Building a Parallel Processing Cluster Using Raspberry Pi 2

Moez Janmohammad

Introduction

The Raspberry Pi 2 is a small, credit card sized, single board computer capable of running Linux and other lightweight operating systems compatible with ARM processors. The devices cost $35 and come with 4 USB 2.0 ports, a 10/100 Ethernet port, GPIO headers, and HDMI output. The entire device can be powered through a soldered micro USB port or direct 5 V lines to the GPIO headers. Raspberry Pi has quickly become one of the most utilized chipsets for embedded Linux projects due to their small size, power efficiency, and flexibility. The Raspberry Pi platform comes in a variety of sizes and configurations to ensure its usability for all kinds of projects.

A supercomputing cluster is typically a collection of identical computing hardware networked together and running any kind of parallel processing software. The software allows each unit of hardware to share data and computational power. Typically, each node is an extremely powerful desktop computer that uses a 64 bit 8 core processor and 16 GB of Random Access Memory (RAM). Nodes can carry their own storage space but storage can also be networked for access by all nodes. Architecture is typically a master-slave design, where one node is designated as the head node and delegates tasks to the rest of the cluster. Most clusters use a system called Message Passing Interface (MPI) to carry these instructions and calculation results and split tasks. The system uses networking to allow programmers to create parallel programs to split tasks and run those tasks on multiple nodes simultaneously.

The Parallel Pi cluster was initially formulated as a part of a personal improvement plan where I would teach myself a new skill each month. February 2016 was learning parallel processing, and while I had access to SMU's multimillion dollar cluster, ManeFrame, I experienced some setbacks because I had user only access to the system. I couldn't install my own software, I had no control over hardware, and my code was at the mercy of the CONDOR task scheduler.

On a grander scale, there are many schools and users that contract time on private supercomputers when their needs can be met by much smaller clusters. My goal was to create standardized designs and software packages for these users to be able to build their own clusters and scale them to their needs.

Raspberry Pi devices also have the very unique feature that almost all low-level hardware is user accessible, which allows me to test embedded systems by using the GPIO headers. By building my own cluster, I would have direct control over the hardware and software.

Looking further into the future, I knew that I would be leaving SMU soon, and my access to ManeFrame would be cut off soon after graduation. I began to wonder how difficult it would be to build my own cluster using inexpensive hardware.

This document will serve as a write up of the project, and will contain all of the major design decisions made during the process of the build. For questions or clarification, please contact me at mjanmohammad@SMU.edu.

The Build

Hardware

I tasked myself to use only hardware that is easily accessible, which means I could not fabricate any parts or special order anything. I set this constraint to make the build as easy as possible for someone with little to no experience building clusters. The easiest way to order parts for clusters is online, and since Amazon has a ubiquitous presence in academia and home-brew projects, I used it as my sole supplier for parts.

Component	Description	Quantity	Total Cost
Raspberry Pi 2 Model B+	Computing hardware	12	$420
SanDisk 8GB MicroSD	Storage and OS	12	$108
1 foot MicroUSB Cables	Power	12	$21
12 Port Powered USB Hub	Power	1	$25
OSOYOO TFT Touchcreen	Display	1	$18
Cisco Gigabit Switch	Networking	1	$70
CAT6 UTP Ethernet Cable	Networking	250 ft	$60
RJ45 cable heads	Networking	100pc	$12
RJ45 Cable Crimper	Networking	1	$20
RJ45 Cable Tester	Networking	1	$10
Thunderbolt RJ45 Adapter	Networking	1	$30
M3x25 PCB Standoffs	Rack Design	25	$9
Acrylic Casing	Rack Design	1	$23
		Total Cost	$826

The total cost of the build came out to just over $946 after taxes and shipping.

A design question asked frequently is why I opted to use the micro USB for power rather than the GPIO headers. There are a combination of reasons, the GPIO pins don't have fuses so any sort of power surge could potentially destroy the cluster. Almost all USB ports contain fuses to ensure clean power delivery. Additionally, using the GPIO headers would require a specialized power unit which wasn't available on Amazon, or modifying a PC power supply, a daunting task if the user hasn't done it before.

I also opted for bulk uncut cable to make my own custom length networking cables. This was an effort to make the cabling look neater and inflated the cost of the cluster. A bulk package of 1 foot cables could easily be substituted, which would also reduce the cost of the project since the RJ45 heads, crimper, and tester would not be needed. For applications where the cluster will be scaled up in the future, selecting bulk cable in different colors could help to improve aesthetics as well as organizing stacks of nodes by cable color.

I also opted to use Raspberry Pi boards rather than the Parallela boards which are designed specifically for parallel processing. This is mostly due to cost ($35 vs $99), but also because the documentation for linking the Parallela boards to run in parallel with other boards is either nonexistent or difficult to understand.

Software

There are several suitable builds of Linux for Raspberry Pi that will allow me to accomplish parallel processing, and since most builds of Linux are open source, I can modify any of them to work perfectly with our hardware.

The primary linux distribution for Raspberry Pi is Raspbian, a very simple and easy to set up distribution that comes preloaded on all Raspberry Pi units. It is a custom distribution optimized to run on the Raspberry Pi, which makes it very easy to get started in Raspberry Pi and Linux in general. The downside to Raspbian is its bloat. It includes a large number of libraries and extra applications to support its ease of use, which greatly impact performance and boot time. A cold boot to Raspbian can take upwards of 75 seconds.

On the other side of the spectrum, Arch Linux is extremely minimalistic. It boots in under 10 seconds and provides an extremely barebones environment, using only the command line interface for set up and general use. It doesn't even include a C compiler! This is ideal because you start with the cleanest set up and install only what you need for your purposes.

In the build, I used Arch Linux, but all libraries are compatible with Raspbian so the two can be used interchangeably. I chose Arch Linux because it uses a hardware floating point for calculations which enables it to handle decimal points more efficiently. Raspbian uses a soft-float, which uses software to simulate a floating point at the cost of performance.

Firstly, I set up the networking using the hard coded networking tools. I created a network with the head node at 192.168.1.1 and the subsequent nodes at IPs from 192.168.1.2 to 192.168.1.12. The gateway was set up so that the head node would handle all traffic. The cluster wouldn't need internet access except to phone home to a dynamic DNS server and update its own IP address for remote management.

Because all of the Raspberry Pis except the head node would have the exact same settings, it is very easily expandable to a much larger cluster since the Raspberry Pi can be cloned to an identical device. I used a Mac program called Apple Pi-Baker to clone each unit and assign it a new IP address. Static IPs were chosen because it leaves the MPI configurations static. Using DHCP would have required two additional steps. First, a DHCP server would have to be created on the head node to assign IP addresses. While this is a trivial task, it would a leech for processor cycles to assign the IPs. DHCP also requires a broadcast message to be sent at intervals to renew the IP address lease which would clog the network. The second additional task would have been to write a script for MPI which scans for the IP addresses and adds them to the MPI configuration file to use the nodes in the calculation.

My next task was to be able to access the cluster from anywhere. Usually this would mean obtaining a static IP from your service provider and assigning that IP to the cluster but since this is such a small cluster I thought it would be cool to be able to pick it up and move it and still be able to log in without configuring the network all over again. For this, I used a dynamic domain name service. Dynamic DNS lets you access a device from anywhere. The device is configured to contact the Dynamic DNS and notify the server of its new location. The Dynamic DNS then reroutes all traffic to a predetermined address to that IP. For example, I can access the cluster from anywhere using an SSH login to moez@parallelpi.(hostname).com. I could carry the cluster around, plug the network port into any standard ethernet jack, and be able to access the cluster within minutes, without physically connecting a cable from my computer to the cluster. I could also leave the cluster at home and continue working on the configurations while I was in the library at SMU or out of town.

Performance

The biggest measure of performance for any parallel processing cluster or supercomputer is its speed. To test its computational power, I used the Linpack Benchmark (https://www.top500.org/project/linpack/). This is the same stress test that is used to calculate the rankings of the Top500 supercomputers. The benchmark uses a very complex matrix to solve a system of linear equations. It is a very specific test that doesn't measure overall performance but it is the currently accepted benchmark for computational speed for floating point operations. The Parallel Pi cluster clocked in at 5.76 GFLOPS and a max power draw of 63 Watts. At idle, the system drew 15 Watts of power (which I suspect was mostly the switch since it has no low power mode). That score was attained with each of the nodes overclocked to 1.1 GHz and the almost all of the RAM allocated to the CPU. On the Raspberry Pi platform, 1 GB of RAM is shared equally among the CPU and GPU. In the pi-config menu, a user can manually change the allocation, with 48MB being the minimum required for the GPU for the system to boot safely. Since none of the nodes are generating any graphics, the RAM allocated for the GPU was reduced to the minimum.

After sniffing the packets transmitted on the cluster, I realized there were two bottlenecks that prevented optimal performance:

The processors in the Raspberry Pi can keep up with most mathematical calculations, but transmitting the data is limited to 100 MB per second speeds on the soldered ethernet port. A USB to gigabit ethernet adapter is available but with a cost of $12, outfitting the whole cluster would cost $144 and add a lot of bulk.

Adding gigabit adapters would be moot because the read/write to the memory card is limited to the speed of the memory card. Even using the fastest Class 10 UHS-1 cards had a max speed of 35 MB per second, which is by far the largest bottleneck of the cluster.

Heat isn't a huge issue for a 12 node cluster. The passive cooling on the Raspberry Pi is more than enough to prevent damage to the cluster, but for larger clusters a small computer fan may be a good investment.

Improvements

While my work on the cluster is complete, if I had the opportunity to rebuild the whole system from the beginning, I would make a few changes. First I would use the new Raspberry Pi 3 boards. The new Raspberry Pi 3 has a 1.2 GHz 64 bit quad core ARMv8 CPU. Using a 64 bit library almost doubles the speed of the board and would have huge performance gains for Parallel Pi. The new Pi 3 board has the same cost as a Pi 2, but are a little harder to get since demand for the new boards is so high. This comes at a cost of power usage increase. The Raspberry Pi 2 at idle uses 1Watt of power, while the new Pi 3 board doubles that at 2 Watts. At full load, I expect the Pi 3 boards to use double the power as well, bumping the cluster to over 120 Watts. Comparing this to a full sized cluster, the power usage and resources are extremely lean. ManeFrame has a full support staff and a dedicated building to house it, at a much larger cost than Parallel Pi.

The next improvement would be to change the way updating libraries on the nodes works. Currently, if any library (MPI, gcc, etc.) needs to be updated, I have to manually update it on each of the nodes. This is time consuming and inefficient. A workaround could be to have a script that runs at boot on each node that clones the /bin and /usr directories from the head node to each of the other nodes. Since all of the libraries used in the cluster are contained within those two folders, any update on the head node would be propagated to each of the other nodes. To make it even more efficient, an update checking script could be written to see if the version of the libraries on the head node is different from the ones on the worker nodes and only copy the folders if the versions are different.

In the materials section of this paper, I included the cost of an OSOOYOO TFT Touchscreen that was never mentioned. My plan for the screen was to have a live view of the current usage of the cluster. It would display CPU usage per node, power draw, and network activity to the head node. It is possible to pull all of this data and display it, I just never got around to it.

Conclusion

For under $1000, a 12 node Raspberry Pi cluster works pretty well! Its a great way to learn the ins and outs of parallel processing. With enough scaling and a cooling solution in place, it could even be used as the primary cluster for low level supercomputing needs. In fact, I used it almost exclusively my entire last semester at SMU to test parallel software before loading it to ManeFrame for the heavy lifting.

I would like to thank Susan Kress, SMU Engaged Learning, and SMU Big iDeas for giving me the funding to build this cluster, Robert Kalescky for guiding me through the process of getting the whole cluster working, Dr. Randall Scalise of the SMU Physics Department for inspiring the idea by asking me to parallelize a Monte Carlo Robin Hood simulation, and Dr. Stephen Sekula for his motivation while building the cluster. This project would not have been possible without your support. Special thanks to TEDxSMU for giving me the opportunity to show off my work at the March 2016 audition.