Raspberry Pi Computer Cluster

Noel J. Petit
Ken Johnson
Paulina Vo
Don Vo
Christian Grant

April 10, 2015

Computer Science
Augsburg College
2211 Riverside Avenue
Minneapolis,MN 55454
petit@augsburg.edu
vop@augsburg.edu

1 Abstract

A cluster of Raspberry Pi computers has been created to test and demonstrate the effectiveness of small computers in parallel processing. The computers use MPICH, a message passing protocol to share a large task and then coordinate their results at the end of the processing among a group of 8 or more Raspberry Pi computers. We compare the single CPU (such as the common Intel Core i5) vs cluster performance of various parallel algorithms such as sorting and searching. We found limitations in the ability to speed up a process because of the communication and operating system overhead among processors. We tested various operating systems and configurations for their ability to speed up a process shared by many CPU's. Results were improved by using simpler operating systems and limiting the tasks assigned to the Raspberry Pi. For example, by removing services such as Network Time Protocol, Remote Desktop, and system logging, we were able to approximately double the processing speed of an 8 node parallel system. In addition, we tested file sharing with NFS and SecureNFS as well as file sharing with file systems outside the cluster (for example, Google storage). We will review the technical findings as well as what we learned by building a cluster of small computers to simulate a high performance processor.

2 Introduction

Created a cluster of inexpensive Raspberry Pi computers to demonstrate the power of parallel computing. Each node of the cluster is a Raspberry Pi minicomputer in a family of computers introduced in 2012 as a full-featured minicomputer for $35. At that cost, it is possible to acquire many computers and network them via Ethernet or WiFi. In our case we used an Ethernet switch to join 6 to 8 model B Raspberry Pi's and demonstrate various parallel processing features.

3 Raspberry Pi

This minicomputer has passed through three versions – B, B+, and 2. Here is a rough comparison of the models available.

Model Number	B	B+	2
Processor	Broadcom BCM 2835	BCM 2835	BCM 2836
CPU Speed	700 MHz Single Core ARM 1176	700 MHz Single Core ARM 1176	900 MHz Quad Core ARM Cortex A7
Memory	512 MB SD RAM	512 MB SD RAM	1 GB SD RAM

All have USB 2.0. ports, audio and video outputs as well as Ethernet and power connectors. Each draws about 700 ma at 5 volts for a power draw of about 4 watts.

A number of operating systems are available. The install manager for the Raspberry Pi is NOOBS. The operating systems included with NOOBS are:

- Arch Linux ARM
- OpenELEC
- Pidora (Fedora Remix)
- Puppy Linux
- Raspbmc and XBMC open source digital media center
- RISC OS - The operating system of the first ARM-based computer
- Raspbian

In our case we wanted to have as many features and languages as possible so we used the Raspbian available from the Raspberry Pi foundation. That may not be the fastest of operating systems but it is very close to the Ubuntu that students use in our public lab and provided the widest of features. The B and B+ processors were measured at about 40 Megaflops with the quad core 2 processor at 98 Megaflops (1). On the CPUlevel the performance is similar to a 300MHzPentium IIof 1997-1999. Connecting machines is done with a simple 100 Mbps Ethernet switch. Each processor in our cluster is assigned a fixed IP.

4 MPICH

The MPICH consortium of contributors who develop and support a library or message passing protocols for communication among computers. MPICH includes compilers for FORTRAN, C, C++. In our case we chose C as our language and used many of the example parallel programs to demonstrate the cluster. MPICH includes a specialized compiler and run time for C which manages the distribution and collection of tasks and data among connected processors. To allow processors to share compiled programs and their data, all computers share files via Network File Sharing (NFS) on either one of the Raspberry Pi's or a separate Ubuntu server running NFS server.
For simple programs, MPICH starts all of the programs on as many processors as specified and runs all to completion. Each processor is aware of how many other processors are in the cluster as well as its index in the array of processors. Thus, every processor knows who it is as well as how to address all of the other neighbor processors. Some of the programs distribute the task among processors by breaking the shared data into blocks. Some distribute the tasks by sending data to each processor and waiting for the "master" processor to receive data from all the slaves.

5 Parallel vs Single Processor Tasks

As a start, let's consider running a simple mathematical task on a single processor and then distributing this task among several processors. There will always be start-up time spent distributing the task to multiple processors so we expect short tasks to take longer when distributed among processors.

5.1 Calculating Prime Numbers

Prime MPI calculates prime numbers and distributes the work amongst the various numbers of processors. Prime MPIs code is derived from jburkardt@fsu.edu. The work is divided up amongst two Raspberry Pi 2s. Each Raspberry Pi 2 has quad core capabilities. The way the work division is represented in this paper is shown as ...

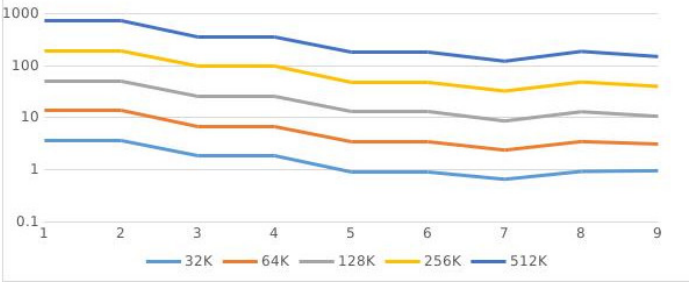
(**Raspberry Pi #1**) / (**Raspberry Pi #2**)

The work distribution of the two Raspberry Pis was tested on prime number lists of size $3.2 \cdot 10^4$, $6.4 \cdot 10^4$, $1.25 \cdot 10^5$, $2.56 \cdot 10^5$, and $5.12 \cdot 10^5$. The predicted data for this segment will show a logarithmic plot for the graph. It is also predicted that the graph will display close to the same results between processes between corresponding division. For example, 1/0 will display the same data as 1/1. The process should run faster, however the time must also account for the communication between Pis. It is also predicted that at 5/4 and 5/5, the runtimes will increase since the code will divide up the processes to be run on more processes that exist. The data collected is shown below.

Number of Processes per Pi

Number of Primes	**1/0**	**1/1**	**2/1**	**2/2**	**3/2**	**3/3**	**4/3**	**4/4**	**5/4**	**5/5**
32,000	3.66	3.66	1.84	1.84	0.92	0.92	0.65	0.92	0.95	1.25
64,000	13.6	13.6	6.84	6.84	3.4	3.4	2.37	3.46	3.1	
128,000	51.3	51.3	25.7	25.7	13.0	13.0	8.62	13.0	10.6	
256,000	192.8	192.8	96.6	96.6	48.5	48.5	32.5	48.6	40.1	
512,000	729.9	729.9	367.3	367.3	183.6	183.6	122.4	188.6	150.1	

Using Microsoft Excel, the data was entered and compiled to form the graph below. The horizontal axis represents the number of processors the work is distributed amongst. The vertical axis represents the runtime for each test.



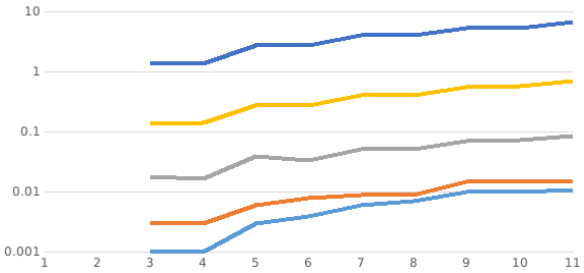
As shown from the data collected, the predictions made were true. There was some variation once the program was run on eight (4/4) and nine (5/4) processors. The variation between the two is fairly small except for size 512,000 (difference of 66.2). There is a max of eight processors, so when the code was tested on nine processors the runtime decreased which was also predicted.

5.2 Ring MPI

Ring MPI sends messages of size 100, 1000, 10000, 100000, and 1000000 from processor 0 to 1 to 2 to ... to P-1 then back to 0. P represents the number of processors used. We expect the program to take longer with more processors since the message needs to be relayed to more processors before returning to processor 0.

Number of Processes per Pi

Values Sent	**1/0**	**1/1**	**2/1**	**2/2**	**3/2**	**3/3**	**4/3**	**4/4**	**5/4**	**5/5**
100	N/A	0.001	0.001	0.003	0.004	0.006	0.007	0.010	0.010	0.011
1000	N/A	0.003	0.003	0.006	0.008	0.009	0.009	0.015	0.015	0.015
10000	N/A	0.018	0.017	0.039	0.034	0.051	0.051	0.070	0.074	0.086
100000	N/A	0.139	0.141	0.277	0.280	0.417	0.419	0.560	0.573	0.705
1000000	N/A	1.365	1.385	2.730	2.740	4.095	4.117	5.464	5.496	6.834



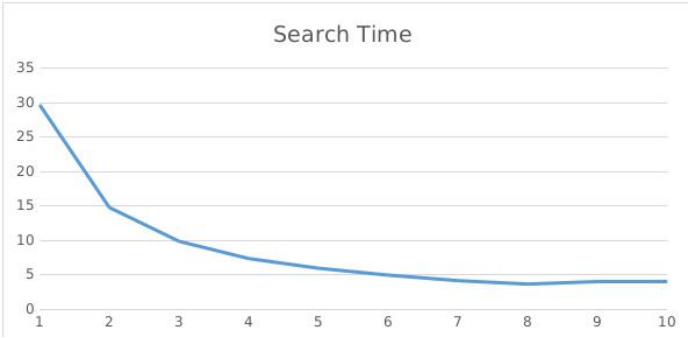
As expected, the amount of time it took to relay a message took longer with more processors. There is no result for 1/0 processors because there is no other processor to communicate with processor 0.

5.3 Search MPI

Search MPI is a function which utilizes parallel programming to find a value J which satisfies the condition F(J) equals some value C. It works by searching integers between two endpoint values, A and B, and evaluating each integer on a function F. Based on the number of processors the function has access to it will divide up the "search" work accordingly. We should expect to see the time it takes for the program to run to decrease as the number of processors increases.

Number of Processes per Pi

Range	**1/0**	**1/1**	**2/1**	**2/2**	**3/2**	**3/3**	**4/3**	**4/4**	**5/4**	**5/5**
1	3.0	1.5	1.0	75.0	0.64	0.51	0.46	0.38	0.39	0.41
$1 \cdot 10^8$	29.7	14.8	9.9	7.4	6.0	5.0	4.2	3.7	4.0	4.0
$1 \cdot 10^9$	297.0	148.0	99.0	74.0	60.0	50.0	42.0	37.0	43.0	37.0



Looking at the table above we can see the time it takes for the function to execute decay as more processors are employed. The second row displays a time of 14.8 when two are active and a time of 4 when 10 are active. This data output is what we expected and agrees with our assumption about the relationship between processors and time to execute.

6 Conclusion

After a series of tests, it can be concluded that there are benefits as well as limitations when implementing parallel processing on Raspberry Pis. Some of these limitations are simply due to the communication overhead that is inherent to many parallel processing structures. For certain problems, there is a significant gain in performance however the best cases for these are divisible problems that are not interdependent on results of other nodes. Cases that depend on computation results from other nodes in a parallelized cluster experience less of a gain in performance time. This method of parallel processing is ideally suited for Monte Carlo simulations.

7 Refrences

B. Peguero. (2014). *MpichCluster in Ubuntu* [Online].
Available https://help.ubuntu.com/community/MpichCluster
J. Burkardt. (2011). *MPI C Examples* [Online].
Available http://people.sc.fsu.edu/ jburkardt/c_src/mpi/mpi.html
MPICH. (2015). *MPICH Guide* [Online].
Available http://www.mpich.org/documentation/guides/
J. Fitzpatrick. (2013). *The HTG Guide to Getting Started with Raspberry Pi* [Online]. Available http://www.howtogeek.com/138281/the-htg-guide-to-getting-started-with-raspberry-pi/all/