dependent.

However, since a Pi cluster can come in many different physical orientations I decided to attempt another test run to see if I could force an overheating scenario. A third test was performed where each Raspberry Pi was within their case and I stacked them in a formation that would minimize at least one Pis ability to passively cool. The figure /refoverheatnocase shows the arrangement of the cluster for the third test. This arrangement should give the center Raspberry Pi the largest chance to overheat short of placing it near a major heat source or covering it in a thermal blanket.

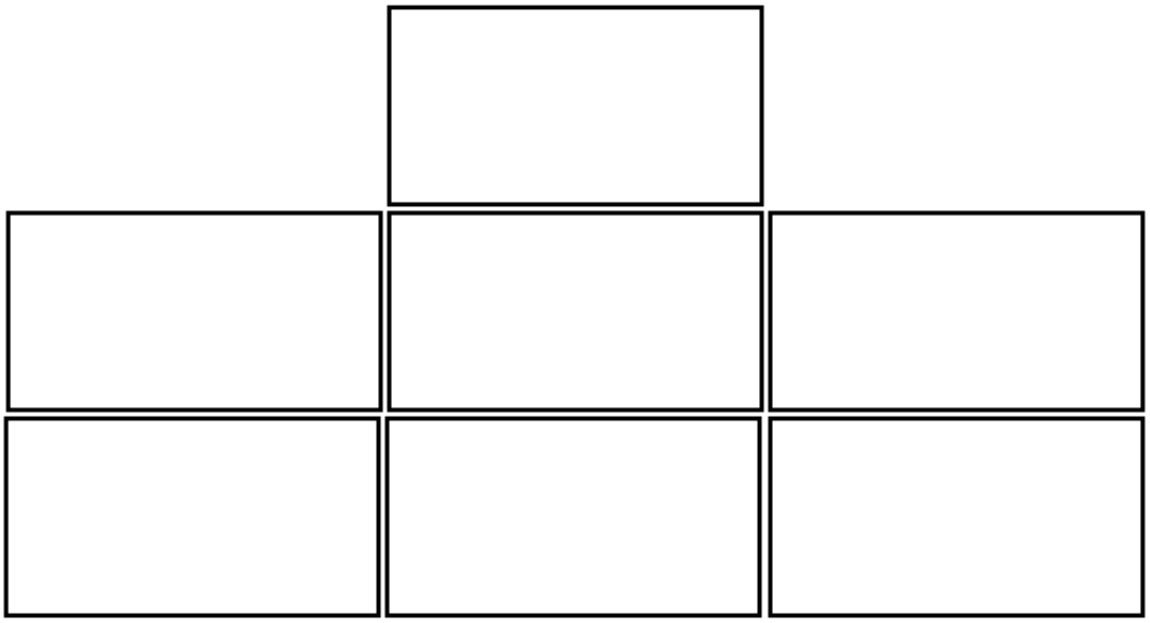

**Figure 4.3**: *Pyramid Stack Diagram*

The thought process leading to this configuration was that the passive cooling with the Pis outside environment was sufficient even through the plastic casing. This configuration however cuts the central Pis access to the environment and worse exposes it to the heat of the other nodes. This test like the others allowed the Pis to return to their idle temperature before stacked. The Pi in the middle of the pyramid formation would be the Pi to be monitored during the HPL benchmark test run.

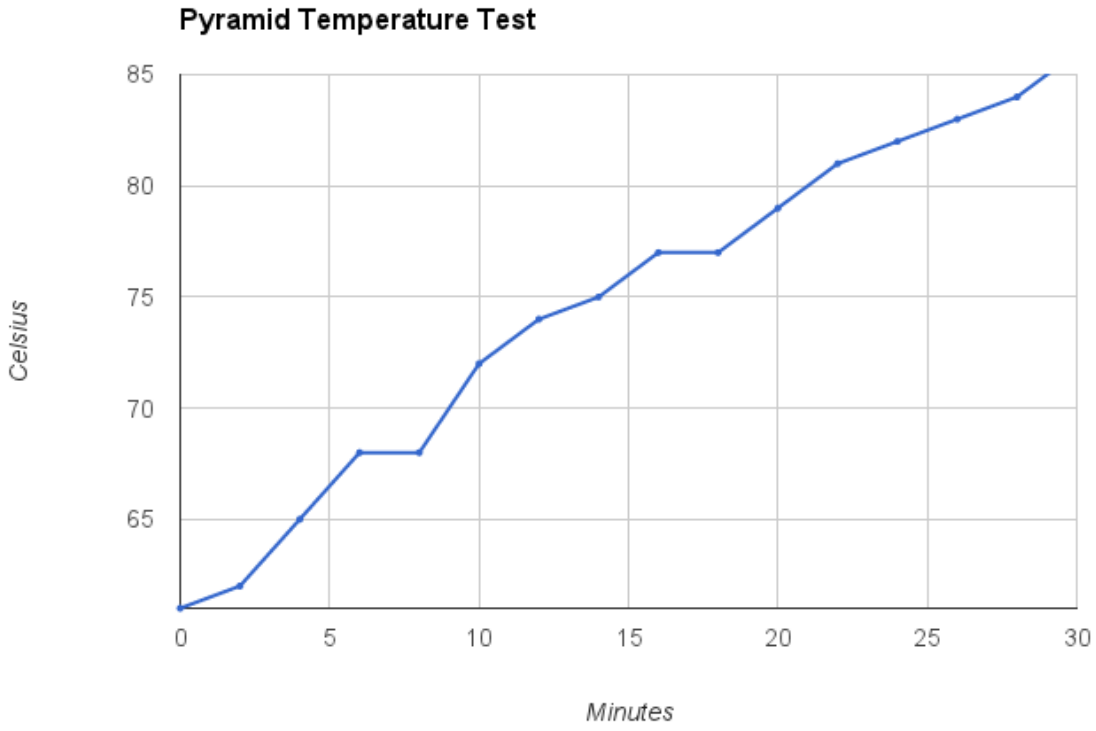


**Figure 4.4**: *Overheating Test Results With Stacking*

The idle temperature reached by the central Pi as expected was higher than previous tests, 60 degrees celsius. This experiment was the only overheat result of the temperature measurement tests. As a precaution, if the measurements hit the 85 degree overheat threshold the monitoring shell script would disable the cluster. Around the 30 minute mark

the measured Raspberry Pi breached that threshold and the cluster shutdown before any hardware damage could occur.

In conclusion the overheat tests where promising to me. In general the Raspberry Pi without overclocking doesnt have to worry about overheating. These tests help show that even with an overclock to 1Ghz the Raspberry Pis passive cooling is still sufficient. The exception of course is if the Pi cluster is placed within a configuration that limits their exposure to the surrounding air or strongly insulates the surrounding air. In these situations some form of active cooling is going to be required such as a small fan. For any future Pi clusters theses tests show that to control heat production the main factor is in the physical orientation of the cluster.

### 4.1.2 Networking Observations

During some of these tests the Pis cpu utilization was monitored. The absence of a network controller on the Raspberry Pi board shows that in practice the overhead of using the ethernet port falls upon the cpu. Any test utilizing NFS had a spike in cpu utilization by the NFS daemons. This is expected as the master node is distributing the binaries not loaded onto the worker nodes. During this distribution the master node cpu utilization will stay at a constant 100

With this information the OpenFOAM tests will not be utilizing NFS to distribute the data. OpenFOAM by default is designed to separate the simulation into distributed chunks. Each node will simulate on a given chunk, writing their portion of the simulation to their own independent file systems. With NFS this will flood the master with unnecessary writes causing an avoidable network overhead to the entire simulation and as noticed beforehand would take up the majority of the masters cpu utilization from OpenFOAM on every write cycle.

### 4.1.3 SD Card Benchmarks

The SD cards showed to have an effect on how the Raspberry Pi performs in general. In some small tests performed to measure the read/write speeds of the three cards on hand. CrystalMarkDown allowed me to determine the speeds of the Pis for continuous and random access read times. CrystalTo see the impact the cards had in Raspbian a sample test to obtain read and write speeds in linux from elinux[25] was performed to observe the read/write rate of blocks in the filesystem.

| Type | Seq. Read | Seq. Write | Raspbian Read | Raspbian Write |
|---|---|---|---|---|
| Raspbian 8GB SD Card | 22.60 MB/s | 6.0 MB/s | 10.8 MB/s | 6.0 MB/s |
| Transcend 16GB SD Card | 22.61 MB/s | 10.26 MB/s | 18.4 MB/s | 10.6 MB/s |
| SanDisk 16 GB SD Card | 22.81 MB/s | 13.8 MB/s | 19.0 MB/s | 13.1 MB/s |

**Table 4.1**: *SD Card Benchmarks*

All three cards had similar sequential read speeds but the Raspbian SD card tends to have a slower sequential write than the others. While the Transcend and SanDisk both showed their class 10 qualities, the Raspbian card appears to be a class 6 equivalent SD card. The benchmarks show that the class however has little impact on random access times, and that for large block sizes the read times approach their sequential reads. But as the block size shrinks the performance of random access degrades with size. For block sized reading and writing in a Raspbian environment approaches the sequential speed of the card used. As far as any observed effects the SD cards have on Raspbian, in general programs tend to initialize much slower when using the Raspbian card. Meanwhile almost

| Type | Read (512KB) | Write (512KB) | Read (4KB) | Write (4KB) |
|---|---|---|---|---|
| Raspbian 8GB SD Card | 22.3 MB/s | 1.14 MB/s | 7.4 MB/s | 1.9 MB/s |
| Transcend 16GB SD Card | 22.20 MB/s | 8.6 MB/s | 5.0 MB/s | 1.39 MB/s |
| SanDisk 16 GB SD Card | 22.5 MB/s | 2.7 MB/s | 7 MB/s | 1.3 MB/s |

**Table 4.2**: *SD Card Random Access Benchmarks*

no difference was noticed between the SanDisk and Transcend cards. This implies that the Raspberry Pis general file system operations can effectively utilize the faster speeds of higher class cards better than I originally wouldve assumed.

An example of the noticeable slowdown was in initializing an SSH connection. For any Pi node utilizing the Raspbian card establishing an ssh connection took at minimum 6.25 seconds while on the Transcend or SanDisk card the connection was established in less than 1.3 seconds. For cluster management this can leave a preference in the higher quality cards. The faster read and write speeds will allow deploying images onto the Pis SD cards in a quicker fashion and allow commands to be relayed through the cluster in a faster fashion. However as a program and its assets are loaded into ram the SD card has very little to no impact. Programs such as running a single thread HPL benchmark showed no difference between SD cards used. However programs that periodically write and read from the filesystem such as OpenFOAM will notice a performance hit due to the slower file system throughput. Programs that also generate lots of page misses due to poor locality will also experience a performance decrease when utilized by lower quality SD cards.

## 4.2 OpenFOAM

The results of the OpenFOAM test runs were interesting. My original hypothesis going into this experiment was to use OpenFOAMs parallelism to demonstrate speedup produced by adding addition Raspberry Pi cores to the distribution. However as shown in figure **??** there is a speedup trend followed by a sharp increase in runtimes. After a certain threshold each additional node added to the computation incurs an increase in simulation calculation runtime.

This led me to begin investigating the OpenFOAM simulation in more detail to find the source of the slowdown. To understand the source I investigated how OpenFOAM

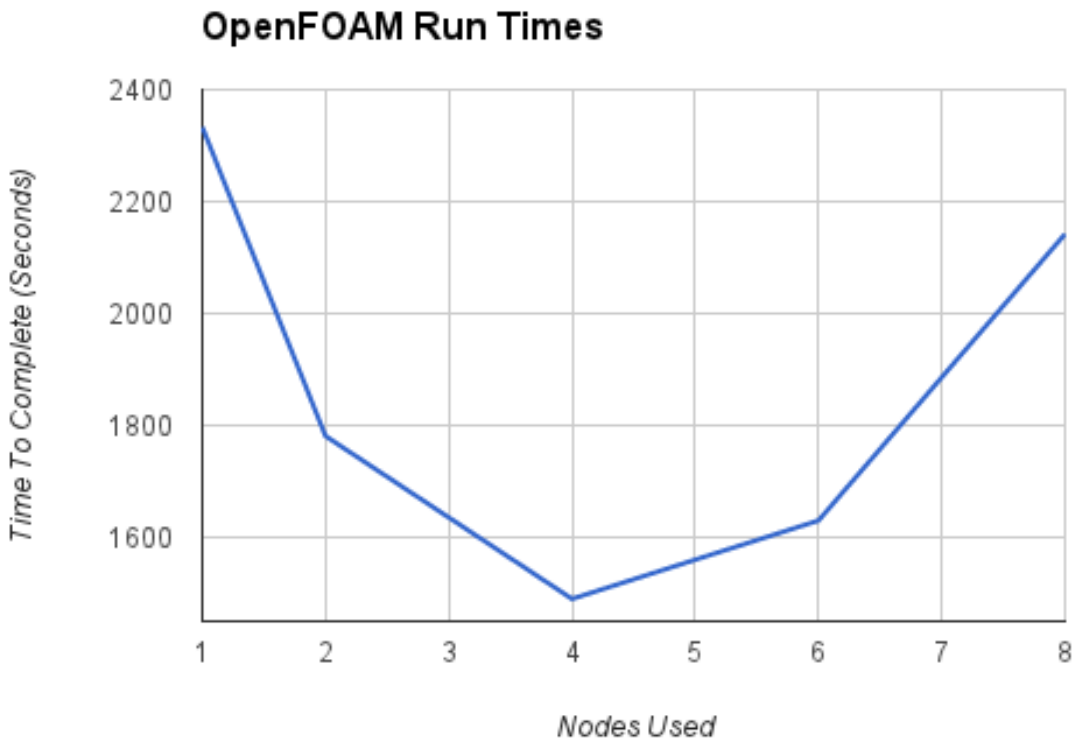


**Figure 4.5**: *OpenFOAM Run Times*

works more closely. OpenFOAM runs in cycles working on a simulation frame followed by writing the results. Then repeat by computing the next frame followed by another write. In this cycle there are 3 major steps, communication and computation followed by an I/O write. Adding more Pis to the cluster will then reduce the computation step, to which I based my original hypothesis upon. The I/O write step also would become faster by adding additional Pis, with each frame written to the file system is effective 1/Nth of the entire frame of the entire simulation. This means adding more Pis reduces how much is written on an individual Pi level. With these realizations the communication step becomes more suspicious. The main cause of the slowdown is related to communication during these tests and deals with how OpenFOAM distributes work between nodes combined with the fact

that this experiment divided the same amount of workload into successively smaller and smaller workloads.

OpenFOAMs strategy for parallelizing simulations is called domain decomposition[11]. Domain decomposition works by taking the entire domain of the simulation, the entire 3D space to simulate, and slicing it into equal sized chunks. For example one can take a 3D cube and cut it in half, now there are two 3D objects to distribute to the nodes. However the borders where the two subdomains touch can interfere with each other. This means that while processor A can work on most of its subdomain without any cooperation, the border of its domain requires assistance from processor B to solve, and likewise there is an area that processor B requires assistance from A to solve. Figure 4.6 visualizes the effect of decomposition over a static problem size. As the static problem set becomes more and more divided the highlighted areas that border the divisions represent the areas requiring the neighboring subdomain to be computed.

OpenFOAM models its 3D space into individual cells, thousands to hundreds of thousands of cells can make up an entire simulation space. These are like points on a graph. A domain with a set number of cells divided in half will then have half of the cells in each sub domain. However the border cells require communicating with the adjacent subdomains cells. The slowdown can be seen as each Pi node being added reduces the amount of work that can be completed alone and increases the amount of communication required. Originally I believed OpenFOAM would be a straightforward case for clustered speedup, but instead has turned out to become a somewhat complicated case of how the ratio of computation versus communication can interfere with the progress of the program.

To view how many cells are divided to each node and how much communication is required is outputted by the OpenFOAM utility called decomposePar. Below is a table that shows the trend of decreasing cell count with increasing communication requirements.

The effect of the increased network complexity on the Pi leads to large increases in

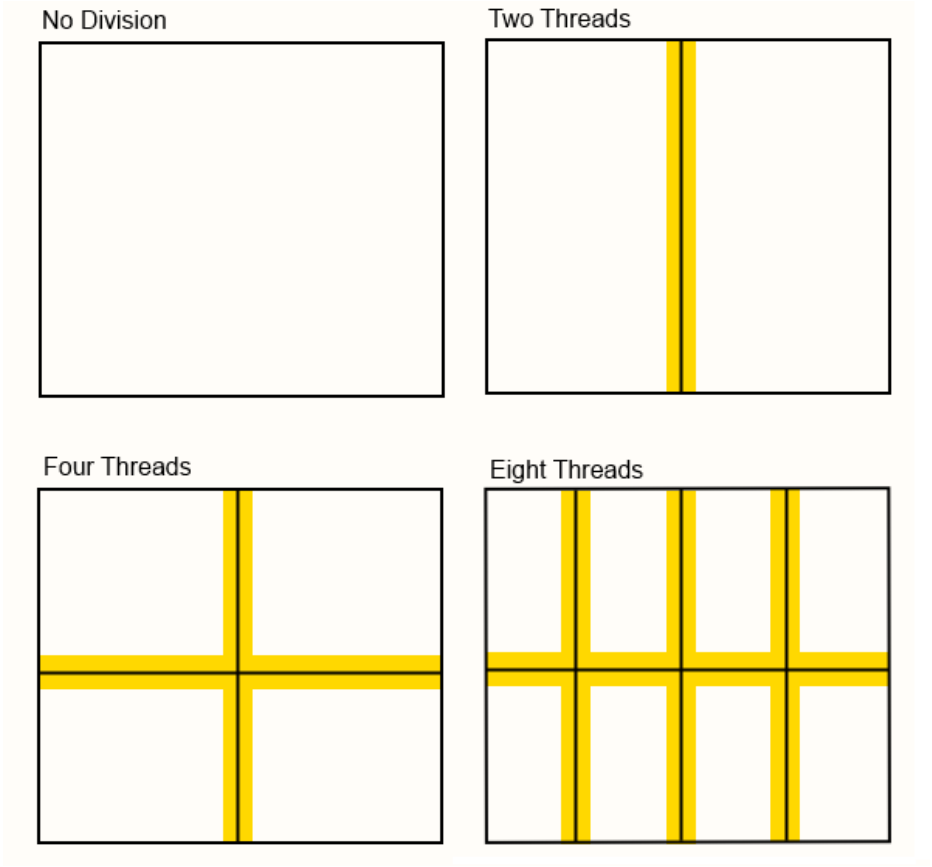


**Figure 4.6**: *Domain Decomposition Example*

| Node Count | Cells Per Node (Average) | Communication Faces (Average) | Ratio (Work/Comm) |
|---|---|---|---|
| 2 Nodes | 3850 | 87 | 44.2 |
| 4 Nodes | 1925 | 89 | 22.6 |
| 6 Nodes | 1283 | 88 | 14.6 |
| 8 Nodes | 962.5 | 88 | 10.9 |

**Table 4.3**: *decomposePar Output*

runtime on OpenFOAM simulations. Ways to increase the workload per Pi would be to increase the resolution of the problem as one increases the distribution. This would also lead to an increased run time of the simulation compilation.

This implies that a given OpenFOAM simulation has a thread saturation limit. Where adding threads after a certain threshold causes program run time increases as opposed to

decreases. For this I ran the program at an increased resolution on my laptop. The increase in resolution allowed my laptop to spend more time on the simulation since it runs at a higher speed than the Raspberry Pi with a 2.6 GHz processor. My laptop experienced an initial speedup, followed by the same slowdown trend of communication vs computation. This shows that the problem can be repeated on any general system and not just specific to the Raspberry Pi. However the Raspberry Pi shows greater increases in run time due to its slower processor and networking compared to running in parallel on a multicore machine.

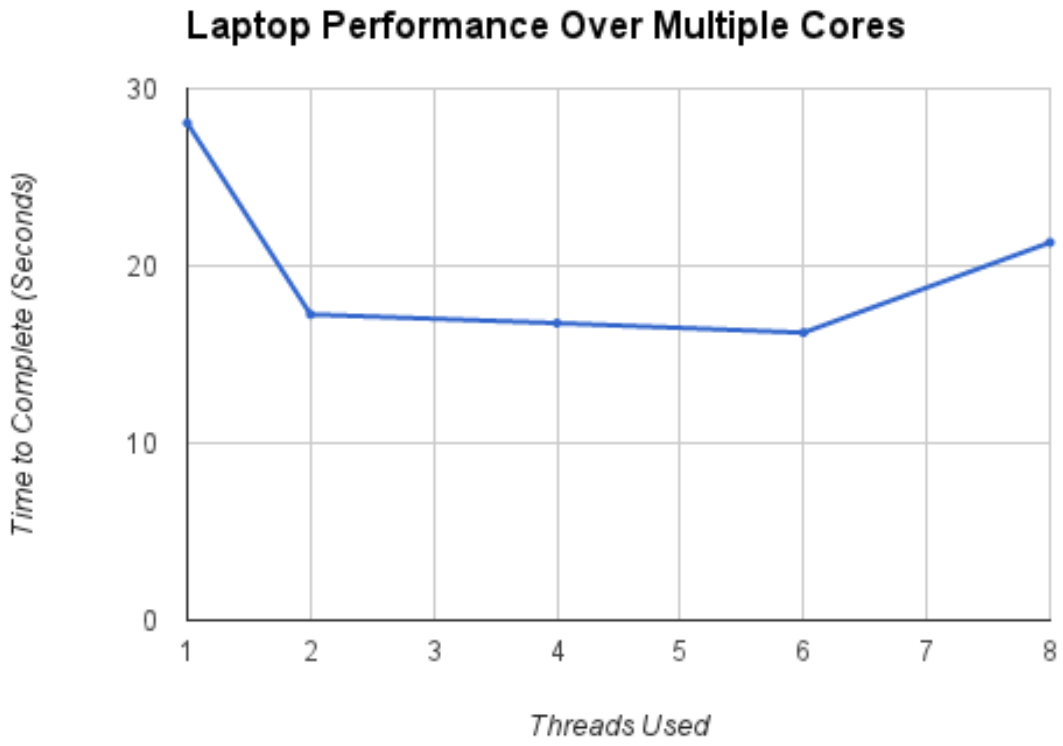


**Figure 4.7**: *Laptop Runtime over Increasing Threads*

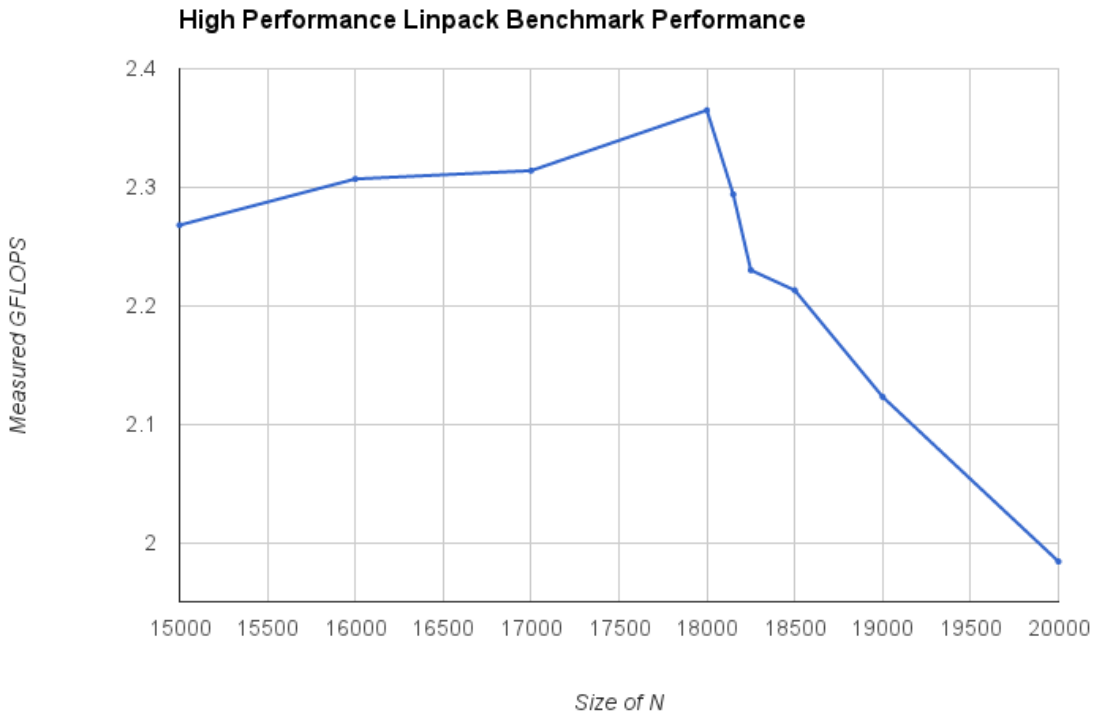


**Figure 4.8**: *HPL Performance Results Over Problem Size N*

## 4.3 HPL

The performance graph shows the measured gflop performance of the cluster over various HPL problem sizes of N. At around N=18000 the benchmark returned a peak performance of 2.365 Gflops. At this value of N almost all of the available ram the cluster owned was filled by the coefficient matrix of the benchmark run. Before this value there is an upwards trend towards the peak as N increases. This is because as the problem size increases the time spent calculating floating point operations tends to overshadow the overhead costs of the rest of the program including message passing[13]. Above the 18000 N size threshold theres a significant trend in decreased measured performance. This is the case because weve exceeded the amount of memory available to the cluster and thrashing begins. Also

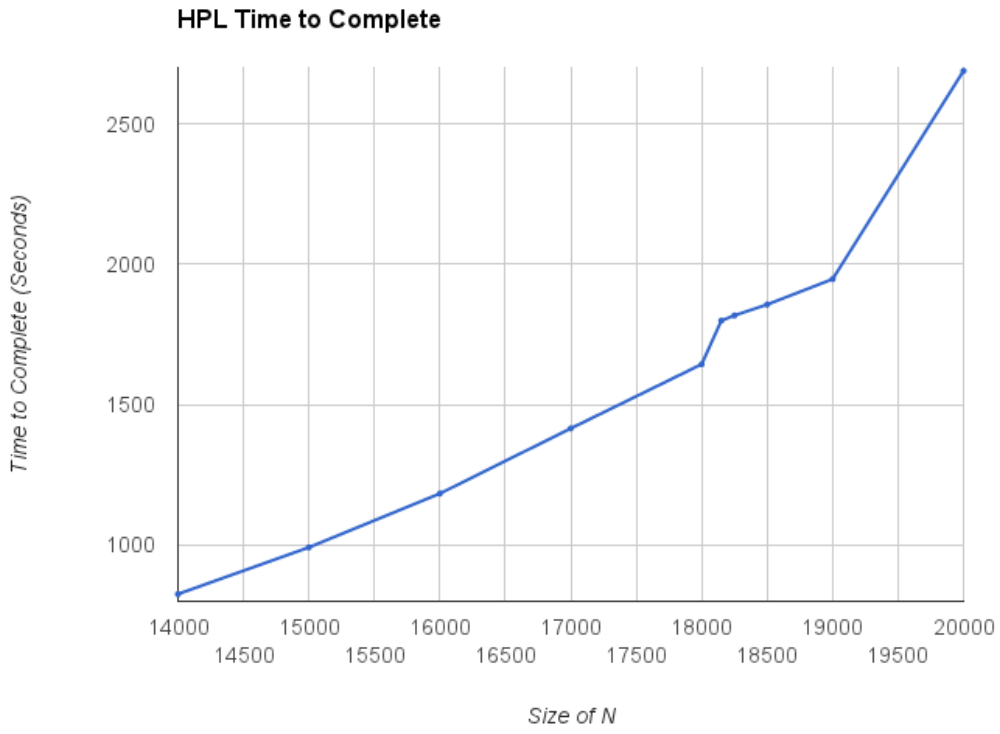


**Figure 4.9**: *HPL Runtime Over Problem Size N*

exceeding the amount of memory available began to lead the Raspberry Pi cluster to have trouble completing HPL at all. This higher range of N will tend to return test cases that do not pass residual checks. The default residual check makes sure the experimental calculated result is within 16.0 of the theoretical result calculated by a formula. However the more the system exceeds the available memory, the more likely the cluster is to fail as individual nodes crash from out of memory errors.

The fastest speed a Raspberry Pis arm6 chip can compute a floating point operation is two cycles[?] . Because of this, our theoretical peak processing power on floating point operations with a 1GHz overclock is 500 MFLOPS, or .5 GFLOPS. Thus the entire clusters theoretical peak computing power comes to 4.0 GFLOPS. With a theoretical limit of 4.0

GFLOPS, and an observed performance of 2.365 GFLOPS our cluster was able to tap into 59.1

# Chapter 5

# Conclusion

The projects original goal was to investigate the Raspberry Pis abilities in a clustered environment. The experiments and general managing of the experimental cluster leads to the conclusion that the Raspberry Pi has potential for educational uses. With the Pi cluster effectively being a scaled down supercomputer the cluster still represents the concepts and principles of high performance computing. The high performance linpack demonstrated the the Pis ability to cooperate and solve parallel problems while OpenFOAM showed the principles of message complexity. Programs that approach trivial levels of communication such as many matrix operations would however show a more appropriate speedup that is well within the properties of HPC concepts such as Amdahl's law. This allows the Pi cluster to easily become a candidate for demonstration purposes for project or lab work within a computer science curriculum. While LittleFe was created to make a clustered system affordable for any education facility the Raspberry Pi has shown to be a viable alternative with this less than $500 dollar computer cluster created for these experiments.

## 5.1 Future Work

The concept behind this project can benefit from continued research. With the release of the Raspberry Pi, the popularity of a single board computer has grown in the market. Other small systems on a chip have began to appear at similar price points. Most notable is the BeagleBone Black. At $45 dollars with similar levels of hardware the BeagleBone Black has similar potential for computer science education as the Raspberry Pi does. While more of these single board computers appear on the market they may create clusters with different strengths than the Raspberry Pi. With these other candidates clusters could also be constructed to be heterogeneous to demonstrated other HPC concepts such as load balancing or high availability. Due to the trend of introducing computer concepts such as basic use and programming to earlier middle school levels these clusters can also be available for introducing these more advanced concepts of computing in outreach events. As inventors such as the bitcoin miners have shown the Raspberry Pi can be very versatile for its $35 price point and it along with other single board systems deserve further research in their potential uses.

# Bibliography


[1] E. Upton. Introducing turbo mode: Up to 50. URL http://www.raspberrypi.org/introducing-turbo-mode-up-to-50-more-performance-for-free/.

[2] B. Brown J. Bughin R. Dobbs C. Roxburgh J. Manyika, M. Chui and A. H. Byers. *Big data: The next frontier for innovation, competition, and productivity.* McKinsey Global Institute, May 2011.

[3] A. McAfee and E. Brynjolfsson. Big data: The management revolution. *Harvard Business Review*, October 2012.

[4] J. Listgarten D. Heckerman and C. Lippert. Identifying genetic factors in disease with big data. Science@Microsoft.

[5] *LittleFe Overview.* LittleFe.

[6] Raspberry pi documentation.

[7] R. G. Brown. *Engineering A Beowulf-style Compute Cluster.* Duke University Physics Department, May 2004.

[8] K. Parrish. Raspberry pi model b sold out within minutes. URL http://www.tomshardware.com/news/Raspberry-Pi-Eben-Upton-RS-Components-Premier-Farnell-Linux,14851.html.

[9] *ARM1176JZF-S Technical Reference Manual.* ARM Holdings.

[10] E. Upton. Oracle java on raspberry pi, . URL http://www.raspberrypi.org/oracle-java-on-raspberry-pi/.

[11] Features of openfoam. URL http://www.openfoam.org/features/.

[12] A. Kosik. The cfd simulation of the flow around the aircraft using openfoam and ansa. URL http://www.beta-cae.gr/events/c5pdf/8B_1_kosik.pdf.

[13] Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers.

[14] W. Kramer. Top problems with the top500. URL http://www.ncsa.illinois.edu/news/stories/TOP500problem/.

[15] Littlefe overview. URL http://www.shodor.org/media/content//petascale/materials/general/presentations/littlefe-overview_pdf.pdf.

[16] LittleFe Inc. Littlefe homepage, . URL Littlefe.net.

[17] Bitcoin Inc. What is bitcoin?, .

[18] J. Brodkin. Meet the manic miner who wants to mint 10 URL http://arstechnica.com/information-technology/2014/03/meet-the-manic-miner-who-wants-to-mint-10-of-all-new-bitcoins/.

[19] K. Kasumu. Crystal dew world. URL http://crystalmark.info/?lang=en.

[20] OpenFOAM. Dambreakfine openfoam tutorial. URL http://www.openfoam.org/docs/user/damBreak.php.

[21] Netlib. Netlib blas. URL http://www.netlib.org/blas/.

[22] Raspberry Pi Foundation. Rasbpian. URL http://www.raspbian.org/.

[23] Rhelogic. URL http://rheologic.at/.

[24] R. C. Whaley and J. J. Dongarra. URL http://www.netlib.org/lapack/lawnspdf/lawn131.pdf.

[25] elinux. Rpi sd cards. URL http://elinux.org/RPi_SD_cards.