Raspberry Pi Cluster

Derek Haas
Dustin Knight
Glenn Lumbert

1. Introduction

For a final project, a Raspberry Pi cluster was created using available Raspberry Pis. The goal of this project was to create a fully functional distributed system with Raspberry Pis. To test if the cluster worked, message passing interface (MPI) code developed in previous homeworks were used as our baseline test for the cluster. This MPI code involves convolving a sobel operator matrix with the pixel values of an image to produce an output. The sobel operator is a type of edge-detection matrix used for removing all of the image except “edges”, where the image pixels contrast sharply. In addition to running the MPI code, timing and power measurements were recorded to show both improvement in computation time and increased power draw as more nodes are utilized.

Section 2 covers related work about Raspberry Pi clusters. Section 3 covers the hardware used for the cluster. Section 4 covers the software used. Section 5 gives a project overview and shows results gathered. Lastly, Section 6 concludes the report.

2. Related work

The first related work found concerns the affordability, low power consumption, and small computational ability of Raspberry Pi’s and how a cluster of Pi’s can improve the computing power. The authors imply that these Raspberry Pi’s can be used as an inexpensive means of testing cloud computing research [1]. The ways that our work differs is we implement the proposed Raspberry Pi cluster talked about in this article.

The second author creates a Beowulf Raspberry Pi Cluster using 32 nodes, each made of a Raspberry Pi. The Pi’s are connected similarly to how we are connecting our Pi’s together, which is through a network. They use MPI to compare the performance of the Raspberry Pi by parallelizing different numbers of nodes. The success of the author highlights the validity of our approach towards creating a Pi cluster [2]. Our project uses significantly fewer Pis than the Beowulf Raspberry Pi Cluster, however we implement slurm for job scheduling unlike this author.

The third related work concerns a short tutorial on how to set up MPI to run on a cluster within a local area network [3]. This source was used a starting point to familiarize ourselves with an idea of how to go about running MPI on our cluster.

A fantastic related work, this article compares the memory, cost, GFLOPS, power consumption and other power related measures of seventeen different ARM boards to each other, then a cluster is built from 25 Raspberry Pis as opposed to our 3 Raspberry Pis. Power and performance measurements will be made on our Raspberry Pi cluster, so this is a good related work to compare results [4].

The last related work is a much larger scale of our project. The technical report discusses the hardware layout, systems, and testing of a 68 node cluster. The 68 node cluster utilizes slurm for resource management and job scheduling as we will and all the nodes share a network file system (NFS) much like ours does. The 68 node cluster was a success and provides further evidence for the feasibility of our 3 node cluster [5].

3. Hardware

For the Raspberry Pi cluster, 3 Raspberry Pi’s were used. This cluster is a distributed system as each Pi has its own CPU and memory. A Raspberry Pi 3B was used as the head node and 2 Raspberry Pi 3B+ were used as the compute nodes. The Raspberry Pi 3B has a quad core 1.2 GHz ARM Cortex-A53 Broadcom BCM2837 64-bit CPU with 1 GB of RAM.
The Raspberry Pi 3B+ is slightly better as it has a 1.4 GHz 64-bit quad core ARM Cortex-A53 CPU and 1GB LPDDR2 RAM at 900 MHz. Since the 3B+ is slightly faster than the 3B, it was chosen as the compute nodes. It should be noted that these Pi models were chosen as they were the only ones available to the team.

After the Pi models were chosen, an ethernet switch was obtained to connect all the Pi’s on a reliable network. A NETGEAR 5 port gigabit ethernet switch and three 100 megabit ethernet cables were purchased. The speed of these devices are found to be fast enough for the purpose of this project. When the Pi’s are connected to the ethernet switch, each Pi is assigned an IP address from the switch (different than a public IP) so that each Pi may communicate with each other.

To keep track of power usage, all devices were plugged into a power strip. This power strip was then plugged into a “Kill-A-Watt” meter. This allowed the team to measure the total power draw from the 3 Raspberry Pi’s and the ethernet switch when parallel code is running.

4. Software

The software that was used for this Raspberry Pi cluster ran on the Raspbian operating system kernel 4.14 and involved MPI, Slurm, and a network file system (NFS) client and server. MPI is the message passing interface that is used in distributed system to pass information between devices. Slurm is a common job scheduler that is used to queue jobs and processes on the cluster. NFS is a file system that was set up to easily pass files between machines.

The programming language used for MPI was C. Parallel code that used MPI was written in homeworks 6 and 7 and was ported to the Raspberry Pi cluster for testing. The C programming language is a reliable low level programming language. In addition to reliability, C is also very powerful, giving programmers a significant amount of power over memory access and variable casting. Software errors may still occur when using C. Because the programmer has more access to memory and variables, this may allow a program to access memory that is not meant to be accessed. With memory access being an issue, system security becomes something to worry about. Proper error checking should be handled in code to prevent bugs that can lead to memory that is not meant to be accessed.

The MPI code that was written performs a sobel convolution on an image. This process can be done in parallel as each pixel can be done separately. The code splits the image up based on the amount of threads specified, and each core of the system will perform the sobel convolution on that part of the image. Once the convolution is complete, each core sends its data back to the first thread to combine the results into a final image.

There were not any hardware errors encountered, however hardware errors should still be taken into consideration. Because our system is simply doing image processing, there is no real threat of the hardware failing. If a larger image was to be processed, more time would be taken to complete the convolution. In this case, a checkpoint of the convolution may need to saved in the event that the system loses power or fails. This is to ensure that when that node is restarted, it does not have to begin from the beginning.

As mentioned, Slurm is a common job scheduler that is used to queue jobs on a distributed system. Slurm was used on the homeworks to queue jobs to get accurate timing results. This was attempted to be replicated on the Raspberry Pi cluster to give the user the ability to queue processes. However, Slurm was taken for granted on the Haswell-EP machines and was found to be difficult to configure. After configuring Slurm on all the nodes, the computing nodes were reporting to be down (inactive) when they proven to be up by the group. This took some time to understand the error as no one in the group had experience with Slurm.

The last software component used was the NFS. NFS is a file system that the group did not have experience with, but was able to find a tutorial online that could easily walk through the process. Essentially, the “hosts” file on all the machines was configured with all of the nodes IP addresses that were assigned by the ethernet switch. A simple NFS server was set up on the head node while NFS clients were configured on the compute nodes. To test that the NFS worked, a test file was created and placed in the shared directory. This file appeared on all nodes, showing that the NFS was configured correctly.

5. Project Overview and Results

A Raspberry Pi cluster was used to parallelize image processing code to improve the overall time to complete. The cluster was constructed with three Raspberry Pis or “nodes” and connected together via a network. The network connection between the Pi’s allow messages to be passed within the cluster using MPI. An NFS is also created to share files. The MPI code used to process the image passes sub-sections of the image to different cluster nodes to be convolved at the same time. By convolving different parts of the image simultaneously, the overall time to process the image decreases. In addition to MPI, slurm was configured for the cluster in order to schedule jobs/tasks on the cluster.

To test the MPI code, a different amount of threads was used to determine the speed up when using more threads. The number of threads tested included 1, 2, 4, 8, and 16 threads. It should be noted that when the number of threads are below 4, the convolution code is still running on 1 Raspberry Pi, as each thread utilizes 1 core. Figures 1, 2, and 3 show the timing and power results for the specified threads.

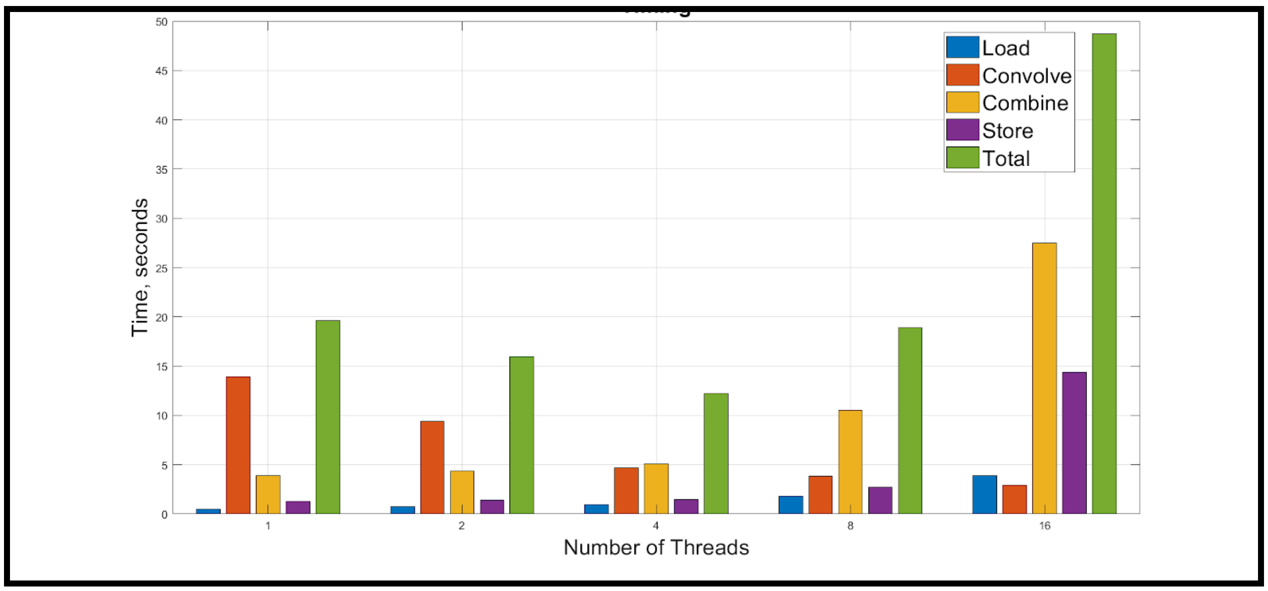


Figure 1. Timing Results for MPI code

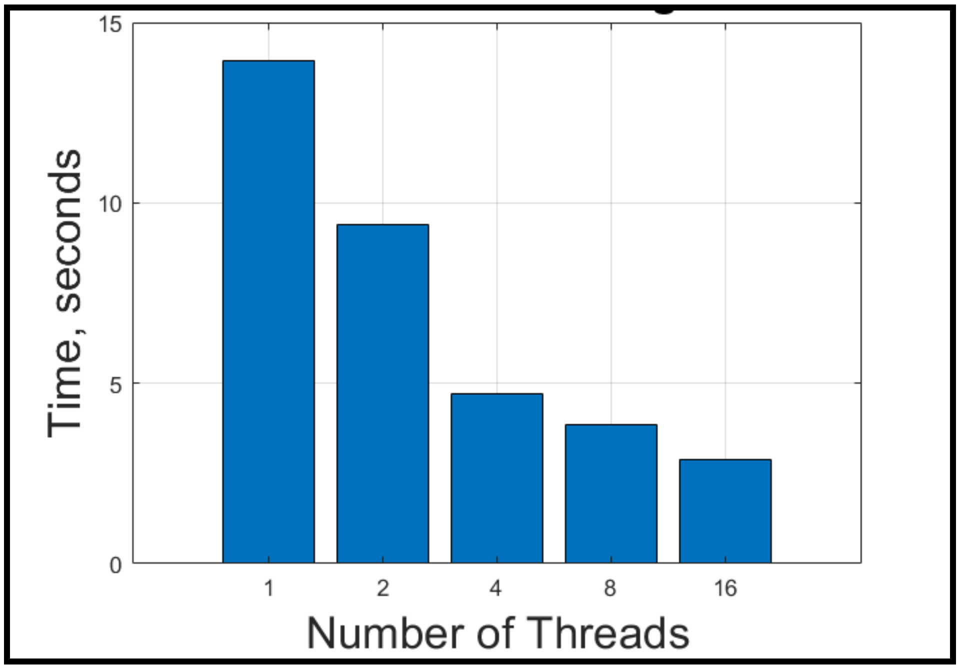


Figure 2. Convolve Timing Results

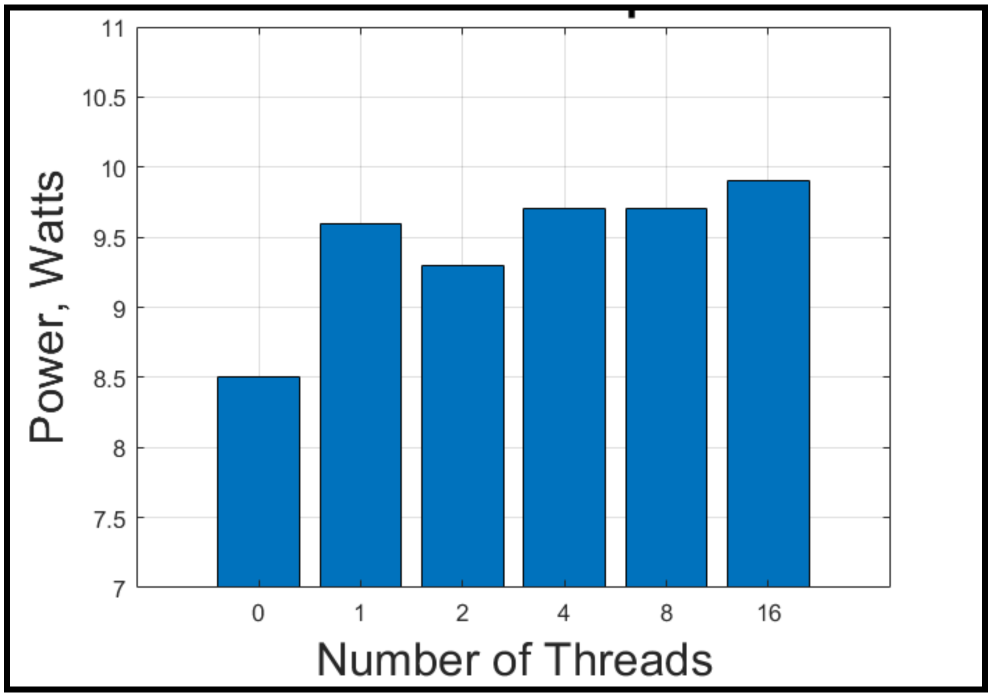


Figure 3. Power Consumption

It can be seen in Figure 1 that the total time to complete the image processing decreases up until 8 threads. As mentioned, when less than 4 threads are used, the image processing is done on a single Raspberry Pi. When the threads are greater than 4, message passing between the nodes occur. The time to pass messages between the Pi's takes quite a bit of time in relation to the total time, thus slowing down the convolution.

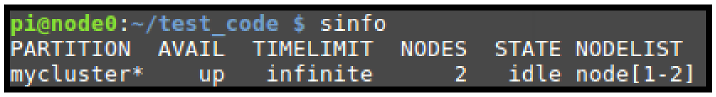
The image processing code was written to speed up the convolution portion of the code, thus the only important time to look at is the convolution time. Figure 2 shows just the convolution timing for all the threads used. The last results that were gathered are the power consumption using all the threads. The power consumption can then be translated to energy, energy delay, and energy delay squared. Table I shows the speedup and parallel efficiency of the convolution code and the energy, energy delay, and energy delay squared to process the entire image.

TABLE I: EFFICIENCY VARIABLES

Threads	Total Time (s)	Energy (J)	Energy Delay (Js)	Energy Delay Squared (Js^2)	Speedup of Convolution	Parallel Efficiency of Convolution
1	19.6	166.6	3265.4	64001.1	N/A	N/A
2	15.9	147.9	2351.6	37390.6	1.48	0.740
4	12.2	118.3	1443.7	17613.7	2.97	0.743
8	18.9	183.3	3464.9	65487.3	3.63	0.454
16	48.7	482.1	23479.7	1143461.4	4.81	0.301

In general, all results showed to improve up until 8 threads. Once 8 threads was used, more time and power was used, as expected.

The last portion of the project to get working was Slurm. Slurm was not a priority to get working as it is not necessary to have, although it would be nice to have a proper job scheduler running on this distributed system. As mentioned, it was found that the compute nodes kept reporting that they were down, when in fact they were active. After researching the issue, it was found that the compute nodes were reporting to be down as they had rebooted. There is a flag in the slurm config file called "ReturnToService" that had to be set to a value of 2 to allow the nodes to report that they are active. Figure 4 shows an sinfo command that proves all the nodes are idle and active.



```
pi@node0:~/test_code $ sinfo
PARTITION  AVAIL  TIMELIMIT  NODES  STATE NODELIST
mycluster*    up   infinite      2   idle node[1-2]
```

Figure 4. Status of the nodes from Slurm.

6. Conclusion

The following table shows the topics worked on by each individual in the group:

TABLE II: GROUP MEMBER CONTRIBUTIONS

Group Member	**Derek**	**Dustin**	**Glenn**
Topics	Pi setup, Slurm Config, NFS Config, MPI Config, Fixed Slurm	Related work, Pi Setup, Slurm Config, NFS Config	Pi Setup, Slurm Config, NFS config

The assembly and usage of a Raspberry Pi cluster was accomplished in this project. The cluster uses NFS for file transfer and runs MPI code written in the C programming language. The cluster produced a significant decrease in computation time when more threads were utilized.

For future work, it would be nice to have a fully functional job scheduler work with MPI. It was found that Slurm was hard to configure with MPI in the homeworks, thus not much time was spent getting it working for this cluster. Another improvement could be to add more nodes on this cluster. This would require the team to purchase more Pi’s, ethernet cables, and get a larger port ethernet switch. Overall, the cost a Pi cluster is cheap in comparison to other clusters. On a small budget, the necessary hardware for a three node cluster is easily attainable.

References

[1] P. Abrahamsson, S. Helmer, N. Phaphoom, et al, “Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment,” *IEEE 5th International Conference on Cloud Computing Technology and Science*, 2013.

[2] J. Kiepert, “Creating a Raspberry Pi-Based Beowulf Cluster,” Boise State University, Idaho, Tech. Report. 22 May 2013.

[3] D. Nath. “Running an MPI Cluster within a LAN.” Internet: http://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/, 2019 [April 2019].

[4] M. F. Cloutier, C. Paradis and V. M. Weaver, “*A Raspberry Pi Cluster Instrumented for Fine-Grained Power Measurement*,” University of Maine, Orono Maine. Report. 30 April 2016

[5] J. P. Mitchell, A. R. Young, J. Sangid, K. A. Deuso, et al, “PERFORMANCE, MANAGEMENT, AND MONITORING OF 68 NODE RASPBERRY PI 3 EDUCATION CLUSTER: BIG ORANGE BRAMBLE (BOB),” University of Tennessee, Tennessee, Tech. Report. 2017.