Constructing a powerful and flexible computing cluster using Raspberry Pi computers

Camilo Correa Restrepo, Universidad EAFIT

February 18, 2016

1 Introduction

Not too long ago, creating a cluster of computers, using affordable and commercially available hardware, that was suitably portable for education and durable enough to withstand the harshest conditions a classroom could offer was a task not very well suited for the faint of heart, specially because of the fact that wrestling for hours on end with unreliable software would probably wreak havoc on one's mind and hardware. However, as time has gone on, and with the advent of small, reasonably fast and extremely cheap computers, this idea no longer is a dream of many in the academic community, but rather, an interesting reality that's ripe for exploration. As such, we will now embark on the journey of constructing our very own cluster, with the quite sincerely amazing Raspberry Pi. With that in mind, the goal of this project is, evidently, to create a small and very easy to dismantle cluster that is very easy to set up and get up and running for educational purposes (for after all, you will not be able to do very complex computations on this hardware). This cluster will be very easily customizable, easy to manage and maintain, and will behave exactly like a real world high performance computer would on a much smaller scale.

Despite the simple nature of this project, I will, however, assume you have a basic knowledge and experience using UNIX systems. We will set out to achieve a very simple goal to channel our efforts, as, without it, we would go nowhere. We will work towards making this system as robust and as well set up as we can, in order to prepare it to run one very special application, John the Ripper, that bridges the gap between the world of High Performance Computing and the world of Cyber Security. Though this might not even make sense, it is imperative that you understand, going into this, that the true purpose of this is to enable whoever may read it to acquire and absorb a myriad of skills that will allow them to continue enjoying and exploring this daring and fantastic new world with effectiveness and preparedness.

Contents

2 What is a Raspberry Pi and why are we using it?

It is absolutely impossible to talk about low cost and low consumption devices without mentioning a raspberry pi, it is an absolutely phenomenal device, that carries has propelled forth a true paradigm shift in the way we look at embedded devices, and it is has also proven itself to be a truly spectacular way to teach the ways of computing to others. Thus, let me first begin by presenting a quote from the Raspberry Pi Foundation, that presents what a raspberry pi is better than I ever could:

> "The Raspberry Pi is a credit-card sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does, like spreadsheets, word-processing, browsing the internet and games. It also plays high-definition video. We want to see it being used by kids all over the world to learn programming." [1]

As you can now, hopefully, imagine, these little devices are a true marvel of modern engineering, as they pack within a truly great "bang for your buck" in terms of capabilities, and are an absolutely perfect testbed for large engineering solutions.

3 What you will need

In order to construct your very own cluster, you will need, at the very least, the following components:

- 2 or more Raspberry Pi computers
- A micro-USB cable for every Raspberry Pi
- An Internet router with an integrated switch or a network switch
- Enough Cat-5 cables to connect all computers to the network
- SD cards for every Raspberry Pi of at least 4GB of space (ideally 8GB or larger).
- A USB flash drive of at least 4GB of space (ideally 8GB or larger).
- A USB keyboard
- An HDMI-capable screen
- A computer with an SD card reader.

[1] https://www.raspberrypi.org/help/faqs/

And additionally, while not strictly necessary, these additional components will make your life a lot easier:

- An Internet connection (extremely recommended)
- An additional computer with ssh capabilities.
- Spare SD cards (should one fail or get damaged)
- Spare flash drives (to expand the internal storage of the pi's)
- Cases for your Raspberry Pis, as they might get damaged when you move them around or connect them to power sources or network interfaces.
- USB wall adapters, so you can hook up your Raspberry Pis to your wall outlets.

Please do keep in mind that these are the bare essentials to begin the construction of the system; thus, to scale this system, all you'll need to do is begin expanding the quantities of the materials involved, and performing all setup operations on every new component.

Additionally, if you wish to create a larger array of pis, do keep in mind that their combined heat output will no longer be trivial, and, thusly, it would be in your best interest to begin thinking of a spatial distribution that would allow your array to receive an adequate amount of airflow.

4 Initial tasks

Assuming you have procured all of the necessary components, let us, then, focus our efforts on the first tasks that must be completed before we can begin to use our shiny new cluster.

4.1 Choosing your operating System

The first and most important step of the process, without a doubt, is the operating system you will use on your devices, simply due to the fact that while they will all be based on the Linux kernel, they all have large and small differences that will either make your life easy, or hell on earth, so you must be quite sure of the OS you are going to marry. In my humble opinion, the best choice for a raspberry pi cluster is Raspbian, the Raspberry Pi foundation's operating system, which is a fork of the much revered and loved Debian GNU/Linux operating system. This however, does not in any way demerit all the other available OSs, as they all have their pros and cons, and it is absolutely true that you will run into a couple of headaches by using Raspbian. However, it does, as of the time this is being written, offer the greatest amount of community support and the greatest amount of pre-built packages, which will make getting the system

up and running far easier[2].

If, despite reading everything above, you still feel the desire to work with another OS, I would personally recommend Arch Linux as the best alternative, but do keep in mind that you will need to do a lot of compiling on your own, and, it goes without saying, this definitely isn't for the faint of heart.

4.2 Setting up your master SD card

Once you have chosen a suitable operating system for your device, the next step is to begin preparing the SD card for your master node[3]. First, you should download a copy of the disk image of the operating system you wish to run. For example, for Raspbian, head to the Raspberry Pi foundation's website and find the .img for Raspbian. Once you are done downloading, all you need to do then is do a simple *dd* command to copy the disk image onto your shiny new SD card.

I must however issue a word of caution to anyone using dd for the first time, or for anyone that is inexperienced: *dd* is, like most core UNIX utilities, extremely useful and extremely dangerous, as such, I must emphasize that you must exercise extreme caution using the command, as any mistake in the output target could result in the loss of data.

4.3 Physically setting up your cluster

Now that the SD card is ready, it is now time to begin preparing all the physical components you will be using. Depending on how you've chosen to work, and how many nodes you will have in your network, it might be in your best interest to physically assemble your cluster before doing any work on it. If, however, you plan to operate a simple and small cluster, then you may want to avoid setting too many things up beforehand, and simply getting straight to work on the cluster.

No matter what you've chosen, you will absolutely need, for your first setup, a network connection, a monitor, and a keyboard. As soon as you have hooked all of these up go ahead and insert your SD card into it's slot and connect your power cord.

Once again I must issue a word of warning for those working through these steps. Raspberry Pis, in theory are compatible with

[2] A word about Raspbian must be said, before going forward: quite recently Raspbian switched from being based on Debian Wheezy to Debian Jessie, which, while fantastic for early adopters and those that crave the latest packages, is not ideal for what we'll be doing for our build, simply due to the fact that it has some features that make it not desirable for us, for instance, not all packages have been ported over, and probably quite a few are still quite buggy, additionaly, its kernel version is 4.1, which is known to have a couple of stability issues, specially on hardware that isn't x86.

[3] The one that'll be in charge of controlling all the other nodes

a very large amount of peripheral devices, but, that does not guarantee that the particular ones you will be using will be supported, especially if, for example you want to use an HDMI to VGA adapter, or a wireless keyboard or a WiFi dongle. Nonetheless, there are very many internet resources ready to point you in the right direction, and most, if not all of what you need is only a google search away.

5 Setting up your master node

If everything up to this point went well, then you will probably be staring at a prompt asking for your user and password. If you went with Raspbian, then you need only enter *pi* as your user and *raspberry* as your password. You will now see that you are logged in to your primary raspberry pi, and you are now absolutely ready to begin the setup of your cluster.

5.1 Getting the latest versions of the default packages

Once you've logged in, it is now time to begin the very easy process of obtaining all of the newest and shiniest packages for your system, as it is always a good idea to have the latest and most updated version of the base system utilities. To those of you familiar with the Aptitude package manager, it will be a true breeze to do this process. For those of you that aren't, all you'll need to do is input the following two lines:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

5.2 Running the configuration script

Next, you will need to enable the ssh server on your raspberry pi, so that you can connect to it remotely (should you desire to connect from another more comfortable computer). To enter the configuration script, enter the command:

```
$ sudo raspi-config
```

Using your arrow keys, enter the advanced options and navigate onto the ssh section, then enable the service and exit the script. Once you are done, your raspberry pi will reboot and we will be ready to continue.

5.3 Mounting your USB drive

Every linux user knows that mounting a usb drive is a rather trivial task, however, for our needs we will need to have this drive automatically mounted as soon as our system begins, thus, we will need to edit our */etc/fstab* file so that it is automatically mounted. As a convention, it would be a good idea to mount the drive to a folder such as */share*.

5.4 Installing all the needed utilities

Now that you having enabled ssh, it is time to download all the utilities that we will require in order to mount our cluster. Input to the following command[4] to download everything we'll need:

```
$ sudo apt-get
nfs-kernel-serv
tree htop libss
```

Go ahead and make yourself a cup of tea while you wait, as this will probably take a while to download.

5.5 Preparing your NFS

If everything is working well, and your flash drive is working correctly, then all you'll need to do to make it available to other computers is to create a rule for your mounted folder on your */etc/exports* file[5]. Make sure that the correct permissions are set for all user, and that they have access to the full subtree of that drive. Next reboot your device and pay close attention to the boot-up messages, since somewhere along the way, you'll first see how your drive is mounted and next, you'll see how the NFS services restart and how they broadcast your file system. Make sure that this worked correctly before you proceed as it is absolutely vital to most of the steps we will do going forward.

Next, you'll need to set up a couple folders on your shared directory so that we can, going forward, fill it up with interesting and powerful software and other things. So, go ahead and do the following:

```
$ sudo
```

5.6 Setting up your environment modules

The time has finally come to begin inserting into our cluster some of the best and most powerful software out there. First up to bat is Environment Modules, a simple little program that eats up TCL scripts and allows you to set and unset your working environment just how you like it, so that you can use the exact version you want of a program. In fact, you can modify every environment variable in your system with it, and you can also quickly and safely undo your

[4] As you will see in the following command and in several other commands, a backslash (\) will indicate that the command continues but just on the next line, basically, just imagine that you paste whatever begins on the next line exactly on the spot that the backslash is.

[5] Some of you might consider it unusual that I more or less skipped this part. However, since this is supposed to be an eyeopener to this wonderful world, I have decided that it is only fair that both mounting the USB drive and the NFS server should be a an exercise for the reader, in order to test his abilities to both problem solve, and to make things work. This is easily justified by the fact that things can and will go wrong no matter how faithfully you follow the guide, so, please do understand that this is for your own good.

changes.

5.6.1 Overview of what will be done

In order to get our modules up, we will need to compile them, since, as of this writing, it is not available in any of the public repositories for Raspberry Pi, and, even if it were, it is quite a good exercise to try and get this program up and running yourself. Additionally, compiling it ourselves allows us to control, with the utmost precision, how we want it to behave. Specifically, we don't want it to be installed in a default directory, but rather in our */share/apps* directory, so we only need to install it once and it will be forever available to all other nodes, and, furthermore, we will be able to configure it to look for its modulefiles (more on this later) where we want it to, and nowhere else.

Some of you will break out in a cold sweat just from the word "compile", but you'll quickly realize that as with most things in life, it'll only hurt the first time, and every time after that it'll only get better. So, with that in mind let us begin.

5.6.2 Obtaining the Software

Since we won't be using our trusty if a bit temperamental friend Aptitude, we'll need to nab ourselves the compressed package for Modules, which is easily obtainable from it's website using a simple *wget* command and a *tar* command in order to access all the contents of the downloaded file.

```
$ cd
$ wget
    /M
    $ t
```

5.6.3 Preparing to Compile

First enter our new directory:

```
$ c
```

Then read the *README* and *INSTALL* files, and, once you're done, read them once more just to be safe. Once you are ready, I would advise you to run the following command to see how far it gets, the rationale behind this is that, ultimately, every installation is different and all are missing different things[6], therefore, it would make sense for you to see what the program is expecting in order to be compiled (alternatively you may skip this step entirely and pick up near our next command):

[6]In general, base Linux installations come with TCL but not TCLX and, thusly, it makes sense to not look for it, and, in any case, it will not be required by any of the scripts we write.

If it fails, then it means it is unable to find something that it needs in your system, and, well, this is a prime opportunity to get acquainted with the day to day life of a System Administrator. As such, the onus is on you to fix and procure anything and everything you may be missing for your installation. If, however, it finishes correctly, stop and don't do anything, for we haven't correctly configured it and it would be a mistake to install it as it is. What we must do, now, is run our configure script so that it set all the variables correctly and so our compilation process is as painless and easy as humanly possible.

Our next command is a bit more involved but by no means anything out of the ordinary (especially if you did in fact read both documents):

Once it's done you will now be ready to begin compiling[7] modules.

5.6.4 Compiling Modules

There isn't a lot to say about the actual compilation process, as it involves a very simple command that is in no way difficult to use:

Once you enter the command, I'd advise a short walk or coffee break as it might take a short while. If it completes correctly the you are now ready to test your binary:

Now, if either the checking process, or the compilation process fails, it probably means you did something wrong and you'd be best served by returning to the beginning of this process and trying again. In theory, it shouldn't fail in any step of the way once the configure script passes, but nothing is absolutely assured to work.

5.6.5 Installing Modules

Having hurdled this enormous barrier, it is now time to finally install the products of our labor. In order to do this correctly I would advise you doing the following, as this will temporarily give you write access and permissions over

[7] The more observant among you will note that we will in fact be using the system's default compiler, and that it isn't (in general) the best idea. However, keep in mind that in order to correctly manage several gcc versions we will need modules installed and functioning, and, therefore, it makes absolute sense to simply set up modules with the base compiler. Additionally, we will not need anything more than a basic ANSI C compiler, and, thusly, our system's gcc will suffice. Performance-wise, the same applies, the program is so simple and small that it is a true fool's errand to pursue compilation of modules with any other compiler.

the folders where you will install the modules package and it will then return both the user and the group to root so that it can't be modified by accident.

5.6.6 Configuring modules

Assuming the installation process has gone smoothly, the next step is quite simple, but necessary in order to guarantee that modules will be available to each and every user of the cluster, no matter what machine they use. With this in mind, first head to the folder where you installed modules, which, in our case, would be */share/apps/modules/3.2.10/*.

Once there, move to the folder where the initialization scripts are located, and verify that they indeed are there.

There, a file for all supported shells is available, and ready to be installed. In order to finish the set up, all that needs to be done is run the following command.

Once that command completes, your modules installation should be up and ready to use. Simply reboot your machine and log back in. Once you've done so, run the following command:

If usage instructions appear, the your modules installation is complete and you are now ready to proceed with the installation of any other package you may desire.

5.7 Obtaining a new Compiler

Raspbian, based on Debian wheezy, comes by default with GCC version 4.6.3, which has been thoroughly tested and is quite stable, but, unfortunately, it lacks support for most of the newer features of the C++ programming language, and therefore, we will need to upgrade to a newer version in order to make full use of these features[8]. The version that has been chosen is version 4.8.5, which is quite stable and supports all the features we need for the software we will be installing in the future, additionally, it is very well supported and is far easier to install and use then newer gcc releases, that have been known to be somewhat problematic on embedded systems.

[8] Do keep in mind that if you are using the Debian Jessie release, you will most likely have a newer compiler that does not suffer this issue.

5.7.1 Overview of what will be done

While it is true that precompiled packages do exist in the wild and that they are completely usable, it is a worthwhile exercise to compile your own compiler for your system, mostly due to the fact that going forward, in order to have the newest version and the latest and greatest features, it is necessary to manually build it. Compiling our new compiler will be a somewhat more difficult task than compiling Modules, simply because of the fact that there are a great deal of options and nuanced steps to be aware of. However, despite all of that, in the end it will turn out to be a rather simple project, if all indeed does according to plan.
Do keep in mind that there are no real shortcuts to be taken in the compilation of gcc, but there several somewhat well defined steps to follow, which, in our case, will not suppose too great a challenge. The steps that we shall follow are:

- Download and install the GNU Multiple Precision Arithmetic Library (GMP) library, version 6.1.0
- Download and install the GNU MPFR library, version 3.1.3
- Download and install the GNU MPC library, version 1.0.3
- Download and install the GNU Compiler Collection (GCC), version 4.8.5

5.7.2 Obtaining the Software

We'll need to download the source from the official website, which, in it of itself is a quite simple task. To do this, all that needs to be done is to input the following commands:

Do make sure that you have successfully downloaded and extracted the correct compiler. If you so desire you may verify the integrity and validity of your download.

5.7.3 Preparing to Compile

As with any other software, compilation of gcc can be made as complex or as simple as is desired, though, usually, in order to make it work as well as can

be in a given system, tuning of its compilation parameters is not only recommended, but actually, necessary. With this in mind, several resources must be read, at least cursorily, in order to guarantee a smooth build process. Out of these resources, the one I would perhaps recommend reading through the most is, without a doubt, the GCC compiler's homepage, where, in every single corner, lies some useful gem of information.

Fundamentally, compilers are complex and very demanding programs, that are more then capable of using every single bit of their host operating system, and, as such, are absolutely non-trivial tests of a system's setup and capabilities. In addition, they are more than likely to be difficult to get working correctly on a system, thus necessitating a rather involved set of steps to get them working. It is thus with this that we must now focus our attention on actually beginning to compile our very own compiler. In order to set this process into motion we will need to compile and install several libraries so that we can even begin to build gcc on our system.

5.7.4 Compiling and installing GMP

The first of the aforementioned libraries is GMP (or GNU Multiple Precision Arithmetic Library). Fortunately for us, the build process for it is rather painless, and, of course, we in theory already have made ourselves with all the requirements to get it running. Go ahead and move the folder where it is located:

Now that you've made it here, you will need to configure the program so that we can compile it, thus, let us do exactly like we did with modules and let's make sure that our configure script is capable of finishing with no options:

If it successfully finishes, then that means we are well on our way and that things have been correctly setup. We shall now run the configuration script once again, but, this time around, with the options we will actually use:

We will be installing it following our convention, as it would otherwise not install it in the shared file system. In any case, it should, once again, finish without a problem. Now that you've configured it, we must build the library itself:

If it has finished correctly, and it appears to not have had any errors, then you are now ready to install it to our preferred location on our shared file system. To do so, enter the following commands:

As you can quite clearly see, this is very similar to what we did with modules.

5.7.5 Compiling and installing MPFR

Just as we had down before with GMP, we need to compile and install mpfr, which, fortunately for us, is done through pretty much the same process. Now, it is time to head into its folder:

And, just as we had done before, we need to configure it, however, this library requires that GMP be present and compiled, so it is, therefore, advisable, to not just run *configure* with nothing else added. In addition to that, we will include, at once where we will be installing it, in order to avoid redoing this step. With that in mind, run the following command:

If you managed to do this without a problem, then those are more than phenomenal, as you've also now verified that you gmp installation is correct and it works. If not, then you will most likely find answers as to why in, as always, the *config.log* that you'll find in this very folder. Assuming that it did work correctly, run the following commands[9]:

After you've successfully completed these steps, you must now move on to the next library needed for GCC.

5.7.6 Compiling and installing MPC

Like a broken record, we'll now perform a very similar procedure for MPC, the, thankfully, final library that will be needed in order to compile GCC. This time around the command to configure is just a tad longer, as it requires both GMP and MPFR to be compiled. Head into the folder:

Now run the following configuration command:

[9] I'll now skip over the lengthy explanation, as it is pretty much exactly the same as with GMP.

If it completes you are now ready to install it, which follows more or less the same pattern as before[10], and will require you to input these commands:

Fortunately, if everything up to this point has been successfully completed, then, our goal of installing gcc is quite close, however, there is still some work to be done.

5.7.7 Compiling and installing GCC

At last we've come to face the great challenge that awaits us, compiling the behemoth that is GCC. Though, now being armed with all the libraries we could possibly need, we are now completely ready to embark on this long journey. At first, it'll all be quite simple, and eerily similar to what we've faced before with all of the libraries. However, this time things will be a bit more involved. Now, in order to begin first head to where gcc is:

Once you are here, you must first create a new folder and move inside it:

This is necessary as GCC cannot be built right in its main directory, and as such, this is a convenient and simple solution to this issue. Now, it is time to begin configuring GCC, and, before we do, I need to warn you of the following: **GCC, being such a complex application, is quite difficult to configure and compile correctly on the first try, and it took me quite some time to find which options undoubtedly worked and which didn't with my particular raspberry pi system. As such, do not be alarmed if the instructions you find here do not work perfectly on your system, as it is perfectly possible that some minute detail has been overlooked on either of our ends, or you have tweaked your system in some way that doesn't allow you to build with these specific instructions.** With this out of the way, let us begin, run the following command, which, in theory, already has been tailored to work well with the Raspberry Pi:

[10]Interestingly, this isn't in any way a bad thing, as one of the principles of the *GNU autoconf/automake* system is homogeneity across the programs deployed with it, in the sense that they should all follow a similar installation procedure and be reasonably simple to get into a working state.

I don't exactly recall how long this should take, but be sure that it won't be a quick process, therefore, I would advise that you turn your attention to some other task until you see this finish, and don't forget to be patient. If it fails, you'll probably find the answer as to what happened in the *config.log* file that's left afterwards. If it doesn't, then it means that you are well on your way to compiling GCC. The next step to be taken is quite simple, though it is arguably the most difficult:

While at first glance it seems to make no sense that this step is difficult, it must be understood that the make process might fail and it might be quite difficult to recover from those errors, nonetheless, it is absolutely solvable, and, given enough time, I can assure you that you will find an adequate solution to the issue. Now, another thing that must be said is that you will need to be very patient, more than you could probably imagine, so don't despair if it seems as if it's taking forever. Once it's done, there's quite a great many things to be done. The very next thing you need to do is quite simple, and, as ever, quite similar to what we've done:

Once again, be patient while the application installs, as it might take a while. If you did these steps correctly, you will now be able to do a simple tests to see if things are, at least in some way, correctly installed and ready to use:

You should see some output stating the program's version and how it was configured, among other things. If you do, then you did things, so far, correctly.

5.7.8 Preparing a modulefile for GCC

Now it is time to create our very own module for gcc, as, going forward, we will need to be able to select easily which version we want without having to fiddle around with manually changing environment variables. As such, I will now leave you with the text to the module and you would do well to use it, though please be sure to look up the manual to the modulefiles so you understand what comes next:

```
#%Module10#####################################################
##
## Modulefile for Gcc 4.8.5
##
##
```

```
module-whatis "Prepares the environment to use GCC 4.8.5"

set     topdir          /share/apps/gcc/4.8.5
set     version         4.8.5
set     sys          armv6

module load gmp/6.1.0 mpfr/3.1.3 mpc/1.0.2

setenv      CC          $topdir/bin/gcc
setenv      GCC         $topdir/bin/gcc
setenv      FC          $topdir/bin/gfortran
setenv      F77         $topdir/bin/gfortran
setenv      F90         $topdir/bin/gfortran

prepend-path    C_INCLUDE_PATH      $topdir/include
prepend-path    CPLUS_INCLUDE_PATH  $topdir/include
prepend-path    PATH                $topdir/bin
prepend-path    MANPATH             $topdir/share/man
prepend-path    LD_LIBRARY_PATH     $topdir/lib
prepend-path    LD_RUN_PATH         $topdir/lib
prepend-path    LIBRARY_PATH        $topdir/lib
#####################################################################
```

This, as it stands, isn't ready to be used yet since we need to create the modules for GMP, MPFR and MPC, and only then would the new GCC be truly ready to be used. The first step you must take is to navigate to the following folder:

```
$ cd /share/apps/modules/3.2.10/modulefiles
```

Now, you must create a folder for your new apps modulefile, and create a blank file which you will now edit:

```
$ sudo mkdir gcc
$ cd gcc
$ sudo touch 4.8.5
```

Now that it's ready, it's time to add the module's text to the file:

```
$ sudo chown pi:pi 4.8.5
$ vi 4.8.5
$ sudo chown root:root 4.8.5
```

After you've done all of this, you are now ready to proceed to creating all the modulefiles that you need to do next.

5.7.9 Preparing a modulefile for GMP, MPFR and MPC

In order to finally get GCC working, and allowing you to use it's module, we need to create modules for our GMP, MPFR, and MPC libraries. As such, go ahead and create folders for all of the modules:

```
$ cd ..
$ sudo mkdir {gmp,mpfr,mpc}
$ sudo touch {gmp/6.1.0,mpfr/3.1.3,mpc/1.0.3}
$ sudo chown -R pi:pi {gmp,mpfr,mpc}
$ vi gmp/6.1.0
```

At this point, insert the following text into the *6.1.0* file:

```
#%Module10######################################################
##
## Modulefile for GMP 6.1.0
##
##
proc ModulesHelp {} {
        global version modroot

                puts stderr "Prepares the environment to use GMP 6.1.0"
}

module-whatis "Prepares the environment to use GMP 6.1.0"

set     topdir          /share/apps/gmp/6.1.0
set     version         6.1.0
set     sys         armv6

prepend-path    C_INCLUDE_PATH      $topdir/include
prepend-path    CPLUS_INCLUDE_PATH  $topdir/include
prepend-path    LD_LIBRARY_PATH     $topdir/lib
prepend-path    LD_RUN_PATH     $topdir/lib
prepend-path    LIBRARY_PATH        $topdir/lib
#################################################################
```

Once you've done this, move on to the next file:

```
$ vi mpfr/3.1.3
```

At this point, insert the following text into the *3.1.3* file:

```
#%Module10###################################################
##
## Modulefile for MPFR 3.1.3
##
##
proc ModulesHelp {} {
        global version modroot

                puts stderr "Prepares the environment to use MPFR 3.1.3"
}

module-whatis "Prepares the environment to use MPFR 3.1.3"

set     topdir          /share/apps/mpfr/3.1.3
set     version         3.1.3
set     sys         armv6

prepend-path    C_INCLUDE_PATH      $topdir/include
prepend-path    CPLUS_INCLUDE_PATH  $topdir/include
prepend-path    LD_LIBRARY_PATH     $topdir/lib
prepend-path    LD_RUN_PATH     $topdir/lib
prepend-path    LIBRARY_PATH        $topdir/lib
#####################################################################
```

Once you've done this, move on to the next file:

```
$ vi mpc/1.0.3
```

At this point, insert the following text into the *1.0.3* file:

```
#%Module10###################################################
##
## Modulefile for MPC 1.0.3
##
##
proc ModulesHelp {} {
        global version modroot

                puts stderr "Prepares the environment to use MPC 1.0.3"
}

module-whatis "Prepares the environment to use MPC 1.0.3"

set     topdir          /share/apps/mpc/1.0.3
set     version         1.0.3
set     sys         armv6

prepend-path    C_INCLUDE_PATH      $topdir/include
```

```
prepend-path    CPLUS_INCLUDE_PATH  $topdir/include
prepend-path    LD_LIBRARY_PATH     $topdir/lib
prepend-path    LD_RUN_PATH     $topdir/lib
prepend-path    LIBRARY_PATH        $topdir/lib
##############################################################################
```

Now, if you've done all these steps correctly, you will be able to see new items pop up when you run the following command:

```
$ module avail
```

It should show you that all of our new modules are now ready to be used. Go ahead and test this by doing the following:

```
$ module load gcc/4.8.5
$ module list
```

You should now see that the GCC module is loaded along with the GMP, MPFR and MPC modules. Finally, test that everything went right with the following command:

```
$ gcc -v
```

If all went well, then you should see it say that you are running GCC version 4.8.5, and no errors should show up. And with that, you'd be done with the build and installation of GCC. If you are so inclined, and depending on how much space your SD card has, I would advise you to delete the sources and extracted folders **(but not the installed folders in */share*!)** for GCC, it's libraries and for Modules, though this is entirely up to you.

5.8 Obtaining an MPI implementation

Our machine is now equipped to run and compile C and C++ sources that require some of the newer features of these languages. Unfortunately we're not quite yet ready to begin preparing and deploying parallel applications, as we have no way for the applications to talk to one another whilst they're running on different machines. This is where MPI comes in, as it defines a message passing interface that standardizes how programs ought to talk to each of their instances when they're running on parallel machines. In terms of what this means for us, well, it means we will need another layer of software on top of our compiler, and its associated libraries and headers in order to compile the parallel programs we will want to run.

5.8.1 Overview of what will be done

In a stroke of luck for us, setting up our MPI implementation isn't truly that difficult, and it, thankfully, relies on the now familiar paradigm of the *automake/autoconf* system. In this case, we've chose to stick to a tried and true MPI implementation, which is OpenMPI version 1.10.2, which is the latest stable release. As with what we've done before, we'll delve into the world of

compilation once more in order to build this software just how we want it. With this in mind, what we'll do is the following:

- We'll download and obtain the source for OpenMPI version 1.10.2
- We'll compile it and configure it to our liking
- We'll install it under */share* so that in due time, all of our nodes can access it.

5.8.2 Obtaining the Software

In order to get the latest version of OpenMPI, we'll need to download it from their website with the following set of commands (and we'll once again download everything to our home):

```
$ cd
$ wget https://www.open-mpi.org/software/ompi/v1.10/\
    downloads/openmpi-1.10.2.tar.gz
    $ tar xzvf openmpi-1.10.2.tar.gz
```

Now that you've extracted it, it is time to prepare to compile it.

5.8.3 Preparing to compile

Interestingly, there really aren't a great deal of new things to take into consideration once we begin the journey of building OpenMPI, and as such, it makes sense to say that pretty much all of the warnings and considerations that applied to GCC apply to OpenMPI, though, in this case, with a minimally customized configuration and our newly installed GCC things should compile rather smoothly. If, for some reason OpenMPI has strange configuration issues, you'd do yourself a great favor, as always, by checking your *config.log*. With not much else, then to say, we are ready to begin.

5.8.4 Compiling and installing OpenMPI

First, head into the folder you've extracted:

```
$ cd; cd openmpi-1.10.2
```

Once you're there, I'd advise reading through the *README*, for it may contain interesting information that may be directly relevant to you particular use of the cluster. Once you're done with that you may realize that it is then time to begin configuring the software to our specific system's needs. This time, not too may options will be needed to be used to produce a fully functional OpenMPI build. I must, nonetheless, advise that as time goes on you come to find a version that you trust and find completely stable on your particular system, as, in due time, you'll find that it is nearly inveitable to find bugs in the OpenMPI you have installed, and, this willingness to change is quite important and useful, even if it is just in search of performance enhancements[11]. Thus if you've come to terms

[11] This, though, does not mean that the version we'll use is in any way unsable or bad.

with this, I may suggest that you do the following[12]:

```
$ ./configure --prefix=/share/apps/openmpi/1.10.2/g
                                    --build=arm-linux-gnueabihf
```

This configuration will probably take some time, but, in theory should not fail, provided you've loaded the correct modules and made sure that you inputted all of the parameters correctly. The next step is somewhat different to what we've done before, as it is recommended that you do both the build and installation in one step, so we'll prepare our folders before we do that and then begin compiling:

```
$ sudo mkdir -p /share/apps/
$ sudo chown -R pi:pi /share
$ make all install
$ sudo chown -R root:root /s
```

If all of these steps complete without an issue, then the software has been installed. Verify this by running the following command and checking that the output is indeed of the GCC you compiled it with:

```
$ /share/apps/openmpi/1.10.2
```

Once you've verified that all is well, it is now time to realize that work still has to be done before we can use our new OpenMPI.As you might imagine, we'll create a new module for it, that will allow us to use it with ease.

5.8.5 Preparing a modulefile for OpenMPI

Just as we had to do with GCC, we will need to create a module for OpenMPI, so that we may simplify our lives just a bit whenever we are to use it. With that in mind, let us do the following[13]:

```
$ cd /share/apps/modules
$ sudo mkdir openmpi
$ sudo touch openmpi/1.1
$ sudo chown pi:pi openm
$ vi openmpi/1.10.2_gcc-
```

This new file we'll be constructing will be somewhat longer but in now way really different to what we're used to, so with that in mind, input the following into your file:

[12] This time around you'll probably notice that I'll use a different and quite strange foldering convention for our new OpenMPI, and, the reason for this is rather simple, as time goes on and the needs of your cluster change, you may be inclined to test the same version of openmpi with a different compiler, and, as such, it becomes important to be to easily file the software as you go along. This practice is inspired by how our University's cluster has done things, and it has worked wonders, as it greatly simplifies searching for the specific piece of software we are looking for, though, of course, you are free to change this up however you want. It also somewhat eases scripting, if you ever need to do such things.

[13] Once again you'll notice a difference in how the module is named, as it is quite important to know exactly what a module really is loading, and it isn't an issue as modules does have autocomplete built in

```
#%Module10###################################################
##
## Modulefile for OpenMPI 1.10.2 built with GCC 4.8.5
##
##
                                                    proc ModulesHelp {} {
                                                        global version m

                                                            puts std
                                                    }

module-whatis "Prepares the environment to use OpenMPI 1.10.2"

set     topdir  /share/apps/openmpi/1.10.2/gcc/4.8.5
set     version         1.10.2
set     sys         armv6

module load gcc/4.8.5

prepend-path    PATH            $topdir/bin
prepend-path    C_INCLUDE_PATH      $topdir/include
prepend-path    CXX_INCLUDE_PATH    $topdir/include
prepend-path    CPLUS_INCLUDE_PATH  $topdir/include
prepend-path    LD_LIBRARY_PATH     $topdir/lib
prepend-path    LD_RUN_PATH     $topdir/lib
prepend-path    LIBRARY_PATH        $topdir/lib
prepend-path    MANPATH         $topdir/share/man
setenv      MPI_BIN         $topdir/bin
setenv      MPI_SYSCONFIG       $topdir/etc
setenv      MPI_RUN         $topdir/bin
setenv      MPI_INCLUDE     $topdir/include
setenv      MPI_LIB         $topdir/lib
setenv      MPI_MAN         $topdir/share/man
setenv      MPI_COMPILER        mpicc
setenv      MPI_SUFFIX          _openmpi
setenv      MPI_HOME            $topdir
setenv      CC                  mpicc
setenv      CXX                 mpiCC
setenv      F77                 mpifort
setenv      F90                 mpifort
setenv      FC                  mpifort
setenv      MPI_INTERCONNECT    p4
setenv      MPI_VENDOR      openmpi
###################################################################
```

As you can see, this particular modulefile is somewhat long, but is nonetheless

very useful, as it contains most environment variables you might need at some point with OpenMPI. Going forward we will probably make quite a bit of use of OpenMPI, so these steps have now saved us quite a bit of time.

5.9 Configuring your network

Up to this point your raspberry pi has, most likely, been using a dynamic ip address, and this, unfortunately is not how things should be, as it is quite useful to actually know where your node is on the network[14]. With this in mind we will need to modify our network configuration and make some assumptions that may be beneficial to us. For now, however, we will do a bare bones process that will allow us to prepare our cluster for it's initial testing, and will, hopefully, give you the insight necessary to continue expanding the cluster later on.

5.9.1 Setting up a static IP

First of all, if possible, verify which IP's in your router's LAN have been taken, and select one that doesn't conflict with any of those. In my case (and hopefully yours) we found *192.128.1.130* to be a perfectly acceptable address to give to our master node. Therefore, we will now change our network configuration to make this our machine's default state:

```
$ cd /etc/network/
$ sudo vi interfaces
```

Now, once you've opened up this file, modify it so that it looks like this:

```
auto lo

iface lo inet loopback
iface eth0 inet static
address 192.168.1.130
netmask 255.255.255.0
gateway   192.168.1.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

[14]Now this brings up a very interesting and difficult thing about creating a cluster with Raspberry Pis. They typically only have one standard network interface to work with, however, in a cluster, you should ideally have two networks, one that is connected to the outside world and isn't. The basic idea goes something like this: your master node should be connected to the internet as it will most likely need it at some point, in fact, we have used it quite a bit; however, you compute nodes should not be connected to the internet, as they can and will receive everything they need from your master node. Unfortunately this isn't really possible unless you have some sort of external network interface and an additional network switch to create your secondary network. Given these limitations, it should then come as no surprise that we will connect all of our nodes to the same switch/router, and therefore they will also have access to the internet, fortunately, this really is no problem for us.

5.9.2 Modifying your */etc/hosts* file

Now, another important aspect of a cluster is the fact that these machines should be able to resolve a simple hostname in order to simplify connecting and setting up MPI machine files, this, also, has the benefit of making it easier for us to know where exactly we are connecting, and it also removes a large part of the guessing game that would come with just using IP addresses. To this end we must then modify our */etc/hosts* file[15]. Now, in order to do this, we must first decide where we will place our next nodes on our network, and how we will do our naming scheme. In this case, we will follow the standard convention of *compute-X-Y*, where X indicates what rack it is on and Y indicates its position on the rack itself. Additionally, we will start assigning our new node's IP's from *192.168.1.131* onwards. Therefore, at the very least we will modify our master node's */etc/hosts* file with at least one node. In order to do this do the following:

```
$ cd /etc/
$ sudo vi hosts
```

Once you've opened it, go ahead and add the following to the very end of the file:

```
192.168.1.131 compute-0-0
```

This, now, will allow you to type out compute-0-0 (when we create it) so that you don't have to know each node's IP address by heart. As you can now probably imagine, this will greatly simplify using ssh and filling out which computers you want to run parallel programs on.

5.10 Final steps of the basic setup

Now that you've finished doing all of these long and admittedly arduous tasks, you are now very nearly done. The next steps I'd recommend doing is some basic cleanup on all of the places you've worked, and some basic re-verification that all of the software is working as it is intended and that the modules work correctly. One easy way to do this is to check what you environment variables look like after you load the OpenMPI module, as they should, in theory, be full of the paths you've set for them in your scripts. You can easily do this as follows:

```
$ module load openmpi/1.10.2_gcc-4.8.5
$ env | less
```

Once you are satisfied with what you see there, and you are somewhat sure that all works as it should, I would then say it is now advisable to reboot your device so that the changes you've made to your network configuration are made permanent. You can easily do this as follows:

[15] Interestingly, distributions such as Rocks have an automated *hosts* modification system that automatically adds every new machine to every node's *hosts* file, which greatly simplifies the administration of the cluster.

```
$ sudo reboot
```

After this you can now confidently state that you have functional (if a bit barebones) master node. Unfortunately we don't really have any cool applications to run on it and test that seductively mysterious MPI functionality over our network. To fix this we must first set up for ourselves a second node, that will act as our trusty computation node[16]. After we've done that, then we can finally go and set up some amazing things to see just how powerful and versatile our cluster is.

6 Setting up a secondary node

Now that our master node's basic setup is done, it is now time to begin preparing our first compute node, for one very important reason: we will need it to test our parallel applications, and it is therefore quite useful to have it ready. Fortunately it is not a very difficult process, as, differing from what we've done before, we will not need to a lot of things in order to prepare our node. Nodes, in theory, should not have a lot done to them aside from a very basic setup process, since, from now on, they will get all of the things they need from our shared storage. It is therefore advisable to not do too much to them in order to not clutter them with unnecessary things. With this in mind, we will begin with what we need to do.

6.1 Getting the latest versions of the default packages

Similarly to what we did on our master node, you should have a base and clean raspbian image mounted on this Raspberry Pi, there, once you've done that, proceed to login using the same default user and password (user=*pi*, password=*raspberry*), and make sure you are now successfully logged in. Now, run the following commands to update all of the base and default packages:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Once that finishes, you are now ready to proceed to setting your node up.

6.2 Running the configuration script

This time around you, just as before, will need to run Raspbian's included configuration script so that ssh is enabled from now on. In addition to that,

[16] As time passes by and, hopefully, your love and ambition for your own cluster grows, you will begin to think of a ton of ways you can employ new nodes. For example, you could have one node as a sort of firewall for your master node, another as your storage node, a couple as dedicated computation nodes, another as an account authentication node, and so on and so forth. Truly, the only limits to this are your imagination and your budget. If you're feeling specially adventurous you may want to even try out creating complex and novel network arrangements.

you will also need to change this Raspberry Pi's hostname, in order to avoid confusion and problems. To do this, do just as before and run the script:

```
$ sudo raspi-config
```

Once you are there, under the advanced options, you will find the options you need to do the aformentioned steps. Go ahead and do that and let the device reboot after you do so.

6.3 Installing all the needed utilities

For now, all we will truly need on this node is the ability to see the files that our master node has on its shared drive. For this we will need some utilities, though they are very easy to install, and will allow us to get our pi in working order in no time. To do this run the following command:

```
$ sudo apt-get install rpcbind nfs-common
```

After it succesfully installs it, we will need to enable a service on our system, as, without it, we will be unable to mount our new filesystem, and, additionally we will want it to start as soon as we boot. With that in mind, run the following commands:

```
$ sudo service rpcbind start
$ sudo update-rc.d rpcbind enable
```

At this point we are now able to mount the NFS file system we set up at the very beginning on our master node.

6.4 Mounting the master node's NFS share

By now you've probably noticed that I really didn't install too many things on our new node and instead only sought to prepare it to mount the master node's */share/* folder on our system. This is mainly due to the fact that it makes no sense to clutter up this node with unnecessary applications that will be shared anyways, and as such, it is a much smarter move to things this way. Additionally, it allows one to learn about something that basically all modern clusters do. In order to this in a permanent way, allowing us to always have it available when the device turns on, we will need to follow a couple of steps. First, create an empty folder where you'll mount the remote share:

```
$ cd /
$ sudo mkdir share
```

With that done, go over to your */etc/* folder and edit your */etc/fstab* file:

```
$ cd /etc/
$ sudo vi fstab
```

At the end of the file add a line as follows:

```
192.168.1.130:/share/ /share
nfs      0     0
```

This, from now on, will automatically mount the file system on boot, but, for now, we'll do it manually so we don't have to reboot the device[17]. So in order to do this, enter the following command:

```
sudo mount -t nfs 192.168.1.130:/share /sh
```

Now, go ahead and enter into the folder and verify that our apps folder is there and that you can now see all the apps we compiled and created back on our master node. You can do this as follows:

```
$ cd /share
$ ls
```

If you can both access and read the files you now mounted on */share*, then it all went well, and you are ready to proceed.

6.5 Preparing modules

Now that we can access the files on our master node, it is time to add some of the functionality that is on the main server onto our new node, most notably, we might, at some point, require modules on it, so it's a great idea to prepare this computer to use them on demand. Fortunately, now that we can access them remotely, we do not need to compile anything as it's been already done for us, this then, has the fantastic added benefit that we can now simply do some simple setup steps and work far quicker than before. To attain this, we must run some simple commands:

```
$ sudo cp /share/apps/modules/3.2.10/3.2.1
    modules.sh /etc/profile.d
    $ source /etc/profile.d/modules.sh
```

This will now allow us to do full use of modules whilst on our node, and we can immediately test this by doing the following:

```
$ module avail
```

This command should now let us see all the modules that we now have at our disposal.

6.6 Configuring your network

The next, and extremely important step to be taken, is to correct our pi's network configuration, just as we had done with our master node. The reason for this is simple, it is far simples to work with machines that have a statically assigned IP address, as it'll ease the work needed to both use and find them.

[17] Don't worry, this will in no way break what we've done with */etc/fstab*, it will simply allow us to begin working from now on with our shared file system.

As you might imagine, we'll need to modify our */etc/network/interfaces* file and we'll need to modify our */etc/hosts*, though, just as before, this will not be complicated. Ideally as time goes on and we add more devices, we'll simplify this process and hopefully automate it, but, for now, we'll do it manually.

6.6.1 Setting up a static IP

Just as we had done before, we'll need to do some modification to out */etc/network/interfaces* file, in the exact same manner that we did with our master node. To do so, follow the following steps:

```
$ cd /etc/network
$ sudo vi interfaces
```

Edit the file so that it looks as follows:

```
auto lo

iface lo inet loopback
iface eth0 inet static
address 192.168.1.130
netmask 255.255.255.0
gateway   192.168.1.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_suppl
iface default inet dhcp
```

Once you've done that, it is now time to take a look at your */etc/hosts* file.

6.6.2 Modifying your */etc/hosts* file

Running the risk of sounding like a broken record, it necessary to once again do this procedure, as it is simply a good practice, and it doesn't really take any effort. To do this move to your */etc* folder and open your *hosts* file:

```
$ cd /etc/
$ sudo vi hosts
```

Next insert the following into the file at its end:

```
192.168.1.130 raspberrypi
```

Once you've done this, the node is now, in theory, ready to be used.

6.7 Final steps of the basic setup

There is really not much to be done to the node now that you've gotten it up and running. Just make sure you don't have too much of its local filesystem and make sure that you don't have any unnecessary application launching into the

background. After you've done all these checks, it is time to reboot it to apply your new network configuration and test your initialization configurations. So run:

```
$ sudo reboot
```

Once it reboots you should see that the NFS share has been successfully mounted, and that you now have a static IP address. Once you are happy with this, you may now say that you do indeed have two working nodes in a Raspbery Pi Cluster. From now on we will focus on actually testing our cluster's capabilities, and making sure that it actually works as we intended. Though we might encounter some issues along the way, I'm completely sure that none are too large in any single way to stop us now!

7 MPI, Stability and Performance Tests

Now that we have all of the basic functionalities done and ready to be used, a wonderful time comes, we are going to actually begin compiling and using powerful apps that will allow us to see just how well all our work went. This time around, we will focus on two quite practical and interesting tests that will help us measure if our newly created infrastructure is actually any good. The first of these is a simple application whose only job is to calculate Pi[18] utilizing parallel methods; meanwhile, the second focuses on giving an accurate reading of the peak performance of the cluster as a whole, and is therefore quite interesting to put into motion. Ideally you will do this procedure on the master node, as it should be done, but in theory you could do this from any other node aswell, atleast the first program.

7.1 MPI Calculation Example - C Version

This is the aforementioned program that calculates Pi's value utilizing parallel methods to solve it. It is very interesting in the sense that it exemplifies exactly what a parallel program should do, and it is also quite simply and therefore allows us to immediately gauge and see that our whole tool chain is correctly done. As such this first test is of immense value to us, and will be our true starting point for the utilization of our cluster.

7.1.1 Overview of what will be done

In order to utilize this code we will need to compile using our new OpenMPI wrapper, so that it has inside of it the proper MPI support and libraries linked. We will obtain it from its source and then do a quick compilation. Afterwards we will do our first proper test of the program on both of the machines at once, hopefully with no issues, indicating to us that we've done a good job setting everything up, and if for some reason we haven't, we'll probably be able to find out where we screwed up.

[18] Which, in a way, is quite appropriate for this project

7.1.2 Obtaining the software

The software in question is part of the examples of an Introduction to Parallel Computing website prepared by *computing.llnl.org*, and as such is hosted there. Therefore our first step will be to download the software and place it into an appropriate folder. Given the fact that the app is just one source file, we'd do well to simply place the source in our */share/apps* folder, as it really would make no difference. With this in mind, let's do the following:

```
$ cd /share/apps/
$ sudo mkdir -p piexample/noversion/gc
$ cd piexample/noversion/gcc/4.8.5/ope
$ sudo chown pi:pi 1.10.2
$ cd 1.10.2
$ wget computing.llnl.gov/tutorials/mp
```

Now, at this point, we would normally return the folder's permission's to root, however, we will not, as we will go ahead and compile it right away right there.

7.1.3 Preparing to compile

In order to correctly compile this program you will need to load OpenMPI's module, as it will contain all the necessary include paths and library paths to make this work. To do this just run:

```
$ module load openmpi/1.10.2_gcc-4.8.5
```

After doing this you should be more than ready to compile the program!

7.1.4 Compiling the software

Compiling this program is an extremely simple task, and, additionally, allows you to familiarize yourself with the use of our MPI Implementation's GCC wrappers. To do the compilation, all you need to do is run the following command:

```
$ mpicc mpi_pi_reduce.c -o mpi_pi_redu
```

This will give you a file named *mpi_pi_reduce* (without the *.c* extension), which is the program we will eventually run.

7.1.5 Testing the software

Here is where we finally are able to get the gratification of actually running the software on several machines, and this is where everything will either go really well or really badly. Thus, with the entireity of our hopes pinned on actually getting this to work right, we shall begin by creating a *(*machinefile), which is a file that OpenMPI will use to know what machines it needs to run the program on. As such, your first task is to create a file and open it:

```
$ touch machinefile
$ vi machinefile
```

Now, with the file open, you should put the following into it:

```
localhost
compute-0-0
```

Essentially, you are telling OpenMPI that you want two processes, one on this machine and one on *compute-0-0*. Now, with that done, all that really remains to be is run the program. To do this, go ahead and run the following command[19]:

```
$ mpirun -np 2 -machinefile machi
    piexample/noversion/gcc/4.8.5/
```

After doing this, you should see the program launch and ask you for a password, this, fortunately is something normal, and something we'll eliminate soon, but, for now, just know that you need to input *raspberry* as it is the default password of the *pi* user. Once you do that, the program will continue and you will, hopefully, see the program output it's approximation of Pi with no issue. If it does, then everything truly went alright after all.

7.2 HPL Performance Benchmark

Now comes the time where we get to really see just how much "oomph" our system packs "under the hood". The HPL Benchmark is the tool that will allow us to do with ease and speed, allowing us to see just how well our system performs under load[20]. This benchmark will also allow us to get a far better feel as to what it will be like to configure, compile and run scientific and distributed software, as they tend to occasionally be quite unwieldy and difficult to put into working order.

7.2.1 Overview of what will be done

HPL is quite interesting program to set up, as it has been prepared to essentially be as tight and well optimized as possible, with nothing that isn't useful to it within it. Now, in order to get it running, we must realize that the program itself has one truly important dependency that has to be built for it, which is some form of a linear algebra library. In this case we have several options to choose from, including one the many BLAS implementation or the VSIPL library, however, for our particular case, we will be using OpenBLAS, which, with how we've tested it, has shown the best performance and behavior. The specific version's we'll be using are 2.1 for HPL and 0.2.15 for OpenBLAS, as these version have been shown to be quite stable. Now, with this in mind, I'll briefly outline the steps we must follow to install HPL and run.

[19] If you experience issues with this, substitute *mpirun* with */share/apps/openmpi/1.10.2/gcc/4.8.5/bin/mpirun* and *machinefile* with */share/apps/piexample/noversion/gcc/4.8.5/openmpi/1.10.2/machinefile*

[20] Now, I understand that there are several other benchmarks out there, that, in theory, give a far better and more accurate representation of how well a system will perform, i.e. HPCG, however, as the TOP500 ranking uses HPL, it seems obvious and practical to choose this one over the other.

- We must download and obtain the source for both OpenBLAS and HPL.
- We must configure, compile and install OpenBLAS 0.2.15.
- We must configure, compile and install HPL 2.1.
- We must tune and configure HPL's parameters file.
- Finally, we must run the application.

7.2.2 Obtaining the software

This time around, we'll need to download and extract two pieces of software, so go ahead and do the following:

```
$ cd
$ wget http://github.co
$ wget http://www.netl
$ tar xzvf v0.2.15.tar
$ tar xzvf hpl-2.1.tar
```

Now that you've finished obtaining the softare, go ahead and head into OpenBLAS's folder:

```
$ cd OpenBLAS-0.2.15
```

With that done, we'll now begin to build it.

7.2.3 Compiling and installing OpenBLAS

OpenBLAS, unlike most of the other applications we've installed so far, has a rather simple build and installation process. This is due to the fact that OpenBLAS has been constructed in a highly portable fashion, and, therefore, has great auto-detecting capabilities to make it easily buildable in most environments. Let us then, begin the build process with one very simple command:

```
$ make
```

Once it finishes, you'll need to create the folder structure where you'll install it, and you'll need to have write privilege. So, to do so, do the following:

```
$ sudo mkdir -p /share
$ sudo chown pi:pi /sh
```

We'll shortly set the permissions back, but for now, do the following:

```
$ make install PREFIX=
```

After this is done, it is now time to return the folder's permissions to *root*, so that we don't accidentally screw anything up:

```
$ sudo chown root:root
```

With this now done, you now have the necessary libraries to be able to compile HPL, so that we may run our second and final test.

7.2.4 Compiling and Installing HPL

HPL, as with most software used on high performance computing clusters, is somewhat tricky to get working, but only somewhat, after a bit, it becomes rather straightforward. In order to correctly configure it, go ahead and move to its folder:

```
$ cd ..
$ cd hpl-2.1
```

Once you are there, you will need to grab one of the pre-made setup files and modify it so that it is compatible with our system. To do this, copy over one of the files from the *setup* folder, and then rename it to something memorable. After that you must open it and modify it to fit what we will use. Do the following:

```
$ cp setup/Make.Linux_I
$ mv Make.Linux_PII_CBI
$ vi Make.PI
```

Now that you've opened it, I will guide you through all the changes you have to make. Line 64 should read:

```
ARCH
= PI
```

Lines 70 through 75 should be:

```
TOPdir
= /home/pi/hpl-2.1/
INCdir
= $(TOPdir)/include
BINdir
= $(TOPdir)/bin/$(ARCH)
    LIBdir   = $(TOPdir)/lib/$(ARCH)
#
    HPLlib   = $(LIBdir)/hpl.a
```

Lines 84 tom 86 should be:

```
    MPdir    = /share/apps/openmpi/1.10.2/gcc/4.8.5/
    MPinc    = -I$(MPdir)/include
    MPlib    = $(MPdir)/lib/libmpi.so
```

Lines 95 to 97 should be:

```
    LAdir    = /share/apps/openblas/0.2.15/gcc/4.8.5/openmpi/1.10.2/lib/
    LAinc    =
    LAlib    = $(LAdir)/libopenblas.a
```

Lines 169 to 171 should be:

```
CC = /share/apps/openblas/0.2.15/gcc/4.8.5/openmpi/1.10.2/bin/mpicc
CCNOOPT = $(HPL_DEFS)
CCFLAGS = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops
```

Line 176 should be:

```
LINKER = /share/apps/openblas/0.2.15/gcc/4.8.5/openmpi/1.10.2/bin/mpicc
```

Go ahead and save the file. Once you've done that, then you'll be ready to build the program itself. This can be done with the following commands:

```
$ make arch=PI
```

Once you are done with this, you will be able to find and executable within a newly created folder. It is this very file that'll be the one that will allow us to run all of our desired tests. Please verify that an executable named *xhpl* is present under the folder *bin/PI/*. To do this, do the following:

```
$ ls bin/PI/
```

You should see, within, the file previously mentioned. Additionally you should find a file named *HPL.dat*, which will be the file we'll be using to do our runtime configuration for *HPL*. If you successfully, found the files, it is now time to perform the install. Do the following commands to do so:

```
$ sudo mkdir -p /share/apps/hpl/2.1/gcc/4.8.5/openmpi/1.10.2/
$ sudo chown pi:pi /share/apps/hpl/2.1/gcc/4.8.5/openmpi/1.10.2/
$ mv bin/PI/{xhpl,HPL.dat} /share/apps/hpl/2.1/gcc/4.8.5/openmpi/1.10.2/
$ sudo chown -R root:root /share/apps/hpl/2.1/gcc/4.8.5/openmpi/1.10.2/
```

Once you've done this, you'll now have a working implementation of *HPL* that you can run.

7.2.5 Testing the software

Now that all, in terms of building and installing, is done, it is time to finally test the software, to guarantee that our system is now stable and in working order. To achieve this goal, we will follow a path that is quite similar to what we did, not too long ago, with our Pi calculator. The steps we will follow are quite simple, and not difficult in the slightest, as it all pretty much is just a rehash of what we've done before, with some slight changes. To begin, we must edit the HPL.dat file, so, go ahead and open it:

```
$ cd /share/apps/hpl/2.1/gcc/4.8.5/openmpi/1.10.2/
$ sudo vi HPL.dat
```

Now that you've opened it, you'll notice that this file is filled with configurations. Most of the parameters, in theory can be left as is, though with, some, notably the N size, which determines how much ram you can utilize at once, and the NB, which determines how the matrix will be split, are perfect examples of parameters that need to be changed. In addition to those, are the P and Q lines, as these indicate how your cluster is laid out. And, since ours is only 2

machines with one core each, we can only really run the setup as it is shown in the file. With all of this in mind, I will now provide you with the configuration I found optimal in this setup for both of our machines:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
9984         Ns
1            # of NBs
192           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
1            Ps
2            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

Now, armed with this, we are only one step away from actually being able to run our test, and, without further ado, here's what you need to do to run it:

```
$ sudo cp /share/apps/piexample/noversion/gcc/4.8.5\
    /openmpi/1.10.2/machinefile .
    $ mpirun -np 2 -machinefile machinefile xhpl
```

And with that, the test should be underway and you'll get a real world performance report when it finishes. Hopefully you'll be delighted to see that the cluster works quite well, and delivers quite good performance. With this in mind, don't forget to save that HPL.dat configuration, as you may come back to tweak it to learn more about optimizing parallel apps.

8 Distributed John the Ripper

Openwall's John the ripper has, for quite some time been a wonderful application for anybody interested in the world of cyber security, as it is a powerful and flexible program for cracking hashes and retrieving the passwords and access codes that lay within. However, a machine's performance has always been a deciding factor as to how effective John the ripper is at cracking the hashes it's fed. With this in mind, it should be no wonder that perhaps if two machines were to work in parallel, they could surely crack the passwords far easier than if they weren't. Now, john the ripper is released in three versions, a very stable and supported "Pro" version, a standard "free" version, and the most interesting of them all, the community enhanced "jumbo" version, where members of John the Ripper's community have taken the time to add many of the features they feel are lacking with the official john the ripper version. That version, then, will be the one we will be using, as it'll afford us the greatest flexibility and greatest amount of features. Additionally, it is easier to install and deploy on a system than its free (and official[21]) counterpart. With all of this now absorbed into our minds, let us now begin to set it up.

8.1 Overview of what's been done and what will be done

As you might probably be imagining, we've had to wade through an enormous amount of work to get here, and, well, it must be explained. First off, the latest John the Ripper "jumbo" contains quite a large body of work done by community programmers that decided to use some of the new features that were introduced for C++14, and unfortunately, compilers that didn't support this standard were simply unable to produce a working John the Ripper executable. Given the fact that the Raspberry Pi, by default, provides a very old version of GCC, it is no wonder then that it was necessary to move onward to a newer version of GCC. This had the fantastic side-effect of allowing me to introduce a large part of what needs to be done to configure and get a cluster in working order, at least at the very beginning, and it allowed me to present how and why things need to be done in order to use a real world application on a distributed system. Many of the things we've done, simply are the way they are so we can actually use John the Ripper, for example, in order to have a distributed John the Ripper work, you need to have the hash files somewhere where they can be easily found and read by all of the systems, inherently necessitating an NFS. This, basically, is what has allowed us to reach this point, where we are nearly ready to run the application that is the true focus of this work, a distributed version of John the Ripper. I will now present and outline, of the work that lies ahead, and afterwards illustrate what the final product of all of this work will be. To build and prepare the application for use, we will need to follow these steps:

[21] Nonetheless, they officially endorse and provide it on John the Ripper's website, and as such, really seems like the best alternative to use, shy of actually buying the official version.

- We will need to download the latest stable version of John the Ripper's "jumbo" community maintained version.
- We will then need to extract the software to a suitable location.
- Once that is done, we'll proceed to configure the software, so that it may be built according to our specific needs.
- We'll compile the software package and subsequently install it.
- We'll prepare a suitable set of hashes to be cracked.

After all of this is said and done, we'll finally have a tried and true cluster of our own, with a near infinite amount of possibilities lying ahead, just waiting to be undertaken.

8.2 Obtaining the software

Obtaining the software is rather simple, and really, given the fact that there are several options, any of them are perfectly fine. We will be using Openwall's official download link. Once we have it, we'll go ahead and extract it too. To obtain the source, run the following commmands:

```
$ cd
$ wget http://www.openwall.com/john/j/john-1.8.0-jumbo-1.tar.gz
$ tar xzvf john-1.8.0-jumbo-1.tar.gz
$ cd john-1.8.0-jumbo-1/src
```

Once you're there you should see a myriad of files, of all types, but, more important than anything else, you must be sure that there is a working configure file.

8.3 Configuring the software

John the Ripper, by default does not include a *configure* built in, making its build process more difficult and involved than it really should be (although it is simple). Fortunately for us, the "jumbo" version has a quite robust and well designed *configure script* which makes the build process rather straightforward. Just make sure that you have your OpenMPI module loaded once you begin configuring. To configure this program, run the following command:

```
$ ./configure --prefix=/share/apps/johntheripper/\
                1.8.0-jumbo-1/gcc/4.8.5/openmpi/1.10.2/ --enable-
                --disable-cuda --disable-openmp --build=arm-linux
```

Once it finishes, we should create the folder structure for it and get it ready for the installation with the following commands:

```
$ sudo mkdir -p /share/apps/johntheripper/1.8.0-ju
    /gcc/4.8.5/openmpi/1.10.2
    $ sudo chown pi:pi /share/apps/johntheripper/
        /gcc/4.8.5/openmpi/1.10.2
```

Now, with that done, it is time to begin compiling the software.

8.4 Compiling and installing the software

Compiling john the ripper, thankfully, is quite straightforward, and therefore really doesn't require much introduction. To begin compiling, run these commands:

```
$ make
$ make install
```

After this, in theory, john the ripper is done and installed and ready to use. So, with that in mind, I think it would make our lives easier to be able to call up John the Ripper and any of his abilities on demand, and, therfore, it makes sense for us to now go ahead and create a simple module for it.

8.5 Preparing a modulefile for John the Ripper

Without overextending myself, let me just say that, as before, having a module for john the ripper makes life a bit easier when working on the fly. Here is one module that would probably allow you to work with john quite well. First, let's create and edit our modulefile:

```
$ cd /share/apps/modules/3.2.10/modulefil
$ sudo mkdir johntheripper
$ sudo vi johntheripper/1.8.0-jumbo-1_gcc
```

Now that it's open, go ahead and edit it with the following info:

```
#%Module10#####################################################################
##
## Modulefile for John the Ripper 1.8.0-jumbo-1
##
##
##
                                        proc ModulesHelp { } {
                                                global version modroot

                                                        puts stderr "Prep
                                        }

module-whatis "Modulefile for John the Ripper 1.8.0-jumbo-1"

module load openmpi/1.10.2_gcc-4.8.5

set     topdir  /share/apps/johntheripper/1.8.0-jumbo-1\
            /gcc/4.8.5/openmpi/1.10.2
            set     version 1.8.0-jumbo-1
            set     app John the Ripper
```

```
prepend-path    PATH    $topdir/run
```

This should be all you need to get it up and running. At this point you really are ready to begin testing John the Ripper.

8.6 Preparing a set of hashes for testing

Now that you've produced a working executable, it is time to begin assembling a suitable set of hashes to crack. Personally, as this is more of test case than anything else, I believe it would be worth it to make it as simple as can be, as such, my suggestion is to create just 3 users and create low character count passwords, so you can tell that the software works, without committing to several hours of work trying to crack the passwords you've created. To do this, run these commands to generate the users:

```
$ sudo useradd user1 -p pw1
$ sudo useradd user2 -p pw2
$ sudo useradd user3 -p pw3
```

Now that you've done this, the system has 3 new users that are ready to crack. Unfortunately, we can't crack them quite yet. Modern linux systems have the user info carefully split among two files, */etc/passwd* and */etc/shadow*, and, in order to be able to crack it, you must use a utility provided by John the Ripper named *unshadow* to combine them into a single file. In order to do this, go ahead and run these commands:

```
            $ sudo su
# module load johntheripper/1.8.0-jumbo-1_gcc-4.8.5_openmpi-1.10.2
# unshadow /etc/passwd /etc/shadow > passdb.txt
# chown pi:pi passdb.txt
# mkdir /share/files
# mv passdb.txt /share/files/.
# cp /share/apps/piexample/noversion/gcc/4.8.5\
                /openmpi/1.10.2/machinefile /share/files/.
# logout
            $ cd /share/files
            $ module johntheripper/1.8.0-jumbo-1_gcc-4.8.5_openmpi-1.10.2
```

Now, finally, we have a file we can actually use, and now, finally, it is time to put our cluster to use.

8.7 Cracking the passwords

After all, we are now, thankfully, one command from completing the goals we've set out for ourselves, and that, in it of itself is more than enough reason to celebrate. Now, basking in the greatness of our accomplishments, let us now run this command:

```
$ mpirun -np 2 -machinefile machinefile john passdb.txt
```

Once it begins to churn through the passwords, it'll in show you that it has successfully cracked three passwords, and as such, it is, coupled to our earlier tests, absolute proof that you've constructed and put into working order a fantastic little cluster.

8.8 Performance

Now, as a small though on the matter, during my testing I found that the real world performance of John the Ripper was highly dependent on how well the network itself performed. However, if used on a network switch with reasonable speeds, there was a slight improvement in runtime cracking up to three passwords at a time. This was on the order of around a couple of minutes. However, once I did my testing with a switch from 1990, I found myself distraught by waiting over an hour for it to finish, instead of mere minutes on a single machine. There certainly is a great deal of work to be done yet experimenting and investigating, so this is still an interesting open question.

9 Thoughts for future expansion

Here at the end of this journey comes a time where now you must decide what you want to use this cluster for, in this case, one awesome usage case would be to have it be a small and almost non power consuming utility to check whether all the users on a given cluster are using safe passwords. But, to be completely frank and sincere, I believe that the possibilities are truly endless, and therefore the onus is now on you to demonstrate your abilities by proving that you are more than up to the challenge of continuing this cluster. If nothing else, this is a fantastic educational tool that can be used to teach a new generation of highly involved and competent system administrators.

10 Conclusion

Having said just about everything, now comes the time to truly reflect on what has been done. Though at times it may feel that this project leaves you wandering in the dark, an overwhelming sense of purpose overcomes you once you hit the first successes, and that is the true fuel and power that this tool has. The motivation of slowly creating something new out of what seems something completely detached to the world of high performance computing is perhaps the most powerful that can be given to a student.

This document's purpose is no other than to serve a small guiding light into the world of high performance computing, and, hopefully, it has lived up to be what it set out to be.

11 References

- D'Amore, M., Baggio, R., & Valdani, E. (2014). A Practical Approach to Big Data in Tourism: A Low Cost Raspberry Pi Cluster. Information and Communication Technologies in Tourism 2015, 169-181.
- Furlani, J. L. (1991). Modules: Providing a Flexible User Environment. Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V), 141-152.
- GNU MP 6.1.0. (n.d.). Retrieved February 04, 2016, from https://gmplib.org/manual/index.html
- GNU MPFR 3.1.3. (n.d.). Retrieved February 04, 2016, from http://www.mpfr.org/mpfr-current/mpfr.html
- Gite, V. (2007, December 18). Ubuntu Linux NFS Server installation and Configuration. Retrieved February 04, 2016, from http://www.cyberciti.biz/faq/how-to-ubuntu-nfs-server-configuration-howto/
- Gowtham S. (2007, July 2). HPL Benchmark For Single Processor Machines. Retrieved February 04, 2016, from http://sgowtham.com/journal/hpl-benchmark-for-single-processor-machines/
- Gowtham S. (2012, February 27). Rocks 5.4.2 HPL 2.0 benchmark with GCC 4.1.2. Retrieved February 04, 2016, from http://sgowtham.com/journal/hpl-2-0-benchmark-with-gcc-4-1-2-on-rocks-5-4-2/
- Html Documentation GNU MPC 1.0.3. (n.d.). Retrieved February 04, 2016, from http://www.multiprecision.org/index.php?prog=mpc
- Installing GCC: Configuration. (n.d.). Retrieved February 04, 2016, from https://gcc.gnu.org/install/configure.html
- Kiepert, J. (2013, May 22). Creating a Raspberry Pi-Based Beowulf Cluster. Retrieved February 4, 2016, from http://coen.boisestate.edu/ece/files/2013/05-/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf
- Leonard, P. (2012, June 18). Parallel Processing on the Pi (Bramble). Retrieved February 04, 2016, from http://westcoastlabs.blogspot.co.uk/2012/06/parallel-processing-on-pi-bramble.html
- Modules Software Environment. (n.d.). Retrieved February 04, 2016, from https://www.nersc.gov/users/software/nersc-user-environment/modules/
- Network configuration. (n.d.). Retrieved February 04, 2016, from https://wiki.archlinux.org/index.php/Network_configuration
- OpenBLAS User Manual. (n.d.). Retrieved February 04, 2016, from https://github.com/xianyi/OpenBLAS/wiki/User-Manual

- OpenMPI Project. (n.d.). FAQ: Building Open MPI. Retrieved February 04, 2016, from https://www.open-mpi.org/faq/?category=building
- Parallel and distributed processing with John the Ripper. (n.d.). Retrieved February 04, 2016, from http://openwall.info/wiki/john/parallelization
- RPi Distributions. (n.d.). Retrieved February 04, 2016, from http://elinux.org/RPi_Distributions
- Raspberry Pi Foundation (n.d.). What is a Raspberry Pi? Retrieved February 4, 2016, from https://www.raspberrypi.org/help/what-is-a-raspberry-pi/
- Raspbian Team. (n.d.). Raspbian FAQ. Retrieved February 04, 2016, from https://www.raspbian.org/RaspbianFAQ
- Redmond, E. (2012). Building a Riak Cluster on Raspberry Pi. Retrieved February 04, 2016, from http://basho.com/posts/technical/building-a-riak-cluster-on-raspberry-pi/
- Whitney, E., & Sprague, M. (2001). Drag your design environment kicking and screaming into the '90s with Modules! Synopsys Users' Group. Retrieved February 4, 2016, from http://modules.sourceforge.net/docs/MC2_whitney_paper.pdf