Parallel Processing Using a Raspberry Pi Bramble

Kyle Bond
7/22/2013

Table of Contents

1. Introduction

High Performance Computing (HPC) clusters provide universities, corporations and governments the ability to create immersive simulations and compute complex problems. These HPC clusters provide a powerful computing ability, but are typically out of reach for most people, due to cost and complexity. In order to facilitate learning about the benefits of HPC, similar tasks can be performed on smaller, cheaper computers, although the speed of the simulation/computation will be significantly less. This allows users to learn about HPC clusters without the need for expensive equipment, as well as allowing users with access to HPC clusters the ability to test code before reserving time on their cluster.

The Raspberry Pi (RPi) provides an excellent substitute for the nodes used in HPC – the small form factor and inexpensive price (approximately $35) allow home enthusiasts to purchase several of these and place them on a network to simulate HPC clustering. A cluster of RPis is colloquially referred to as a "bramble". However, connecting multiple RPis together via Ethernet does not make them functional – software is required to coordinate efforts between the RPi nodes. The Message Passing Interface allows nodes within a bramble to communicate by passing messages across Ethernet. Message Passing Interface – Chameleon 2 (MPICH2) provides a framework to develop applications in C or Fortran 77 for distributed computing purposes. MPICH2 can also be extended to support Python, the language of choice used in this project.

This project's aim is to create a bramble with four RPis and analyze the reference human genome for patterns.

2. Distributed System Architecture

2.1. Raspberry Pi

The Raspberry Pi Model B, comes equipped with a System-on-a-chip (SoC) Broadcom BCM2835 700MHz ARM1176 processor, with VideoCoreIV GPU and 512 MB SDRAM. Persistent storage is handled by removable SD card. The RPi also contains two USB ports, a composite RCA port, HDMI port, 3.5mm audio jack, an Ethernet port, and is powered with a micro-USB port, requiring approximately 700 mA of power (eLinux.org, 2013). Operating systems can be preloaded on to the removable SD card, allowing users to create a single image and distribute the image to multiple SD cards through the use of a desktop computer. The master node uses a 16 GB SD card to handle the installation of a graphical user interface, while slave nodes use 8 GB SD cards.

2.2. Operating System

For this project, the RPis use the Arch Linux ARM (ALARM) operating system. ALARM provides low overhead and full customizability – two important needs for this project (eLinux.org, 2013). ALARM begins as a barebones operating system, containing only the packages necessary to get started. Users can create a master copy of the disk image by copying the

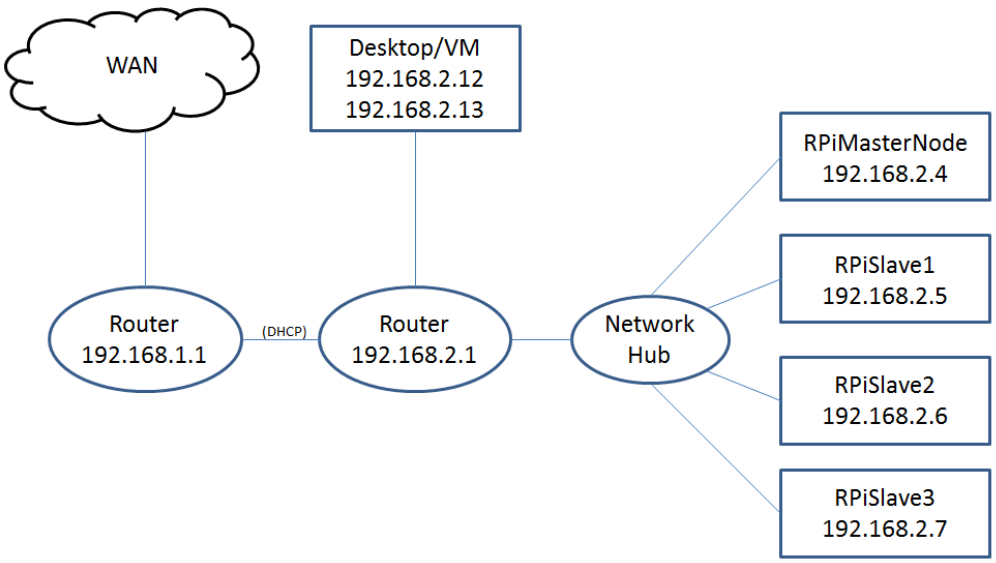
ALARM image on to an SD card, installing the necessary packages using the RPi or a desktop, then copy the SD card contents to another image file for distribution to other SD cards.

2.3. Network Architecture

The network provides the backbone for the RPi bramble. Communication between the different processes running between two nodes in the bramble is a key component of distributed computing. In this bramble, the master node receives communications from the three slave nodes via the network hub and router which provides coordination and status update information.

Although a keyboard and a mouse attached to the bramble through a USB hub, a virtual machine running on a desktop computer most often controls the bramble, by using SSH to create a connection to the master node. RSA keys were shared amongst the nodes for ease of access. Running commands from the virtual machine is the preferred method of operation.

In order to facilitate the installation of packages, the bramble is also connected to the internet through a second router.



3. Software Architecture

3.1. ALARM Packages

ALARM requires the installation of several packages before loading MPICH2 or the Python extension for MPICH2, Mpi4Py. These packages include:

- gcc
- gcc-fortran-multilib

- python2
- make
- python2-pip

These packages provide the bare necessities required to operate the MPICH2 and Mpi4Py software. Another useful package is numpy, which allows Mpi4Py to convert Python data types to C data types.

3.2. Message Passing Interface – Chameleon 2 (MPICH2)

MPICH2 provides a framework for message-passing between nodes in a cluster. This allows master nodes to send instructions for execution and data to slave nodes, as well as allowing slave nodes to send master nodes results. Although there are several sets of instructions for installing MPICH2, Phil Leonard's instructions provide clear, concise steps to follow, with a few exceptions. One notable instruction missing: all nodes need the same directory structure for the MPICH2/Mpi4Py applications and the Python scripts. Some of these instructions are specifically for running Mpi4Py and therefore Phil Leonard's instructions do not include these steps.

Here are the basic steps (based off of option 3 in the instructions (Leonard, 2012)):

1. Retrieve and unpack the tar ball from Argonne National Laboratory.
2. Navigate to the folder where the tar ball was unpacked and run the following command to enable shared libraries and build the software:
    a. `./configure CFLAGS=-fPIC --enable-shared`
    b. `sudo make`
    c. `sudo make install`

3.3. Mpi4Py

Once MPICH2 is installed, Mpi4Py installation requires a single step. Running the command `pip2 install mpi4py` as root or sudo installs the application. The pip2 package manager provides simple installation and management services for Python-based software - Python versions 2.4 – 2.xx require the use of pip2, while Python versions 3.1+ require pip.

3.4. Post Installation Testing

After installing MPICH2 and Mpi4Py, performing application testing to ensure proper installation and operation should be top priority. MPICH2 provides a sample testing application to determine that the installation is working properly. The CPI application, written in C, determines pi to 16 decimal places using the Monte Carlo method of estimation (Cox, 2013). Detailed instructions for running the CPI application can be found at the University of Southampton Raspberry Pi Website.

Mpi4Py did not come with a stock testing application; however, Jeremy Bejarano's website (no longer in operation) provides an application useful for testing. This particular test

calculates the area under a curve using the trapezoidal rule (the code for this can be found in the source code ZIP file, in path /MPI4PyCode/PrimeGenerator/parallelTrapezoidal.py). This application executes on the following command:

```
mpiexec –machinefile /home/rpi/mpi_testing/machinefile –n 4
python2 parallelTrapezoidal.py 0.0 2.0 150
```

This calculates the approximate integral of $f(x) = x^2$ from 0.0 to 2.0 using 150 trapezoids, the answer being 2.666725.

A simple breakdown of the command:

- **mpiexec:** the executable for MPICH2
- **-machinefile <pathname>:** the path to the machine file, a list of computers that can execute the application
- **-n <int>:** the number of processes MPICH2 should spawn to execute the application
- **python2 parallelTrapezoidal.py:** the python executable and script name, followed by parameters:
  - **0.0:** lower bound for the trapezoidal rule
  - **2.0:** upper bound for the trapezoidal rule
  - **150:** number of trapezoids to fit between the lower and upper bound

4. Prime Generator

Prime number generation is the first test written and applied to the bramble for this project. Although the Sieve of Eratosthenes method of prime number generation (Caldwell) does not lend itself to division of efforts that a bramble could provide, the algorithm is simple to implement and allows the nodes to pass messages back and forth to be displayed in the terminal.

5. DNA Analysis

The DNA Analysis application offered an easy test of both the computing power of the RPis and capabilities of the MPICH2 and Mpi4Py systems. The application accepts four parameters: input file name, base pair size, output log file name, and CSV output file name. The application then loads the input file (which is in a FASTA format) and strips the header information and the extraneous material used in the file format. This leaves only the remaining sequence data. The application then

begins searching for duplicate sets of DNA with the same base pair size. For example, searching with a base pair size of 10 would yield the following results:

ATATTACTGCTGAAAAACAGATATTACTGCCAGATATCAGATATATATT

The work is divided up between the RPis using the ranking system provided by MPICH2. Each node receives a rank, and then divides up the data to work by dividing the work based on the number of

nodes. The starting index is calculated by multiplying the rank times the data chunk size. For example, with a bramble with 4 RPis working on a DNA Sequence of 100 characters, the node with rank 0 would start at index 0 (0 * 25) and end at index 25 (0 * 25 + 25). The index shifts by one each iteration and the search starts over again. J. Bejarano's tutorial site for Mpi4Py contained information that was instrumental in developing this algorithm. Unfortunately, the site is no longer available.

Each node will traverse the entire DNA sequence for each search to ensure that every match is found. This process can be slow – each RPi takes approximately four minutes to traverse the smallest chromosome for a single search at 15 base pairs. A bramble, such as the one being used for this project, with 4 RPis would take the same number of minutes to traverse the chromosome as there are number of characters. To put this in perspective, chromosome 21 contains the smallest amount of DNA when excluding the buffer – a sequence that is 35,105,520 characters in length would take approximate 66.79 years to complete analysis.

Although the time-complexity of the algorithm being employed by the application is $O(n^2)$, running the alternative algorithm solutions are less appealing due to hardware constraints imposed by the Raspberry Pi. A map/reduce algorithm would start to run out of space in the heap due to the memory capacity of the RPi, which is the limiting factor in most algorithms. Implementing these algorithms to use persistent storage instead of heap space may be possible, but the read/write speed for flash memory is significantly less than read/write for the SoC memory.

6. Conclusion

Although the Raspberry Pi may be lacking in processing power, they serve as an excellent replacement for nodes in a high performance computing cluster. The concepts of distributed computing remain the same, regardless of the processing speed or amount of memory of the nodes inside the cluster.

The low-end hardware used for the RPi allows the Raspberry Pi Foundation to produce the RPis for between $25 and $35 (Raspberry Pi Foundation), depending on the model purchased. This hardware makes it difficult to run distributed applications for two reasons:

1. Single-core processor: The RPi has a 700 MHz, single core CPU. The CPU has to respond to all interrupts, and although there may not be many interrupts, this still slows any application that is running.
2. 512 MB memory: The memory available limits what types of algorithms can run. For the DNA Analysis application, every available megabyte counts. Python's garbage collection does not seem efficient enough to clear out strings as fast as the application can make them, sometimes causing issues with memory management.

Overall, the Raspberry Pis do serve their purpose – a cheap, functional computing platform that can run as a fully-fledged computer.

Works Cited

Caldwell, C. K. (n.d.). *Sieve of Eratosthenes.* Retrieved from Prime Pages: http://primes.utm.edu/glossary/xpage/sieveoferatosthenes.html

Cox, S. (2013, January 9). *Steps to make Raspberry Pi Supercomputer.* Retrieved from University of Southampton: http://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton.htm

eLinux.org. (2013, June 24). *ArchLinux Install Guide.* Retrieved July 20, 2013, from eLinux.org: http://elinux.org/ArchLinux_Install_Guide

eLinux.org. (2013, July 6). *RPi Hardware.* Retrieved July 20, 2013, from eLinux.org: http://elinux.org/RPi_Hardware

Leonard, P. (2012, 06 18). *Parallel Processing on the Pi (Bramble).* Retrieved from http://westcoastlabs.blogspot.co.uk/2012/06/parallel-processing-on-pi-bramble.html

Raspberry Pi Foundation. (n.d.). *FAQs.* Retrieved from raspberrypi.org: http://www.raspberrypi.org/faqs

Works Referenced

Bejarano, J. (n.d.). *MPI with Python.* Retrieved 07 05, 2013, from http://jeremybejarano.zzl.org/MPIwithPython/overview.html

Dalcin, L. (2012, January 20). *MPI for Python, Release 1.3.* Retrieved 07 21, 2013, from SciPy - Mpi4Py: http://mpi4py.scipy.org/docs/mpi4py.pdf

Fitzpatrick, J. (2013). *How to Configure Your Raspberry Pi for Remote Shell, Desktop, and File Transfer.* Retrieved from How-To Geek: http://www.howtogeek.com/141157/how-to-configure-your-raspberry-pi-for-remote-shell-desktop-and-file-transfer/

Kiepert, J. (2013, 05 22). *RPi Cluster: Creating a Raspberry Pi-Based Beowulf Cluster.* Retrieved from Boise State University ECE: http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf