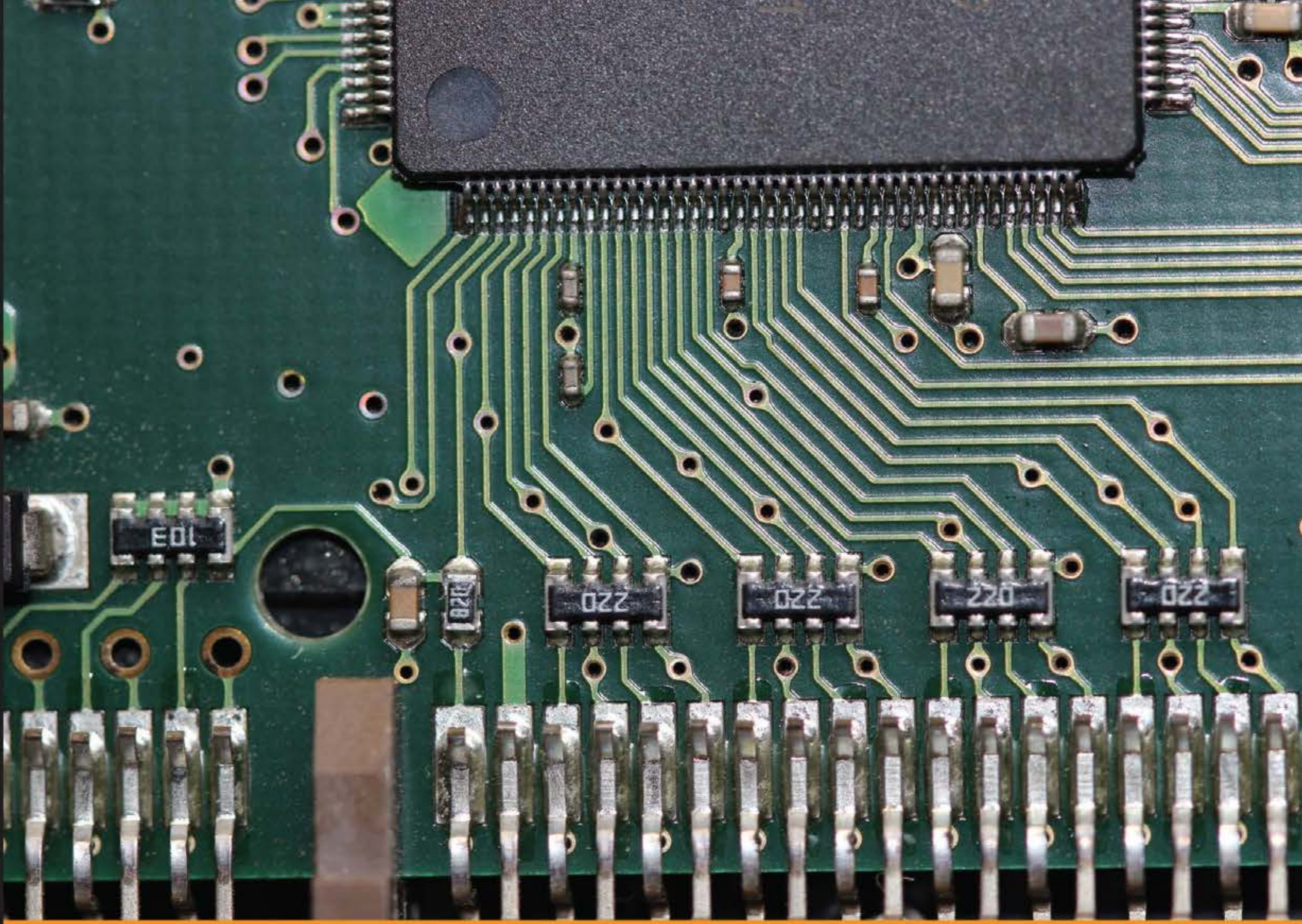


Raspberry Pi Super Cluster

Build your own parallel-computing cluster using Raspberry Pi in the comfort of your home

Andrew K. Dennis

[PACKT] open source* community experience distilled PUBLISHING

Raspberry Pi Super Cluster

Build your own parallel-computing cluster using Raspberry Pi in the comfort of your home

Andrew K. Dennis

[PACKT] open source*
PUBLISHING
community experience distilled

BIRMINGHAM - MUMBAI

Raspberry Pi Super Cluster

First published: November 2013

Production Reference: 1131113

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78328-619-5

`www.packtpub.com`

Cover Image by Aniket Sawant (`aniket_sawant_photography@hotmail.com`)

Credits

Author
Andrew K. Dennis

Reviewers
Prasanna Gautam
Sungjin Han
Claes Jakobsson

Acquisition Editors
Anthony Albuquerque
Edward Gordon

Commissioning Editor
Amit Ghodake

Technical Editors
Faisal Siddiqui
Sonali S. Vernekar

Project Coordinator
Aboli Ambardekar

Proofreader
Stephen Copestake

Indexer
Monica Ajmera Mehta

Graphics
Abhinash Sahu

Production Coordinator
Alwin Roy

Cover Work
Alwin Roy

About the Author

Andrew K. Dennis is the Manager of Application Development at Prometheus Research. Prometheus Research is a leading provider of integrated data management for research and the home of HTSQL, an open source navigational query language for RDMS.

Andrew has a Diploma in Computing and a BS in Software Engineering; he is currently studying a second BS in Creative Computing in his spare time.

He has over 10 years of experience working in the software industry in the UK, Canada, and USA. This experience includes e-Learning, CMS and LMS development, SCORM consultancy, web development in a variety of languages, open source application development, and running a blog dedicated to maker culture and home automation.

His interests include web development, e-Learning, 3D printing, Linux, the Raspberry Pi and Arduino, open source projects, parallel computing, home automation, amateur electronics, home networking, and software engineering.

Many of these topics were covered in his previous book from Packt Publishing, *Raspberry Pi Home Automation with Arduino*.

I would like to thank my wife Megen for supporting me throughout this project, my parents for their support with my interest in technology whilst growing up, and the team at Prometheus Research for making this a great and interesting place to work and helping to change the face of data management.

I would also like to thank Aboli Ambardekar, Amit Ghodake, and Edward Gordon at Packt Publishing for their guidance throughout this process, and the technical reviewers for their thoughtful comments.

About the Reviewers

Prasanna Gautam is an engineer who wears many different hats depending on the occasion. He graduated from Trinity College in 2011 with honors in Computer Science and Mathematics. At Trinity, he worked on building robots that extinguished fires in firefighting contests, implemented the JAUS communication protocol in LabView, and worked on architecting robots to work in realtime. He's worked on the Linux Network stack on phones, writing task distribution algorithms to be used on the Open Science Grid, and building Beowulf clusters ranging from 8 to 80 nodes.

Currently, he works as a Software Engineer at ESPN where he still gets to wear his hats. He and Andrew met at `NewHaven.io` and found they had the same idea with regard to teaching people about Parallel computing by getting them to set up their own clusters on Raspberry Pis. Fortunately, Andrew was already writing the book. In his free time, Prasanna attempts to play the guitar and make sense of music theory.

Sungjin Han loves to play games and tinker with Linux and Ruby. In this sense, the Raspberry Pi was an interesting toy and a powerful tool for him.

Thanks to all the people who make the world more convenient and happier, especially the ones on many open source projects.

Claes Jakobsson started his career in the mid-90s and quickly became involved in the open source community—hacking code and organizing stuff in his hometown of Stockholm. Although Perl is the primary focus, he forays into PostgreSQL, cURL, and other projects. His daytime occupation has been mostly with financial systems, but at night embedded systems, microcontrollers, virtual machines, and compilers keep his mind sharp. He is a technologist at heart with a sharing mind and is always eager to see what happens next.

Table of Contents

Preface

Have you ever read about parallel computing clusters and supercomputing, and wondered how to do it at home?

Do you have a number of Raspberry Pis and don't know what to do with them?

Then this is the book for you!

The field of parallel computing is certainly an exciting one. With the introduction of the Raspberry Pi, building a cluster at home is even easier. Hobbyists can now construct a small parallel computing cluster at low cost and using minimal physical space.

This book will walk you through building a parallel computing cluster using two Raspberry Pis and commodity off-the-shelf hardware.

Having set up your cluster, you will explore parallel computing paradigms such as MPI and MapReduce through exciting software projects.

Using MPICH and the C programming language, step-by-step guides will walk you through writing your own MPI-based applications. You will then test these in parallel on your two Raspberry Pis.

Following this, MapReduce will be examined through Apache Hadoop, which you will install and set up. You will then learn to interact with Hadoop by writing programs in Java.

Finally Raspberry Pi Super Cluster provides you with some fun jump-off points where you can explore the topics discussed in the book in further detail.

Having completed the various chapters' projects, you will have gained a basic knowledge of parallel computing and how it can be implemented on Raspberry Pi.

What this book covers

Chapter 1, Clusters, Parallel Computing, and Raspberry Pi – A Brief Background, provides an introduction to the topic of parallel computing and its history. You will also learn a little about the Raspberry Pi and why it is a good fit for experimenting with parallel computing.

Chapter 2, Setting Up your Raspberry Pi Software and Hardware for Parallel Computing, builds upon the first chapter by providing a guide to setting up a two node Raspberry Pi cluster and its associated hardware.

Chapter 3, Parallel Computing – MPI on the Raspberry Pi, introduces the topics of MPI (Message Passing Interface), and MPICH. These are explored through examples in the C programming language.

Chapter 4, Hadoop – Distributed Applications on the Raspberry Pi, explores Apache Hadoop and Java through practical examples. From installing Java through to Hadoop configuration, you will get a taste of the two technologies.

Chapter 5, MapReduce Applications with Hadoop and Java, explores the paradigm of MapReduce: the core technology at the heart of Hadoop.

Chapter 6, Calculate Pi with Hadoop and MPI, expands upon previous chapters with experiments on calculating Pi using Hadoop and MPICH. Here you will work with a Java example and write another C application implementing MPI.

Chapter 7, Going Further, finishes off the book with some projects ranging from building a Lego Raspberry Pi case to writing a Fortran application. You will also learn about some alternative approaches to powering your Raspberry Pi.

Appendix, provides you with a list of resources for further reading and exploration. Links to topics covered in this book are provided for the reader to follow up.

What you need for this book

The following list includes the recommended and optional hardware to complete the projects in this book:

- Two Raspberry Pi Model B's
- An HDMI monitor and cable
- USB keyboard
- USB mouse
- Two Micro-USB power units compatible with the Raspberry Pi

- Three network cables
- A small network switch
- Two Raspberry Pi compatible SD cards
- Internet connection
- A desk mounted power strip with both USB and mains outlet (optional)
- Raspberry Pi cases/project enclosures (optional)
- USB hard drive (optional for a project in *Chapter 7*, *Going Further*)
- Lego (optional)

Who this book is for

Have you ever wanted to build your own super computer? Wonder what parallel computing is all about and want to experiment with it? Have a bunch of Raspberry Pis and not sure what to do with them? Then this book is for you.

Aimed at the super computing novice and Raspberry Pi enthusiast alike, this is the perfect introductory text for those wishing to get their hands dirty building their own system.

While some programming experience is required, no prior knowledge of the technologies associated with parallel computing is assumed.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Navigate into `mpich3` and create the following two directories."

A block of code is set as follows:

```
/*
Hello RPI implemented using MPI
*/

#include <stdio.h>
#include <mpi.h>
```

Any command-line input or output is written as follows:

```
ssh pi@192.168.1.85 'sudo echo "raspberrypi2" | sudo tee /etc/hostname;
sudo shutdown -r now'
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Select your SD card drive from the **Device** dropdown on the right-hand side".

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Clusters, Parallel Computing, and Raspberry Pi – A Brief Background

The domain of parallel computing is an interesting one, but building a cluster for fun has often required the use of expensive or bulky off-the-shelf hardware, such as desktop PC's or implementing complex virtual machine setups.

So what is a cluster? This term will come up often in the following chapters and essentially means, in the context of this book, a group of separate devices networked together. Each device on this network is often referred to as a node.

Thanks to the Raspberry Pi's low cost and small physical footprint, building a cluster to explore parallel computing has become far cheaper and easier for users at home to implement. Not only does it allow you to explore the software side, but also the hardware as well.

While Raspberry Pis wouldn't be suitable for a fully-fledged production system, they provide a great tool for learning the technologies that professional clusters are built upon. For example, they allow you to work with industry standards, such as MPI and cutting edge open source projects such as Hadoop.

This chapter will provide you with a basic background to parallel computing and the technologies associated with it. It will also provide you with an introduction to using the Raspberry Pi.

A very short history of parallel computing

The basic assumption behind parallel computing is that a larger problem can be divided into smaller chunks, which can then be operated on separately at the same time.

Related to parallelism is the concept of **concurrency**, but the two terms should not be confused.

Parallelism can be thought of as simultaneous execution and concurrency as the composition of independent processes. You will encounter both of these approaches in this book.

You can find out more about the differences between the two at the following site:

`http://blog.golang.org/concurrency-is-not-parallelism`

Parallel computing and related concepts have been in use by capital-intensive industries, such as Aircraft design and Defense, since the late 1950's and early 1960's. With the cost of hardware having dropped rapidly over the past five decades and the birth of open source operating systems and applications; home enthusiasts, students, and small companies now have the ability to leverage these technologies for their own uses.

Traditionally parallel computing was found within **High Performance Computing** (**HPC**) architectures, those being systems categorized by high speed and density of calculations. The term you will probably be most familiar with in this context is, of course, supercomputers, which we shall look at next.

Supercomputers

The genesis of supercomputing can be found in the 1960's with a company called **Control Data Corporation** (**CDC**). *Seymour Cray* was an electrical engineer working for CDC who became known as the father of supercomputing due to his work on the **CDC 6600**, generally considered to be the first supercomputer. The CDC 6600 was the fastest computer in operation between 1964 and 1969.

In 1972 Cray left CDC and formed his own company, Cray Research. In 1975 Cray Research announced the **Cray-1** supercomputer. The Cray-1 would go on to be one of the most successful supercomputers in history and was still in use among some institutions until the late 1980's.

The 1980's also saw a number of other players enter the market including Intel via the Caltech Concurrent Computation project, which contained 64 Intel 8086/8087 CPU's and Thinking Machines Corporation's CM-1 Connection Machine.

This preceded an explosion in the 1990's with regards to the number of processors being included in supercomputing machines. It was in this decade, thanks to brute-force computing power that IBM infamously beat world chess master Garry Kasparov with the Deep Blue supercomputer.

The Deep Blue machine contained some 30 nodes each including IBM RS6000/SP parallel processors and numerous "chess chips".

By the 2000's the number of processors had blossomed to tens of thousands working in parallel. As of June 2013 the fastest supercomputer title was held by the Tianhe-2, which contains 3,120,000 cores and is capable of running at 33.86 petaflops per second.

Parallel computing is not just limited to the realm of supercomputing. Today we see these concepts present in multi-core and multiprocessor desktop machines. As well as single devices we also have clusters of independent devices, often containing a single core, that can be connected up to work together over a network.

Since multi-core machines can be found in consumer electronic shops all across the world we will look at these next.

Multi-core and multiprocessor machines

Machines packing multiple cores and processors are no longer just the domain of supercomputing. There is a good chance that your laptop or mobile phone contains more than one processing core, so how did we reach this point?

The mainstream adoption of parallel computing can be seen as a result of the cost of components dropping due to Moore's law. The essence of Moore's law is that the number of transistors in integrated circuits doubles roughly every 18 to 24 months.

This in turn has consistently pushed down the cost of hardware such as CPU's. As a result, manufacturers such as Dell and Apple have produced even faster machines for the home market that easily outperform the supercomputers of old that once took a room to house.

Computers such as the 2013 Mac Pro can contain up to twelve cores, that is a CPU that duplicates some of its key computational components twelve times. These cost a fraction of the price that the Cray-1 did at its launch.

Devices that contain multiple cores allow us to explore parallel-based programming on a single machine. One method that allows us to leverage multiple cores is threads.

Threads can be thought of as a sequence of instructions usually contained within a single lightweight process that the operating system can then schedule to run. From a programming perspective this could be a separate function that runs independently from the main core of the program.

Thanks to the ability to use threads in application development, by the 1990's a set of standards had come to dominate the area of shared memory multiprocessor devices, these were **POSIX Threads** (**Pthreads**) and **OpenMP**.

POSIX threads is a standardized C language interface specified in the IEEE POSIX 1003.1c standard for programming threads, that can be used to implement parallelism.

The other standard specified is OpenMP. To quote the OpenMP website, it can be described as:

> *OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs.*

`http://openmp.org/`

What this means in practice is that OpenMP is a standard that provides an API that helps to deal with problems, such as multi-threading and memory sharing. By including OpenMP in your project, you can write multithreaded applications without having to take care of many of the low-level implementation details as with writing an application purely using Pthreads.

Commodity hardware clusters

As with single devices with many CPU's, we also have groups of **commodity off the shelf** (**COTS**) computers, which can be networked together into a **Local Area Network** (**LAN**). These used to be commonly referred to as **Beowulf clusters**.

In the late 1990's, thanks to the drop in the cost of computer hardware, the implementation of Beowulf clusters became a popular topic, with Wired magazine publishing a how-to guide in 2000:

`http://www.wired.com/wired/archive/8.12/beowulf.html`

The Beowulf cluster has its origin in NASA in the early 1990's, with Beowulf being the name given to the concept of a **Network Of Workstations** (**NOW**) for scientific computing devised by *Donald J. Becker* and *Thomas Sterling*.

The implementation of commodity hardware clusters running technologies such as MPI lies behind the Raspberry Pi-based projects we will be building in this book.

Cloud computing

The next topic we will look at is cloud computing. You have probably heard the term before, as it is something of a buzzword at the moment.

At the core of the term is a set of technologies that are distributed, scalable, metered (as with utilities), can be run in parallel, and often contain virtual hardware. Virtual hardware is software that mimics the role of a real hardware device and can be programmed as if it were in fact a physical machine.

Examples of virtual machine software include VirtualBox, Red Hat Enterprise Virtualization, and **parallel virtual machine** (**PVM**). You can learn more about PVM here:

`http://www.csm.ornl.gov/pvm/`

Over the past decade, many large Internet-based companies have invested in cloud technologies, the most famous perhaps being Amazon. Having realized they were under utilizing a large proportion of their data centers, Amazon implemented a cloud computing-based architecture which eventually resulted in a platform open to the public known as **Amazon Web Services** (**AWS**).

Products such as Amazon's AWS **Elastic Compute Cloud** (**EC2**) have opened up cloud computing to small businesses and home consumers by allowing them to rent virtual computers to run their own applications and services. This is especially useful for those interested in building their own virtual computing clusters.

Due to the elasticity of cloud computing services such as EC2, it is easy to spool up many server instances and link these together to experiment with technologies such as Hadoop.

One area where cloud computing has become of particular use, especially when implementing Hadoop, is in the processing of **big data**.

Big data

The term big data has come to refer to data sets spanning terabytes or more. Often found in fields ranging from genomics to astrophysics, big data sets are difficult to work with and require huge amount of memory and computational power to query.

These data sets obviously need to be mined for information. Using parallel technologies such as MapReduce, as realized in Apache Hadoop, have provided a tool for dividing a large task such as this amongst multiple machines. Once divided, tasks are run to locate and compile the needed data.

Another Apache application is Hive, a data warehouse system for Hadoop that allows the use of a SQL-like language called HiveQL to query the stored data.

As more data is produced year-on-year by more computational devices ranging from sensors to cameras, the ability to handle large datasets and process them in parallel to speed up queries for data will become ever more important.

These big data problems have in-turn helped push the boundaries of parallel computing further as many companies have come into being with the purpose of helping to extract information from the sea of data that now exists.

Raspberry Pi and parallel computing

Having reviewed some of the key terms of High Performance Computing, it is now time to turn our attention to the Raspberry Pi and how and why we intend to implement many of the ideas explained so far.

This book assumes that you are familiar with the basics of the Raspberry Pi and how it works, and have a basic understanding of programming. Throughout this book when using the term Raspberry Pi, it will be in reference to the Model B version.

For those of you new to the device, we recommend reading a little more about it at the official Raspberry Pi home page:

`http://www.raspberrypi.org/`

Other topics covered in this book, such as Apache Hadoop, will also be accompanied with links to information that provides a more in-depth guide to the topic at hand.

Due to the Raspberry Pi's small size and low cost, it makes a good alternative to building a cluster in the cloud on Amazon, or similar providers which can be expensive or using desktop PC's.

The Raspberry Pi comes with a built-in Ethernet port, which allows you to connect it to a switch, router, or similar device. Multiple Raspberry Pi devices connected to a switch can then be formed into a cluster; this model will form the basis of our hardware configuration in the book.

Unlike your laptop or PC, which may contain more than one CPU, the Raspberry Pi contains just a single ARM processor; however, multiple Raspberry Pi's combined give us more CPU's to work with.

One benefit of the Raspberry Pi is that it also uses SD cards as secondary storage, which can easily be copied, allowing you to create an image of the Raspberry Pi's operating system and then clone it for re-use on multiple machines. When starting out with the Raspberry Pi this is a useful feature and something that will be covered in *Chapter 2*, *Setting Up your Raspberry Pi Software and Hardware for Parallel Computing*.

The Model B contains two USB ports allowing us to expand the device's storage capacity (and the speed of accessing the data) by using a USB hard drive instead of the SD card.

From the perspective of writing software, the Raspberry Pi can run various versions of the Linux operating system as well as other operating systems, such as FreeBSD and the software and tools associated with development on it. This allows us to implement the types of technology found in Beowulf clusters and other parallel systems. We shall provide an overview of these development tools next.

Programming languages and frameworks

A number of programming languages including Fortran, C/C++, and Java are available on the Raspberry Pi, including via the standard repositories. These can be used for writing parallel applications using implementations of MPI, Hadoop, and the other frameworks we discussed earlier in this chapter.

Fortran, C, and C++ have a long history with parallel computing and will all be examined to varying degrees throughout the book. We will also be installing Java in order to write Hadoop-based MapReduce applications.

Fortran, due to its early implementation on supercomputing projects is still popular today for parallel computing application development, as a large body of code that performs specific scientific calculations exists. In *Chapter 2*, *Setting Up your Raspberry Pi Software and Hardware for Parallel Computing,* we will provide brief instructions on installing it onto your Raspberry Pi and provide a further project in *Chapter 7*, *Going Further*.

In *Chapter 3, Parallel Computing - MPI on the Raspberry Pi*, we will install MPICH and run an example C application that comes bundled with the library, which will give you the opportunity of using the **Message Passing Interface** (**MPI**).

MPI is a language-independent message-passing communication method developed in the early 1990's to aid parallel computing application development. The topic of MPI will be covered in greater depth in *Chapter 3, Parallel Computing - MPI on the Raspberry Pi*, where we will test an application that calculates **π** using two Raspberry Pi devices.

In *Chapter 4, Hadoop – Distributed Applications on the Raspberry Pi*, we examine the Java programming language and Apache Hadoop in further detail. These form the final two important technologies we will cover in this book.

Apache Hadoop is an open source Java-based MapReduce framework designed for distributed parallel application development.

A MapReduce framework allows an application to take, for example, a number of data sets, divide them up, and mine each data set independently. This can take place on separate devices and then the results are combined into a single data set from which we finally extract a meaningful value.

In *Chapter 5, MapReduce Applications with Hadoop and Java*, we explain MapReduce in detail. The MapReduce model lends itself to being deployed on COTS clusters and cloud services such as EC2. In this book we will demonstrate how to set up Hadoop on two Raspberry Pis in order to mine for data and calculate π using a Monte Carlo Simulator.

Finally the *Appendix* of this book contains a number of links and resources that the reader may find of interest for Fortran, Java, C, and C++.

Summary

This concludes our short introduction to parallel computing and the tools we will be using on Raspberry Pi.

You should now have a basic idea of some of the terms related to parallel computing and why using the Raspberry Pi is a fun and cheap way to build your own computing cluster.

Our next task will be to set up our first Raspberry Pi, including installing its operating system. Once set up is complete, we can then clone its SD card and re-use it for future machines.

So grab your hardware as the next chapter will guide you through this process.

2
Setting Up your Raspberry Pi Software and Hardware for Parallel Computing

Now we are familiar with the concept of parallel computing and we need to set up our Raspberry Pi's hardware and software in order to work with MPI and Hadoop.

In this chapter we will start by discussing our work environment and hardware configuration. Following this we will install the necessary software onto the SD card and complete the basic configuration required.

Setting up our work environment

In order for our devices to communicate with one another, we need to set up the networking hardware and cable connections. We will also need to connect the necessary peripherals in order to interact with the Raspberry Pi's boot loading software.

Provided is a list of recommended and optional items you will need and following this we will discuss each of them in detail:

- Two Raspberry Pi Model B's, preferred with 512 MB RAM
- HDMI-capable or VGA/DVI monitor with appropriate adapter or adapter cable
- USB keyboard
- USB mouse
- Two 1A at 5 V Micro-USB power units

- A desk-mounted power strip with both USB and mains outlets (optional)
- Three Ethernet/RJ45 network cables
- A small network switch
- An existing Internet connection
- Two SD cards that are compatible with the Raspberry Pi
- Housing units for the Raspberry Pi boards (optional)
- USB hard drives (optional)
- Lego (optional)

HDMI-capable monitor or VGA/DVI monitor and adapter

The Raspberry Pi provides two easy options for connecting visual display units. This is via the RCA jack, which allows you to attach the Raspberry Pi to a TV and an HDMI port for connecting HD TV's and HD monitors to the Raspberry Pi.

When following this book we recommend using an HDMI or VGA/DVI monitor with appropriate cabling and adapters for simple ease of use. Being able to easily switch the plug on your monitor between your Raspberry Pi nodes will make debugging issues and setting up software easier.

USB keyboard and mouse

Most standard USB keyboards and mice should work with the Raspberry Pi. The eLinux website provides a list of peripherals that have been verified to work with your device. You can find this list at the following URL:

`http://elinux.org/RPi_VerifiedPeripherals`

Two micro-USB power units

When it comes to powering the Raspberry Pis, you will need two 1A at 5 V power units with micro-USB connectors or a USB hub and two micro USB cables. If you plan on extending your setup beyond three Raspberry Pi's you should consider a USB hub with multiple ports.

There have been a number of problems with power units not working with the Raspberry Pi. You should be aware that USB 2.0 specifies up to 500 mA per port, which is below what the Raspberry Pi can draw at peak (700 mA). This can result in problems especially with cheaper USB hubs.

Before purchasing any power units it is recommended that you review the details on the eLinux Wiki detailing power usage notes:

`http://elinux.org/RPi_VerifiedPeripherals#Power_Usage_Notes`

A desk-mounted power strip with both USB and mains outlets (optional)

Based upon the power usage notes on eLinux, you may wish to find hardware that mimics the following setup. If you are working on a desk having a power strip that contains at least two mains outlets and two USB ports should be easier to access. It will allow you to place your Pi's and monitor close to the power source and thus cut down on trailing wires and having to clamber around below a desk plugging in and unplugging cables.

A device similar to the one shown in the following image should be sufficient:

Three Ethernet/RJ45 network cables

In order to set up your parallel system, you will need three network cables. If you plan on adding more Raspberry Pi units to your setup in the future, you will need one additional cable per Pi.

One network cable will be connecting your switch to your modem and the other two cables will connect your Raspberry Pis to the switch.

A small network switch

For completing the projects in this book it is recommended you purchase a small network switch. A switch is a device that links multiple machines together on a network. For example, when using a switch, if it is connected to a router/modem, devices connected to the switch can share the Internet connection. When working with the Raspberry Pi you will only need a 10/100 desktop switch and although having a gigabit switch is nice, it is not necessary.

Many wireless routers provided by cable companies come equipped with several network ports allowing you to connect multiple Ethernet devices to them. While this is convenient, you may find it easier to isolate your Raspberry Pi so it is connected to a switch and then attach the switch to the router. This will likely give you more Ethernet ports, thus allowing you to expand your parallel computing setup. Also if your router is not located near your work area, cut down on the number of cables you will need to run and the length of cable you need to purchase.

An existing Internet connection

In order to download the operating system, Hadoop, and other software used in this book you will need an Internet connection.

Two SD cards that are compatible with the Raspberry Pi

The Raspberry Pi device uses a **Secure Digital** (**SD**) card in order to boot. The tutorials in this book will also be using the SD card as the Raspberry Pi's hard drive, although in the *Appendix* we do discuss incorporating a USB hard drive if you wish.

Therefore, you will need two Raspberry Pi-compatible SD cards in order to complete the examples in each chapter and an additional SD card for each further Raspberry Pi device you wish to add in the future.

A list of compatible SD card brands can be found at:

`http://elinux.org/RPi_SD_cards`

This subject is also covered later in this chapter.

Housing units for the Raspberry Pi boards and Lego (optional)

The Raspberry Pi usually comes shipped as just the board with the case being optional.

If you plan on using just two Raspberry Pis, a number of individual cases are available for purchase. If you intend on expanding the number of devices in your system, you may wish to consider a housing unit that allows for easy expansion.

The *Appendix* of this book provides a list of suppliers and housing units in either the singular form or designed for expansion for multiple devices. You will also find a brief guide to using Lego for building an expandable housing rack and a guide on the types of blocks you need.

While a housing unit is not required, providing one for your Raspberry Pis helps to cut down on the risk of them getting damaged.

USB hard drives (optional)

Once you have completed the examples in the book you may explore using USB hard drives in order to store your applications. If you do decide to use a USB hard drive you will need to have a powered USB hub in order to ensure it runs reliably.

SD cards have a limited (although large) number of write operations before the device starts to degrade. A USB hard drive is a more robust device, usually much faster than SD and will hold up better to many read/write cycles. The *Appendix* of this book discusses this subject in further detail.

Should you wish to try this out please refer to the eLinux verified peripherals list first:

`http://elinux.org/RPi_VerifiedPeripherals`

Future expansion and a scalable setup

The examples in this book are designed to be scalable. That is if you wish to add more Raspberry Pi's to your network, this should be very easy.

If you plan on running more than two devices you should therefore consider hardware that scales well. Switches for example come with as many as 48 ports.

One area you may find problematic is in providing power to multiple Raspberry Pi computers. Having a huge number of power units of USB hubs can be extremely cluttered. In *Chapter 7*, *Going Further*, we suggest several methods for providing power to multiple units.

With the preceding thoughts in mind, let's now complete the initial set up of our hardware.

Completing the initial setup

Each chapter in this book will guide you through when you need to connect each hardware device and install software as necessary.

There is however some setup you will need to perform initially:

- Start by connecting your switch to your router using a network cable and then connect the power unit to the switch to power it up.
- Using a PC or similar device with an Ethernet connection, plug it into the switch and test that you can access the network and Internet.
- Next select a single Raspberry Pi. This will be your Master unit.
- Connect the mouse, keyboard, and the monitor to the Raspberry Pi.
- Test that the Raspberry Pi is in easy reach of the switch by connecting them together via an Ethernet cable. Once you have confirmed this you can disconnect the Ethernet cable.
- If you are using an USB hub or two separate USB power units, plug these into your mains power and position them so they are within easy reach of the Raspberry Pi.
- Next check that the USB cables can easily plug into the Raspberry Pi's micro-USB port. Once you have tested this make sure to unplug the cable from the Raspberry Pi.

You may also wish to try the same with your second Raspberry Pi to confirm that there is enough room and cable slack for both devices to be set up side by side.

Once you are happy that everything is ready to go, we can move on to setting up our SD card.

Using an SD card as our Raspberry Pi's storage device

The Raspberry Pi comes equipped with a Secure Digital (SD) card port and to begin with we will run the operating system from an SD card. It is possible to use an USB hard drive as we discussed earlier, which are generally faster and in *Chapter 7*, *Going Further* we will discuss this in more detail. For the earlier chapters, however, the SD card is convenient as it is easy to quickly clone for multiple devices, takes up no extra desk space, and leaves the USB ports free for connecting a mouse and keyboard for debugging issue, if for example, you cannot login to the device via the network.

There is a range of SD cards available in the market in a variety of sizes. You will need to use an SD card of at least 2 GB.

You can find a guide to supported SD card brands and models at eLinux's Raspberry Pi Wiki:

`http://elinux.org/RPi_SD_cards`

It is also possible to purchase an SD card with a pre-installed operating system; however we recommend following the steps in this book. In the past there have been problems resulting from changed hardware on the Raspberry Pi that required new firmware images. This resulted in some pre-installed SD cards not booting. Also, following the steps yourself will help to familiarize you with the options available.

SD card setup

Before we can install an operating system, it is recommended that you format your SD card, especially if you are re-using one that has been used on another device.

You may have purchased two or more Raspberry Pis and SD cards in order to complete the projects in this book. To start with we recommend you format all cards that you plan on using even though you will be cloning one card to use on the others.

The SD card will need to be formatted for **FAT** (**File Allocation Table**), which is the format on Windows devices.

Once the formatting process is complete, you can then install a boot selection screen application. We will be using BerryBoot version 2.0 in this chapter. Also available for the Raspberry Pi is a popular boot loader called NOOBS. You can read more about it on the official Raspberry Pi website:

`http://www.raspberrypi.org/downloads`

If you wish you can download NOOBS instead and switch out the BerryBoot specific steps for those located in the NOOBS readme:

`https://github.com/raspberrypi/noobs/blob/master/README.md`

So let's start by walking through the SD card formatting process.

Formatting our card

In the following section we will guide you through SD formatting on Mac OS X, Windows 8, and Linux.

> Many SD cards come pre-formatted in FAT. Due to the robustness of FAT it is used by a variety of devices including digital cameras. However, we still recommend re-formatting the card to head off any potential installation problems. Also remember not to choose the **Quick format** option if it is presented.

If you find that your menu configuration is slightly different from those listed in the following sections, you can find many guides to card formatting on Google.

Mac OS X SD card formatting instructions

The following steps will guide you through the Mac OS X SD card formatting process. Start by placing the card into the SD card port on your Mac:

1. In **Finder** or from the task bar open your `Applications` folder.
2. Next select the **Utilities** folder icon.
3. From **Utilities** select **Disk Utility**.
4. When the **Disk Utility** window opens, on the left-hand side you will see a list containing **Disks**, **Volumes**, and **Disk Images**.
5. From this list located on the left-hand side select your SD card. If it is new it will likely have a label of **NO NAME** or similar.
6. The right-hand side panel will now update with the details about your card.
7. From the menu located at the top of the panel select the **Erase** tab.
8. You will now be presented with a set of instructions for formatting your SD card.
9. Locate the **Format** dropdown and select the **MS-DOS (FAT)** option.
10. Change the **Name** of your SD card to `RPIMASTER`.

11. Double check to ensure you have selected the correct options.
12. You can now select the **Erase** button which will format the card.

> Remember, when you format the card any data on it will be lost.

The formatting process is now complete and you can move onto the next step of installing BerryBoot Version 2 or NOOBS.

Windows 8 SD card formatting instructions

The following instructions will walk you through formatting your SD card under Windows 8.

Start by placing your SD card into the card port on your Windows machine:

1. Select the icon to take you to the Windows 8 desktop.
2. From here open **Explorer** from the task bar located at the bottom of the screen.
3. When Explorer loads, locate your SD card from the list of devices in the left-hand side **Explorer** panel.
4. Now right-click on your SD card.
5. From the pop-up menu left-click on the **Format...** option.
6. You will now be presented with the **Format Removable Disk** menu.
7. From the **File system** dropdown option select **FAT32 (Default)** if it is not already selected by default.
8. The other options should be left at their default.
9. Enter into the **Volume** label text entry field the new name for your SD card: `RPIMASTER`.
10. Select the **Quick Format** checkbox.
11. Click on the **Start** button.
12. You will now be presented with a warning that all the data on the disk will be erased.
13. Click on the **OK** option.

Windows 8 will format your SD card using the settings from the preceding steps and will erase any previous data on the card.

Once the process is complete and the card is successfully formatted Windows will display a popup with the text **Format Complete**.

Select **OK** to close the pop-up window. You can now move onto installing BerryBoot version 2 or NOOBS onto your SD card.

Linux instructions for SD card formatting

There are a number of tools available for partitioning and formatting disk and SD cards in Linux. We are going to use the mkdosfs application via the terminal window.

The mkdosfs program allows us to format the SD card to FAT, which BerryBoot version 2 requires.

Follow the steps below to format your card:

1. Load the **Terminal** if you are not already in a shell.
2. From the prompt run the command: `df -h`.
3. Depending on your Linux version you will see a list similar to the following:

```
Filesystem       Size   Used  Avail   Use%  Mounted on
/dev/sda2        110G   49G    62G    45%    /host
/dev/mmcblk0p1   7.3G   671M   6.3G   10%    /mnt
```

4. Start by making a note of the **Filesystem** name of your SD card and the **Mounted on directory**.
5. If you are not running as root, switch user to root: `su root`.
6. Next unmount the SD card in order to format it, for example:

```
umount /dev/mmcblk0p1
```

7. Once unmounted we can use mkdosfs to format the card.
8. From the prompt run the following command:

```
mkdosfs /dev/mmcblk0p1 -F32
```

9. This will format your SD card to FAT(32).
10. Once complete re-mount the SD card using the **Filesystem** name and **Mounted on location** you noted down, for example:

```
mount /dev/mmcblk0p1 /mnt
```

If you wish to re-label the SD card you can use an application such as mlabel located in the mtools package, the following steps illustrate how to do this:

1. The man page for mlabel can be found at: `http://linux.die.net/man/1/mlabel`.
2. Using your Linux version's package handler, install mlabel. For example, on Debian GNU/Linux distributions you can run:

   ```
   apt-get install mtools
   ```

3. Once again unmount the disk, for example:

   ```
   umount /dev/mmcblk0p1
   ```

4. Next check the current label on the SD card:

   ```
   mlabel -i /dev/mmcblk0p1 -s ::
   ```

5. For a new SD card you will probably see **Volume has no label**.
6. Now rename the label using the following command:

   ```
   mlabel -i /dev/mmcblk0p1 -s ::RPIMASTER
   ```

7. You should see the message again that was displayed when you checked the current label of the SD card.
8. Finally we can verify that our change took place. Re-mount the SD card and then run:

   ```
   sudo blkid
   ```

9. You should now see an output similar to the following:

   ```
   /dev/mmcblk0p1: SEC_TYPE="msdos" LABEL="RPIMASTER" UUID="0F68-
   87C5" TYPE="vfat"
   ```

The SD card is now formatted for FAT and ready to install BerryBoot version 2 or NOOBS.

BerryBoot version 2

Next we are going to download and install BerryBoot version 2 onto the formatted SD card. For those of you who chose to download NOOBS from the Raspberry Pi site, you can ignore this step and should follow the readme guide.

BerryBoot is a Mac, Windows, and Linux compatible universal operating system installer, also known as a boot loader. BerryBoot is packaged as a ZIP file, which when unzipped onto your formatted SD card will launch once the SD card is connected to the Raspberry Pi.

Once BerryBoot launches, it gives you the option of installing one or more Raspberry Pi-compatible operating systems onto the SD card.

Downloading the BerryBoot version 2 ZIP file

In order to install BerryBoot, you will need to download the latest version of the ZIP file from the BerryBoot website. This can be found at:

`http://www.berryterminal.com/doku.php/berryboot`

Locate the download hyperlink; this will probably have a date stamp on it. As of May 28, 2013 the file is approximately 30 MB.

Using an application that allows you to decompress ZIP files, unzip the contents and copy them to the SD card. If your machine does not already have an unzip application you can use one of the following.

Mac OS X

Mac OS X usually comes with an archiving/un-archiving tool built in. However, there are other applications you can choose from that can also be used, these include:

- Archiver: `http://archiverapp.com/`
- WinZip for Mac: `http://www.winzip.com/mac/`

Windows 8

Windows 8 offers the ability to ZIP files using the built in utility available under the **Share** menu in Explorer where you will find a ZIP icon. If you would prefer to try another utility (or are using an older version of Windows) the following are Windows graphical user interface based applications:

- WinZip: `http://www.winzip.com/`
- 7-zip: `http://www.7-zip.org/`

Linux

For Linux we recommend installing unzip.

- For Debian GNU/Linux versions:

  ```
  apt-get install unzip
  ```

- For Red Hat Linux, Fedora, and RPM compatible versions of Linux:

  ```
  yum install unzip
  ```

Remember to make sure the files located in the ZIP file are in the root of the SD card and not within a folder with the ZIP file's name.

After writing the BerryBoot version 2 files to the SD card we can then connect one of our Raspberry Pi in order to complete the next steps.

Starting up the Raspberry Pi

You are now ready to start up the Raspberry Pi and start using BerryBoot version 2.

The following steps will guide you through powering up your Raspberry Pi safely:

1. Start by ejecting your SD card from your computer and place it into the SD card port of the Raspberry Pi you prepared for use earlier in this chapter.
2. Plug in the mouse and keyboard to the USB ports. You will need these in order to complete the operating installation process.
3. Hook up the monitor to the HDMI port.
4. Connect the Raspberry Pi to the switch you set up earlier in this chapter.
5. Now power up your Raspberry Pi by connecting the micro-USB power unit to it.

If you are using BerryBoot, displayed on your monitor will be the Welcome screen. This is the first step in setting up our operating system and is also a confirmation that we copied the boot loader onto the SD card successfully.

BerryBoot version 2 and NOOBS provide us with a number of Linux versions to choose from. For setting up our parallel-based computing system we will be using Raspbian. Users already familiar with Raspbian may skip this section.

Raspbian is a flavor of the Linux operating system based upon Debian Wheezy and is specially optimized and compiled for the Raspberry Pi.

You can read more about it at:

`http://www.raspbian.org/`

The Linux.org website also provides some useful tutorials if you come across topics you are unfamiliar with:

`http://www.linux.org/`

Once you are comfortable with the contents of this book, you may be interested in installing another variety of operating system (OS) to try with your Raspberry Pi. One of the benefits of having used a boot loader is that you can easily choose another OS without having to re-format your SD card and start from scratch.

The following steps will now guide you through installing Raspbian via Berryboot. Once again you can refer to the NOOBS readme if you have decided to install this instead:

`https://github.com/raspberrypi/noobs/blob/master/README.md`

The installation process

The installation process is as given.

1. From the **Welcome** pop-up screen select the following options:
    1. If you see thin green borders at the top and bottom of your monitor select the radio button titled **Yes** (disable overscan).
    2. Since your Raspberry Pi is connected to your switch, from the **Network connection** option select the **Wired radio** button.
    3. From the **Locale** settings choose your **Timezone** and **Keyboard** layout. You can also test whether your keyboard works by typing into the **Type here to test keyboard** field.
    4. Finally select the **OK** button to move onto the next screen.
2. Next you will be taken to the **Disk selection** pop-up screen. Here you will choose the storage device you want to install Raspbian on.
3. Select your SD card and then from the **File system** list choose the option **ext 4 (no trim/no discard)**. This file system is geared towards Linux-based operating system.
4. Next select the **Format** button located at the bottom of the pop-up screen.
5. The formatting process may take a minute and once completed will present you with the **Add OS** screen from which Raspbian can be selected.
6. Select the **Debian Wheezy Raspbian** option, you will notice that it contains a version number, file size, and a message indicating it is the official Raspbian version.
7. Next select the **OK** button.
8. This will kick off the download, which is approximately 500 MB and based upon the speed of your Internet connection may take a few minutes. You should now see the pop-up screen indicating that Raspbian is downloading.

9. Once the download has completed the BerryBoot menu editor will be displayed.
10. You will now see your Raspbian download followed by the version number.
11. Located at the top of the **BerryBoot** menu editor screen is a menu providing a number of configuration options. These are:
    - **Add OS**
    - **Edit**
    - **Recover**
    - **Backup**
    - **Delete**
    - **Set default**
    - **Exit**
    - And a **>>** icon which will load the advanced settings menu.
12. Select your Raspbian installation and click on the **Set default** option. This will now result in your downloaded operating system loading by default when the Raspberry Pi is powered up.
13. Next select the **Exit** option.

After seeing a back splash, the BerryBoot version 2 boot menu will display and after 10 seconds will load the **Raspberry Pi Software Configuration Tool** (**raspi-config**).

The **raspi-config** screen is a menu that contains setup options for your Raspberry Pi. You can navigate the screen using your keyboard arrow keys and use the *Enter* key to select an option.

The setup options menu contains the following:

- **Expand Filesystem**: Ensure that all of the SD card storage is available to the OS
- **Change User Password**: Change password for the default user (pi)
- **Enable Boot to Desktop**: Choose whether to boot into a desktop environment or the command line
- **Internationalisation Options**: Set up language and regional settings to match your location
- **Enable Camera**: Enable this Pi to work with the Raspberry Pi camera
- **Add to Rastrack**: Add this Pi to the online Raspberry Pi Map (Rastrack)
- **Overclock**: Configure overclocking for your Pi

- **Advanced Options**: Configure advanced settings
- **About raspi-config**: Information about this configuration tool
- **<Select>**
- **<Finish>**

From the preceding menu we are going to change the password, select the **Advanced Options**, and disable the desktop environment on loading.

Our first task is to change the default password on the Raspberry Pi to something more secure. Select the **Change User Password** option and follow the steps to enter and re-enter your new password.

By default the password for the Raspberry Pi is set to `raspberry`.

Next we need to modify the boot settings so we boot into the Raspbian command line rather than the desktop environment:

1. From the menu select the **Enable Boot to Desktop** option.
2. You will now see a message: **Should we boot straight to desktop?**
3. From the two options presented select **No**.
4. You will then be returned to the main menu.
5. Finally we need to configure the RPi to allow external connection to it via SSH. You can read more about SSH at Linux.org: `http://www.linux.org/threads/openssh.4162/#post-10380`
6. From the menu select **Advanced Options**. The advanced menu screen will now be loaded containing the following options:
    - **A1 Overscan**: You may need to configure overscan if black bars are present on display
    - **A2 Hostname**: Set the visible name for this Pi on a network
    - **A3 Memory Split**: Change the amount of memory made available to the GPU
    - **A4 SSH**: Enable/disable remote command-line access to your Pi using SSH
    - **A5 Update**: Update this tool to the latest version
7. From the presented menu select the **SSH** option.

8. The following message will be displayed: **Would you like the SSH server enabled or disabled?**.
9. This should be enabled by default. If not from the two options available on the screen select **Enable**.

When your Raspberry Pi starts up in future, SSH will now be enabled allowing you to connect to the command line via an external device, for example, the Mac OS X terminal.

You may have noticed in the preceding menu the option to change the hostname of your device. We are interested in updating this; however in *Chapter 3*, *Parallel Computing - MPI on the Raspberry Pi*, we will explore a technique for updating the hostname remotely. If you wish to change it now, though, select this option and complete the steps presented. Leaving the hostname as is will label your Raspberry Pi: raspberrypi.

Our configuration is now complete; navigate to the **Finish** option and press the *Enter* key to continue. You will now be presented with a message asking if you wish to reboot the Raspberry Pi. From the options presented select **Yes**.

Installation complete

Your Raspberry Pi will reboot itself and the login prompt will appear. For your username use `pi` and try entering the new password you set in the Raspberry Pi Software Configuration Tool (raspi-config) menu.

Once you have successfully logged in you should see the command prompt allowing you to run other Linux commands.

In order to ensure that your Raspberry Pi has access to all of the latest packages, you can now run:

```
sudo apt-get update
```

This will update the packages list in Raspbian.

Testing SSH and setting up keys

Now that Raspbian is installed and we can login successfully, we next need to test that SSH is running correctly on the machine and following this generate some keys for allowing devices to securely connect to the Raspberry Pi without needing a password.

Connecting via SSH

In order to connect to your Raspberry Pi remotely, you will need a second machine running a command-line utility and the IP address of your freshly set up Raspberry Pi.

We will start by obtaining the IP address.

From the command line on your Raspberry Pi type the following:

```
ip addr show eth0
```

You can now find your IP address after the word `inet`. For example:

```
Inet 192.168.1.84/24 brd 192.168.1.255 scope global eth0
```

You will need the number before the `/` that reads `192.168.1.84` in our preceding example.

Once you have the IP address you can try connecting to it from your other machine.

Mac OS X and Linux users can use the command-line utility that comes shipped with their operating system (for Mac users this in Terminal). Windows users can download an application called `PuTTY` that will allow them to SSH into another machine.

The `PuTTY` executable file can be obtained from the following site:

```
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
```

Once downloaded follow the steps under *Windows 8 users with PuTTY*.

Mac OS X and Linux users

Start by loading your Terminal/command-line application. Next run the following command:

```
ssh pi@192.168.1.84
```

Replace the IP address in the preceding line with the IP address you obtained from your Raspberry Pi.

You will be asked to enter the password you set earlier and may see a message in your Terminal suggesting that the authenticity of the host can't be established, for example:

```
The authenticity of host '192.168.1.84 (192.168.1.84)' can't be
established.
```

```
RSA key fingerprint is f6:4a:38:4a:8b:c6:04:a9:bc:51:c3:af:fe:cb:78:e6.
Are you sure you want to continue connecting (yes/no)?
```

You can type `yes` and press the *Enter* key.

Next you will see:

```
Warning: Permanently added '192.168.1.84' (RSA) to the list of
known hosts.
```

And will now see the command line for your Raspberry Pi. This indicates that, when the Raspberry Pi rebooted, SSH started up and allowed you to connect to the device.

Windows 8 users with PuTTY

Windows users should use the following steps to run PuTTY and connect to their Raspberry Pi remotely.

1. Make sure you have copied PuTTY to a place where it is easy to access in future.
2. Next double-click on the `putty.exe` file to open the PuTTY configuration screen.
3. In the **Host Name** (or **IP address**) field add the IP address of the Raspberry Pi.
4. In the **Port** field enter `22`; this is the port SSH runs on, and under **Connection** type select `SSH`.
5. Finally you can click on **Open**.
6. You may see a pop-up box with the title **PuTTY Security Alert** and a note explaining that the server's host key is not cached in the registry.
7. If this appears then select the **Yes** button.
8. In the terminal window you will now see the message:
   ```
   Login as:
   ```
9. Enter `pi`, which is your Raspberry Pi username.
10. In PuTTY you will see a prompt asking for the password, for example:
    ```
    pi@192.168.1.84's password:
    ```
11. Enter the password you set earlier and press the *Enter* key.
12. You will now be logged into the Raspberry Pi, indicating that the SSH server is up and running successfully and accepting outside connections.

SSH running successfully

You can now access your Raspberry Pi from your home network without having to be physically wired up to the device. This will allow you in future to administer Raspbian from the second machine, for example, or allow other automated processes running on different machines to access your Raspberry Pi.

[You may have noticed on your home router that the device now appears as raspberrypi, if you did not update the hostname via BerryBoot. If you wish to change the name of the device now, you can update the `/etc/hosts` and `/etc/hostname` files.]

Now that we can connect to the Raspberry Pi via SSH using a username and password, we will explore setting up RSA keys for SSH; these allow devices to connect to one another without a password. This feature will be important in later chapters when our Raspberry Pi's wish to communicate with each other without human input in order to send data.

Setting up your SSH RSA keys

RSA (**Rivest, Shamir, and Adleman**) is a public-key cryptographic algorithm. It works by generating two keys - a public key and a private key-through a number of operations involving prime numbers.

The public key can be shared with anyone and is placed onto a machine that a user wishes to connect to in the `authorized_keys` file, allowing password-free connections.

The second key, the private key, is kept secure by the user usually on their local machine in the `.ssh` directory.

When a user wishes to connect to the server, the private key generates a signature that can only be authenticated by the server if their public key is present. Thus the login to a machine can be restricted only to that user whose public key is in the `authorized_keys` list, and whose machine contains their private key.

For more information on the RSA algorithm the original white paper can be found on the MIT website:

`http://people.csail.mit.edu/rivest/Rsapaper.pdf`

Now that we have explored a little of the background of RSA keys, let's start by setting them up via the command line:

1. You can either connect to your Raspberry Pi remotely via SSH or access the Raspberry Pi's command line directly via the keyboard.
2. Once logged in you should be located in the home directory of your user `pi`.

[You can navigate to the home directory at any time by typing `cd ~`]

3. From inside this directory type the following command:

   ```
   ssh-keygen -t rsa -C "pi@raspberrypi"
   ```

4. You will then be prompted for the file to store your SSH key in:

   ```
   Generating public/private rsa key pair.
   Enter file in which to save the key (/home/pi/.ssh/id_rsa):
   ```

5. You do not need to type anything in here and can press the *Enter* key.
6. You will then be prompted for a pass phrase for the key:

   ```
   Created directory '/home/pi/.ssh'.
   Enter passphrase (empty for no passphrase):
   ```

7. You have the option here of leaving `passphrase` blank. This is a less secure option, but will also mean that if you reboot your Raspberry Pi or close your shell you will not need to re-enter the passphrase and use the ssh-agent and ssh-add tools.
8. If you choose to enter a non-blank passphrase you will also need to to follow the steps in the section, *The ssh-agent and ssh-add tools*.
9. So enter a new passphrase; or leave it blank and press the *Enter* key.
10. Following this you will be asked to re-type the phrase:

    ```
    Enter same passphrase again:
    ```

11. Once this has been completed you will see a confirmation message and a unique ASCII art image that has been generated.
12. Next we need to add our public key to the authorized key list on our Raspberry Pi. This will allow access to localhost by applications such as Hadoop running on your Master Pi.
13. You can do this by typing the following command:

    ```
    cat /home/pi/.ssh/id_rsa.pub >> /home/pi/.ssh/authorized_keys
    ```

The ssh-agent and ssh-add tools

If you decide to add a passphrase to your key, you will need to complete the following steps:

1. From the command line type:
   ```
   ssh-agent bash
   ssh-add
   ```
2. You will be prompted to enter your passphrase. Your bash shell will now be able to access the key without the passphrase prompt appearing. However, if you reboot your Raspberry Pi or close the bash shell, then you will have to repeat the preceding step.

In order to limit the issue of entering the passphrase to only needing to enter it when you reboot the Raspberry Pi, which should hopefully be infrequent, you should consider using the screen application available in Linux.

Screen is a terminal multiplexer and you can read more about it at:

`http://linux.die.net/man/1/screen`

Later in this chapter we will cover how to install and configure screen if you wish to use it.

SSH setup complete

The SSH key for your Raspberry Pi is set up. Going forward in this book this Raspberry Pi will act as your Master device.

In *Chapter 3*, *Parallel Computing - MPI on the Raspberry Pi*, you will learn how to take the public key you generated and copy it to the other Raspberry Pi. This will then allow your Master device to access the other device without you being prompted for a password.

Wrapping up

We now have our operating system installed and our RSA key set up. The following guides provide some optional tools and compilers you can now install onto Raspbian, including a guide to install Fortran.

Editing text files on Raspbian

While editing and running code on your Raspberry Pi, having a good code and text-editing tool is very useful. By default the Raspberry Pi comes installed with a text-editor called **nano**. You can read more about nano at:

```
http://www.nano-editor.org/
```

Vim is also a text editing tool—especially good for programming—with a wide variety of functions and features, including the ability to run shell commands while you are editing a file.

For those interested in checking out Vim, you can read more at:

```
http://www.vim.org/about.php
```

To install Vim from the terminal line run the following command:

```
sudo apt-get install vim
```

Throughout this book we will be using Vim to edit files; however, feel free to use nano or whichever tool you are most comfortable with.

Installing Fortran

In *Chapter 1, Clusters, Parallel Computing, and Raspberry Pi – A Brief Background*, we introduced you to Fortran. At this point we recommend installing Fortran, from the command line run the following:

```
sudo apt-get install gfortran
```

This will install the GCC Fortran compiler. The compiler supports Fortran 95, 2003, and 2008. You can read more about it at the GNU Fortran compiler site:

```
http://gcc.gnu.org/wiki/GFortran
```

Installing Fortran will make the installation of MPICH in the next chapter slightly easier. This is the application you will also be using for compiling your applications if you wish to try the project in *Chapter 7, Going Further*.

You may also at this point want to create a separate directory on your Raspberry Pi for storing Fortran applications. Using the `mkdir` command create the following folder in your home directory:

```
mkdir /home/pi/fortran
```

Terminal multiplexing with Screen

A Terminal multiplexer is an application we can install onto Linux that allows us to create several virtual terminal windows under a single shell. For the projects in this book we recommend using Screen. Screen is a GNU terminal multiplexing application that can be used to create terminal sessions that will stay active even after we log out of our Raspberry Pi.

This is useful, for example, when we use commands such as `ssh-agent` and `ssh-add` which will be touched upon in the following section.

Screen can be installed via `apt-get`; from the command line run the following:

```
sudo apt-get install screen
```

After installing Screen, we now need to create a configuration file that will style how Screen looks when we launch it:

Start by navigating to the root of your home directory:

```
cd /home/pi
```

Next we need to create the configuration file called `.screenrc`:

```
touch .screenrc
```

The touch command can be used to create an empty file without opening it directly into your text editor of choice. You will find this command useful in future if you wish to create some placeholder files, but are not ready to edit them yet.

Now using your text editor open up the `.screenrc` file and add the following configuration:

```
vbell off
vbell_msg ""
hardstatus on
hardstatus alwayslastline
#Use: info screen "String Escapes" to style your screen
hardstatus string "%{B}%-Lw%{r}%50>%n%f*%t%{-}%+Lw%<"
def monitor on
term screen-256color
shelltitle Window
screen bash
```

Now save the file and exit.

The set of commands you added to the file will provide you with a starting point for customizing your screen's look and feel.

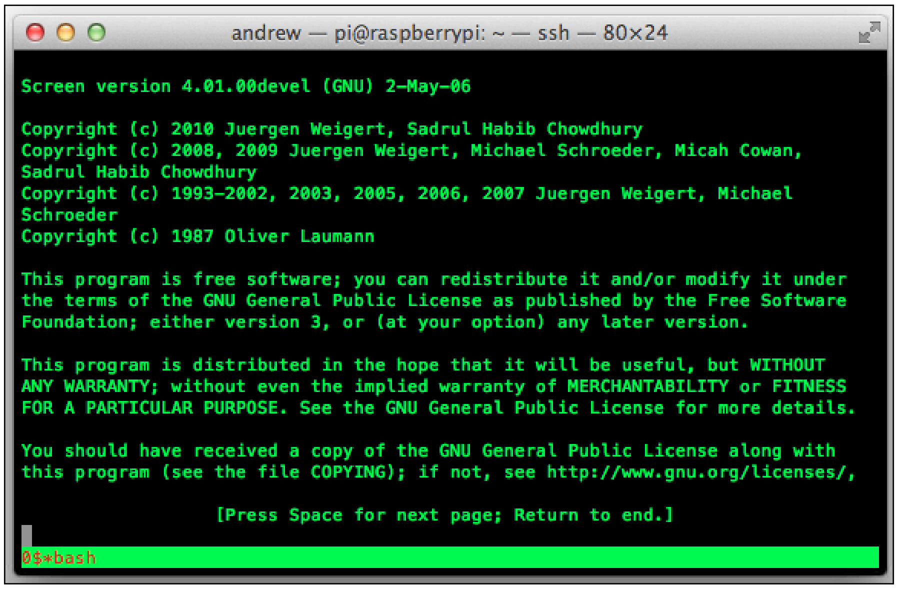
You can read more on how to configure screen by running the following command:

```
man screen
```

We can now test this screen configuration by launching the application. From the command line type:

```
screen
```

You will now see the Screen welcome message as shown in the following screenshot, you can press *Enter* to exit this:



```
Screen version 4.01.00devel (GNU) 2-May-06

Copyright (c) 2010 Juergen Weigert, Sadrul Habib Chowdhury
Copyright (c) 2008, 2009 Juergen Weigert, Michael Schroeder, Micah Cowan,
Sadrul Habib Chowdhury
Copyright (c) 1993-2002, 2003, 2005, 2006, 2007 Juergen Weigert, Michael
Schroeder
Copyright (c) 1987 Oliver Laumann

This program is free software; you can redistribute it and/or modify it under
the terms of the GNU General Public License as published by the Free Software
Foundation; either version 3, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT
ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with
this program (see the file COPYING); if not, see http://www.gnu.org/licenses/,

                  [Press Space for next page; Return to end.]

0$*bash
```

You will also see that the Screen's title is **bash**, we are going to rename this. Use the following key combination to bring up the rename text field: *Ctrl A* + *Shift A*.

Name this window `My first screen` and press the *Enter* key.

You will see the name has now been changed.

Next we will create a second window. From the command line type the following key combination:

```
ctrl a + c
```

This will create a second window, with the title **1$*Window**. Once again we are going to change the screen's name. Using the preceding command, rename the window: `My second screen`.

We now have two windows. To navigate between them use: *Ctrl A* + <*num*>

You should replace <num> in the command with the screen number, for example `0` or `1`.

Now we can switch between the screens. Sometimes we want to exit the Screen application, but leave it running in the background. This can be achieved by using the detach command. The detach command is:

```
ctrl a + d
```

You will now see that you have dropped back into the original shell that you launched Screen from.

To reconnect to an existing Screen session, you can type:

```
screen –r
```

Here we have covered the basics of Screen. You will find this tool invaluable when running multiple applications in Linux and wish to log in and out of the machine without the various processes you are running being terminated.

If you decided to add a passphrase to your RSA key, you may at this point want to run the `ssh-agent` and `ssh-add` commands:

```
ssh-agent bash
ssh-add
```

This will start the authentication agent in the Screen's shell session and add the RSA identity to it.

Now we have completed setting up the software on our operating system; let's review what we have accomplished in this chapter.

Summary

In this chapter we set up our work area and hardware for building our parallel computing system using Raspberry Pi. We then formatted our SD card and installed BerryBoot version 2 or NOOBS. This subsequently allowed us to set up our Master Raspberry Pi device. Finally we tested our SSH service running on the Raspberry Pi and created an SSH public and private key pair.

Next we will install MPICH, set up our second Raspberry Pi, test our SSH keys, and try out a parallel application that runs on both of our Raspberry Pi's.

3 Parallel Computing – MPI on the Raspberry Pi

In this chapter we will be investigating the technology known as MPICH. MPICH is an implementation of the Message Passing Interface standard which we briefly touched upon in *Chapter 1, Clusters, Parallel Computing, and Raspberry Pi – A Brief Background*.

So what subject area do we cover in relation to this technology?

First we will compare MPICH to an alternative implementation of MPI called OpenMPI. Following this we will install and then set up MPICH on our Raspberry Pi (RPi) and run a test application to check if it is working. After this we will clone our SD card and set up our second Raspberry Pi. This gives us the opportunity to execute a test application on two Raspberry Pis and see a calculation of Π being run in parallel.

Finally we will write some simple applications to demonstrate how MPI works.

MPI – Message Passing Interface

As we explained in *Chapter 1, Clusters, Parallel Computing, and Raspberry Pi – A Brief Background*, the Message Passing Interface is a language-independent message-passing communication protocol designed for parallel computing applications.

The standard's beginning can be found in the early 1990's when a number of academics and figures from industry combined their efforts to design a message passing system that would aid parallel computing application development.

The MPI standard defines a core set of routines that can be used by a programmer in order to distribute their application and handle passing back the results of the executed code seamlessly. In MPI's early days, C and Fortran were the languages most closely associated with it; however, Java and Python among others have also gone on to offer support. We will now touch upon two of the C and Fortran implementations.

MPI implementations – MPICH and OpenMPI

There are two prominent implementations of MPI that can be used on the Raspberry Pi. These are: OpenMPI and MPICH.

OpenMPI is an open source implementation of MPI maintained by a collection of industry and academic partners. It has been implemented on a number of the world's top 500 supercomputers including the Japanese K computer.

OpenMPI's origins can be found in several other projects including the University of Tennessee's FT-MPI project, Indiana University's LAM/MPI, University of Stuttgart's PCX-MPI, and LA-MPI from Los Alamos National Laboratory in the USA.

You can find out more about the technology at the official website:

`http://www.open-mpi.org/`

MPICH, originally standing for Message Passing Interface CHameleon, is an implementation of the MPI standard that supports C, C++, and Fortran applications. It was initially developed in the early 90's to provide feedback on implementation issues to the MPI forum.

The MPICH Wiki providing more background on the technology can be found at:

`http://wiki.mpich.org/mpich/index.php/Frequently_Asked_`
`Questions#General_Information`

So which should you choose?

When it comes to application speed between the two libraries there is some debate. Ultimately though, how you optimize your program and the hardware it runs on will make a big difference.

Unlike the newer OpenMPI, MPICH has been around longer and is extremely portable between systems. There are also extensive documentation and support options available online.

Due to its long-term use you will also find more binary applications that work with MPICH (if the source code is not made available) and should not run into as many compatibility issues. If you do decide to use OpenMPI the applications written in this book can be re-compiled using OpenMPI and still work.

For the projects in this chapter and others we will choose MPICH and walk you through the installation process.

Creating an environment and downloading MPICH

Before we install any software, we are going to create a number of directories under our account. These will be used for installing MPICH:

If you are not connected to your Master Raspberry Pi, log back in.

From inside your home directory you can then create the following folder:

```
mkdir mpich3
```

Navigate into `mpich3` and create the following two directories:

```
mkdir build install
```

The `mpich3` is the directory where we will be installing the MPICH software as well. Feel free to change the numeric value in the directory name to match the major version number of the MPICH software you are downloading.

Our next task is to grab the latest package from the MPICH downloads link: `http://www.mpich.org/downloads/`

You can use `wget` to perform this task, make sure the version number is the latest:

```
wget http://www.mpich.org/static/downloads/3.0.4/mpich-3.0.4.tar.gz
```

Once the `tar.gz` file has been downloaded we can unzip it into the `mpich3` directory:

```
tar xvfz mpich-3.0.4.tar.gz
```

You now have all the files you need to build MPICH, so finally navigate into the build directory you created. This folder should be under the directory you are currently in. For example:

```
cd build
```

Building and installing MPICH

With the MPICH code unzipped we can start the build and install process:

First we are going to set a configuration parameter so that MPICH installs into the `install` directory we created earlier.

Run the following command; `mpich-3.0.4` will be the name of the directory created when you unzipped the file earlier:

```
/home/pi/mpich3/mpich-3.0.4/configure -prefix=/home/pi/mpich3/install
```

If you chose not to install Fortran, you will need to ensure that you disable the Fortran option when installing MPICH. You can use the configure application located in your `mpich` installation directory to find out more:

```
/configure --help | more
```

You will now see a message saying something similar to:

```
Configuring MPICH version 3.0.4 with  '-prefix=/home/pi/mpich3/install'
```

Next let's run the Makefile, to do this type the following command:

```
make
```

This process can take a while so you may want to take a break and read the MPICH installation documents to provide you with more background on the process and configuration options. These can be found at the MPICH website in PDF format: `http://www.mpich.org/static/downloads/3.0.4/mpich-3.0.4-installguide.pdf`

Once `make` has completed, you will then need to run the following install command:

```
make install
```

The installation process can take some time to complete. While you wait for it, you may be interested in checking out the MPICH users guide, also in PDF format: `http://www.mpich.org/static/downloads/3.0.4/mpich-3.0.4-userguide.pdf`

After the installation has completed we need to make the `bin` directory available for your profile by including it in the `PATH` variable. You can run the following command to add it to `PATH`:

```
export PATH=$PATH:/home/pi/mpich3/install/bin
```

In order for this to persist between sessions when you log into and out of your RPi, you will need to add `PATH` to your profile. The Linux `.profile` file is located in the root of your home directory, for example:

```
/home/pi/.profile
```

Using your text editor open the file:

```
vim /home/pi/.profile
```

Now add the following code to the bottom of your `.profile` file:

```
# MPI
export PATH="$PATH:/home/pi/mpich3/install/bin"
```

When you log into Raspberry Pi you will be able to run MPICH from the command line without having to type the full path to the executable.

Configuring your Raspberry Pi to run with MPICH

We are now almost complete with our setup process. Raspbian now needs to be configured to work with MPICH:

Start by locating the IP address of the Raspberry Pi, if necessary use:

```
ip addr show eth0
```

Make a note of the IP address. You will need to add this to `pifile`.

The `pifile` is a list of each Raspberry Pi device on the network that you want to run MPI-based applications on. As you add more devices to your cluster you can update `pifile` with their IP address.

You can name `pifile` anything you like, in some online examples you may see it named `hostfile`, or in the MPICH documentation as `machinefile`. Since we will be updating the Linux hosts file of Raspbian, to change the name of the Raspberry Pi we have used the name `pifile` to avoid confusion.

Using your text editor create `pifile` in your user accounts directory:

```
vim /home/pi/pifile
```

Into this file add the IP address you noted earlier.

We can now run `mpiexec` and test that it works. Type in the following command:

```
mpiexec -f pifile hostname
```

If everything has been set up successfully you should now see

```
Output is:
raspberrypi
```

With the setup complete, let's look at testing our installation.

Testing our MPICH installation

We can now check whether MPI is working on our single Raspberry Pi by running the example program: Cpi that comes bundled with the MPICH install.

This application will calculate the value of Pi (Π). In *Chapter 6, Calculating Pi with Hadoop and Java*, we will show you how to write your own Java application that performs the same task using Hadoop.

From the command line run:

```
mpiexec -f pifile -n 2 ~/mpich3/build/examples/cpi
```

You should now see the following:

```
Process 0 of 2 is on raspberrypi
Process 1 of 2 is on raspberrypi
pi is approximately 3.1415926544231318, Error is 0.0000000008333387
wall clock time = 0.018371
```

This confirms that MPICH is working as expected; great work, let's build our second RPi.

Building our second Raspberry Pi

With the first Raspberry Pi up and running, demonstrating MPI and we need to set up our second machine. Rather than running through the BerryBoot installation process we discussed in *Chapter 2, Setting Up your Raspberry Pi Software and Hardware for Parallel Computing*, we can clone our Master machine's SD card and install the image on a second empty SD card.

1. Start by powering down your Raspberry Pi.
2. Next eject the SD card. At this point it is recommended you label it with a marker pen, this will help to cut down any confusion should the SD cards get mixed up.

3. Grab your second SD card and label this as well. Once you have completed that it is time to clone the Master SD card.
4. Place the SD card into the machine you will be using to clone it.

The following instructions are provided for Windows 8, Linux, and Mac OS X machines.

Windows 8

For cloning an SD card in Windows we recommend using a third-party tool called Win32 Disk Imager.

Start by downloading the ZIP file from the following location:

`http://sourceforge.net/projects/win32diskimager/files/latest/download`

Once the file has downloaded follow these steps:

1. Extract the ZIP to a location on your hard drive.
2. Launch the unzipped `Win32DiskImager.exe`.
3. Select your SD card drive from the **Device** dropdown on the right-hand side.
4. Click on the folder icon and choose a location and name for the `.img` file.
5. Click on the **Save** button to close the popup.
6. Click on **Read**, this will start the process of backing up the SD card.
7. Once complete, close the application and eject the SD card.
8. Insert your blank SD card for the Slave machine into the SD card drive.
9. Re-launch `Win32DiskImager.exe`.
10. Select the drive with your empty SD card located in it from the **Device** dropdown.
11. Click on the folder icon and locate the backup you created previously.
12. From the main screen now select **Write**, to kick off the write process.
13. Once complete close `Win32DiskImager.exe` and safely eject the SD card.

You can now insert the SD card into your second Raspberry Pi and follow the steps for powering up your second Raspberry Pi.

Mac OS X

The instructions for building a second RaspBerry Pi on Mac OS X are as follows:

Start by loading the terminal window on your Mac.

You may remember you can type the following command to see the file system:

```
df -h
```

This will show you a list of the mounted partitions.

We are interested in the name of the whole SD disk rather than the partition we created for Linux.

On Mac OS X there is an easy way to grab a device's, disk name. Follow the given steps to find the name of your disk:

1. Select the Apple menu from the top-left of the screen.
2. Next select **About This Mac** from the dropdown.
3. From the popup click the **More info...** button.
4. Another pop-up window will display.
5. From this window select the **System report...** option.
6. Click on **USB** or **Card Reader** depending on whether you connect your SD card via an external or built-in device.
7. Find your SD card in the upper-right panel of the window.
8. Once you have found it, click on it and then search for BSD name in the lower-right panel. The disk name will be in the following format: `diskn`, the `n` will be the disk number itself for example, `disk3`.
9. Make a note of this number, as you will need it for cloning.

We can now back up a copy of the disk to the OS and re-use as needed.

Start by unmounting the whole disk:

```
diskutil unmountDisk /dev/disk3
```

This will unmount all the volumes on your SD card.

Next run the following command. Here `disk3` should be the name you noted down earlier:

```
dd if=/dev/disk3 of=/Users/andrew/raspberrypi.img bs=1m
```

A copy of the disk will now be created. Once the command has finished running successfully, you will see a message similar to the following in the terminal window:

```
129024+0 records in
129024+0 records out
66060288 bytes transferred in 9.034737 secs (7311811 bytes/sec)
```

With a backup of the SD made, you can eject the disk safely from your Mac. We are now going to take our second SD card, and using the backup we just made, copy this image onto this disk.

Insert the second SD card it into your Mac. Follow the preceding steps to find the disk name. Once you have this, you are ready to copy the image over. The following command is essentially the reverse of what you performed earlier:

```
dd bs=1m if=/Users/andrew/raspberrypi.img of=/dev/rdisk3
```

When the process has finished copying the file over from your hard drive to the SD card you will see:

```
7580+0 records in
7580+0 records out
7948206080 bytes transferred in 3945.963361 secs (2014263 bytes/sec)
```

Eject the card from your Mac and place it into your second Raspberry Pi, this is the machine you designated as the Slave.

Linux

The process for cloning an SD card on Linux is fairly straightforward.

1. From the command line use the `fdisk` utility to grab a list of disks currently mounted to the OS:

   ```
   sudo fdisk -l
   ```

   You will now see an output similar to the following:

   ```
   Disk /dev/mmcblk0: 7948 MB, 7948206080 bytes
   4 heads, 16 sectors/track, 242560 cylinders, total 15523840
   sectors
   Units = sectors of 1 * 512 = 512 bytes
   Sector size (logical/physical): 512 bytes / 512 bytes
   I/O size (minimum/optimal): 512 bytes / 512 bytes
   ```

```
Disk identifier: 0x00000000

Device Boot      Start     End       Blocks    Id       System
/dev/mmcblk0p1   2048      131071    64512     eW95     FAT16 (LBA)
/dev/mmcblk0p2   131072    15523839  7696384   83       Linux
```

2. From the output presented, locate your disk and note the disk name.
3. You can unmount all of the partitions on this disk by using the wildcard (`*`) after the disk name, for example:

   ```
   umount /dev/mmcblk0*
   ```

4. Once you have completed this step we can the use `dd` to copy the SD card to the hard drive of our Linux machine. The following example illustrates this process:

   ```
   dd if=/dev/mmcblk0 of=/home/andrew/raspberrypi.img bs=1M
   ```

5. Once this is complete we have an image of the SD card we can re-use whenever needed.
6. Eject the SD card you have just copied from your machine and locate the second (Slave) SD card.
7. Insert this into the Linux machine and follow the preceding steps to locate the disk name and to also make sure all the partitions are unmounted.
8. Once you have completed this, you can copy the backup you made of the Master SD card onto the Slave one.
9. Run the following command to use `dd` again, this time to copy the image over:

   ```
   dd bs=1M if=/home/andrew/raspberrypi.img of=/dev/mmcblk0
   ```

10. The process of copying over the image of the Slave SD card can take a few minutes.
11. Once complete you will see a message similar to:

    ```
    7580+0 records in
    7580+0 records out
    7948206080 bytes transferred in 3945.963361 secs (2014263 bytes/
    sec)
    ```

This indicates that the content has been copied over successfully. You can now eject the SD card and insert it into your second (Slave) Raspberry Pi.

Powering up the second Raspberry Pi

Once you have inserted the SD card, attach the monitor, keyboard, and mouse and then power up the device using the micro-USB cable.

When it has loaded, you should see the BerryBoot boot screen with the operating system you installed in *Chapter 2, Setting Up your Raspberry Pi Software and Hardware forParallel Computing*, presented.

Select this option and click on **OK**. Following this the command-line prompt will be displayed asking you to log into Raspbian. Using the username and password you created for the first Raspberry Pi you can login.

If you are now presented with the shell prompt for your Raspberry Pi, you have successfully cloned the SD card and can move onto setting up the SSH key.

RSA key setup for SSH

We now need to set up SSH on our Slave Raspberry Pi so that we can log into it from the Master Raspberry Pi. In order to do this our RSA key needs to be added to the Slave Raspberry Pi's `authorized_keys` list. However, first we should clean up the Slave Raspberry Pi and remove the copy of the Master Pi's RSA keys.

Start off by making sure you are logged into the Master Raspberry Pi and then secure shelled into the Slave from it. For example:

```
ssh -A pi@192.168.1.85
```

We now need to remove the public and private key located on the Slave machine. These are the keys from the Master Raspberry Pi and were copied over when you cloned its SD card. Going forward you may wish to use the steps given earlier in this chapter to also clone your Slave SD card once you have removed the public and private key. You can then re-use this image without having to repeat the following tasks each time you add a new machine to the cluster.

The `id_rsa` and `id_rsa.pub` keys are also located in the `.ssh` folder of your user. It is this directory we will therefore need to remove them from.

Navigate to the `.ssh` folder for your user on the Slave machine, for example:

```
cd /home/pi/.ssh
```

Whenever you create RSA keys they should be stored here. Note that it is possible to have more than one public and private key if desired.

Listing the files in the directory should show the two keys from the Master RPi:

```
ls -al
```

We can now remove the two keys using the following command:

```
rm id_rsa id_rsa.pub
```

If you have been experimenting with SSH outside the steps presented in this book and created other key pairs, make sure you delete the correct ones!

Now we have removed the keys, let's exit this machine. Remember you can use the exit command to do this::

```
exit
```

For the Master RPi to be able to log into the Slave we need to add the public key to the `authorized_keys` file we mentioned previously.

When running remote commands you do not need to always directly SSH into the target machine. You can run commands in the following format that will login, execute the command you wish to run, and then return the prompt to you on the machine you are currently logged into.

This is demonstrated in the following command for copying over our public key:

```
cat ~/.ssh/id_rsa.pub | ssh pi@192.168.1.85 "cat >> .ssh/authorized_keys"
```

When you run this command you will be prompted with the authenticity of host message you should now be familiar with. Accept this.

Next will be the password prompt; enter the same password as Master Raspberry Pi.

Finally you will be prompted to enter the passphrase. Enter this and the command will copy the file into the `authorized_keys` file and finish executing.

As you may have noticed you did not have to login, type the command, and then type exit to log back out of the target computer. This is a useful method for running remote commands that can help to save time.

Using the same method, we can now update the hostname of the second Raspberry Pi and restart the machine.

From the command line execute the following. If you wish to name your Slave Raspberry Pi something other than `raspberrypi2`, you can change the value in the command:

```
ssh pi@192.168.1.85 'sudo echo "raspberrypi2" | sudo tee /etc/hostname;
sudo shutdown -r now'
```

Each time you add a new Raspberry Pi to your cluster you can use the preceding method to change the hostname rather than edit it via `raspi-config`.

We can now update the `pifile` file we created earlier and add the IP address of the Slave Raspberry Pi to it. On the Master machine, open the file and add the Slave IP address, for example:

```
192.168.1.85
```

Our final task is to test our two Raspberry Pis using the example application from earlier. Once again run this from the command line:

```
mpiexec -f pifile -n 2 ~/mpich3/build/examples/cpi
```

If everything worked successfully you should see:

```
Process 0 of 2 is on raspberrypi

Process 1 of 2 is on raspberrypi2

pi is approximately 3.1415926544231318, Error is 0.0000000008333387

wall clock time = 0.018672
```

With the setup complete we can now write our own application.

Writing an MPI-based application

The following application is a simple Hello World style program that will demonstrate some of the features of MPI.

A list of supported MPI-based functions and datatypes can be reviewed at:

```
http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf
```

You will need to log into the Master Raspberry Pi to start with.

On this machine create a new file in the code directory under `mpich3` called `hello_rpi.c`. For example:

```
vim /home/pi/mpich3/code/hello_rpi.c
```

To this file we are now going to add the following code:

```
/*
Hello RPI implemented using MPI
*/

#include <stdio.h>
#include <mpi.h>
```

Here we include the MPI-specific header directive: `mpi.h`, which will give us access to the MPI library.

Following this we can add the main function to the code. Copy and paste the following under the header directives:

```
int main(int argc, char *argv[])
{
  int rpi; // The raspberry pi node
  int totalrpi; // The total number of rpi's

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rpi);
  MPI_Comm_size(MPI_COMM_WORLD, &totalrpi);

  printf("This is Raspberry Pi %d of %d\n", (rpi+1),
  totalrpi);

  MPI_Finalize();
  return 0;
}
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at http://www.packtpub.com. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.

Inside the function we simply include a number of MPI-specific function calls. The call to `MPI_Init` function initializes the MPI environment.

Following this we include the `MPI_Comm_rank` and `MPI_Comm_size` function calls. The first call determines the rank of the calling process and the second function determines the size of the group associated with the communicator, in our case the number of Raspberry Pis. The values from these two functions are stored in the `rpi` and `totalrpi` integer variables.

Following these function calls we print out the Raspberry Pi currently executing the application followed by the total number of Raspberry Pis.

The application then calls `MPI_Finalize` to terminate the MPI environment and exits.

And that's it, a fairly simple program implementing MPI; a comprehensive API guide lists the available functions:

`http://www.mcs.anl.gov/research/projects/mpi/www/www3/`

Save your code and exit the file. We can now compile our program using `mpicc` from the command line:

```
mpicc -g -o hello_rpi hello_rpi.c
```

In order to use the application on both Raspberry Pis, you will need to copy the `hello_rpi` file you generated over to the Slave Raspberry Pi. You can do this using scp as illustrated:

```
scp hello_rpi pi@192.168.1.85:/home/pi/mpich3/code
```

Now that we have the compiled code on both machines, we can try executing our application. As in the previous example, we will use `mpiexec`. From inside the root of your user run the following command:

```
mpiexec -f pifile -n 2 ./mpich3/code/hello_rpi
```

You will now see:

```
This is Raspberry Pi 1 of 2
This is Raspberry Pi 2 of 2
```

Congratulations, you have written your first MPI-based application. Let's now look at the communication aspect of MPI in more detail.

MPI – point-to-point communication

Point-to-point communication is an import pattern of MPI as it governs all the methods that transmit a message between two separate MPI processes.

In this approach, one process or task acts as the sender and the other as the receiver. At the heart of this are several types of send and receive routines. Each routine is subtly different and can have an impact on the performance of your program.

First we will look at non-blocking and blocking routines before writing an application that implements the blocking method later in this chapter.

Non-blocking methods work by starting an operation and then continuing with their execution. This type of method does not wait for the communication across the network between the two processes to complete. As a result of this it can be used for a number of performance gains. Examples of this include overlapping communication and the computation and the execution of a mathematical operation.

On the other hand, blocking routines request that the MPI library start the operation when free. Thus the process has to wait to ensure that the message data has reached a certain state before it can continue to execute. This type of method is often considered **safer**.

"Safer" in this context is used to mean that:

- The data intended for the sender can be modified
- The data intended for the receiver can be read.

Next we will look at the **synchronous** send. Synchronous is used to imply one action after another. The synchronous send is a blocking method that communicates with the second process and then blocks until the application buffer sending the task is free. It also continues to block until the second process has started to receive the data sent to it.

Closely related to this is a buffered send. The buffer is essentially a region of memory used to temporarily store data while it is being moved from one area to another by a process.

The buffered send is similar to the synchronous send; however, it permits the programmer to assign buffer space for data storage until it is delivered to the second process. Its application can help to mitigate problems associated with buffer space allocation.

Finally worth mentioning is the combined send/receive. This approach works by sending a message and returning a "receive" acknowledgement before implementing a block. The blocking continues until the process buffer is free and until the target process buffer contains the received data envelope.

These terms should provide you with a little more understanding of how programs on the Raspberry Pis communicate with one another across the network.

You can read more about these methods at the following two sites:

https://computing.llnl.gov/tutorials/mpi/

http://mpitutorial.com/

Let's now move onto an example program that illustrates some of the preceding concepts using a blocking send.

Using your text editor create a file called `blocking_send.c` in the same directory as your `hello_rpi.c` file.

To this add the following code:

```
#include <stdio.h>
#include <mpi.h>

int main(int argc,char *argv[])
{
  int rpi;
  int totalrpi;
  int targetrpi;
  int sourcerpi;
  int count;
  int tag=1;
  char inchar;
  char outchar='r';
```

In the preceding code we have defined our `main` function and declared a number of variables to store our results. Included in this is a variable of `char` type that is used to store the data to be sent between our two processes.

Now add the next block of code:

```
  MPI_Status Stat;
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rpi);
  MPI_Comm_size(MPI_COMM_WORLD, &totalrpi);
```

You should be familiar with MPI calls. Three of them were used in the `hello_rpi` application. Here we have also included `MPI_Status`, which is used to return data about a message by the receiving method.

After this add the following two `if` statements.

```
  if (rpi == 0) {
     targetrpi = 1;
     sourcerpi = 1;
     MPI_Send(&outchar, 1, MPI_CHAR, targetrpi, tag,
     MPI_COMM_WORLD);
     MPI_Recv(&inchar, 1, MPI_CHAR, sourcerpi, tag,
     MPI_COMM_WORLD, &Stat);
  }
```

```
    if (rpi == 1) {
       targetrpi = 0;
       sourcerpi = 0;
       MPI_Recv(&inchar, 1, MPI_CHAR, sourcerpi, tag,
       MPI_COMM_WORLD, &Stat);
       MPI_Send(&outchar, 1, MPI_CHAR, targetrpi, tag,
       MPI_COMM_WORLD);
    }
```

The `if` statements are responsible for executing the MPI calls that initiate communication between the two Raspberry Pis. This example would not be scalable for a large cluster, but serves the purpose of illustrating how the `MPI_Rec` and `MPI_Send` functions work with two devices.

In these MPI statements we are also telling the Master machine to send a character to the Slave and, once it receives it, to send it back.

We can complete this program with the following code:

```
    MPI_Get_count(&Stat, MPI_CHAR, &count);
    printf("Raspberry Pi %d: Received %d char(s) from Raspberry Pi
          %d with tag %d \n",
         rpi, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
    MPI_Finalize();
    return 0;
  }
```

Here `MPI_Get_count` returns the number of entries received. Following this we print out a statement to the screen indicating the Raspberry Pi node, the number of chars it received, the source node, and the tag. Finally we indicate that our program is finished.

Save the code you have added to your file and exit your text editor.

The next task is to compile the code using the steps from the `hello_rpi.c` application, calling this program `blocking_message`.

Once compiled copy it over to the Slave Raspberry Pi and run your application:

```
mpiexec -f pifile -n 2 ./mpich3/code/blocking_message
```

If successful you should now see:

```
Raspberry Pi 0: Received 1 char(s) from Raspberry Pi 1 with tag 1
Raspberry Pi 1: Received 1 char(s) from Raspberry Pi 0 with tag 1
```

You have now written and run an application that explores the point-to-point communication features of MPI and some of the technical terms associated with it.

If you are interested in pursuing MPI-based applications further, in *Chapter 7*, *Going Further*, we provide another example application and in the *Appendix* provide further links to MPI-based sites.

Summary

In this chapter, we explored the subject of MPI using the C programming language to give us a taste of parallel computing. We also wrote our first two parallel applications and set up a second Raspberry Pi node by cloning the Master Raspberry Pi's SD card.

In comparison to MPI we can look at a Java-based technology called Hadoop. In the next chapter we shall start this task.

4 Hadoop – Distributed Applications on the Raspberry Pi

In *Chapter 1, Clusters, Parallel Computing, and Raspberry Pi*, we touched upon the technology known as Apache Hadoop.

In this chapter, we will explore the subject in more detail. This will include setting up Hadoop in order to be able to write distributed applications on the Raspberry Pi in Java via the paradigm of MapReduce.

We will start with a brief introduction to Hadoop and then walk you through the installation and the configuration process for a test cluster.

A brief introduction to Apache Hadoop

The technology known as **Apache Hadoop** is an open-source framework for developing distributed applications hosted by the Apache Software Foundation. The framework contains a number of subprojects. The one we are interested in is the Hadoop Core, also known as Hadoop Common.

The Hadoop Common project is located within the overall Hadoop framework. It allows the development of cloud computing environments via off-the-shelf hardware such as the Raspberry Pi. The developer interacts with it by using its Java based API.

Within Hadoop Common there are several significant areas that help us achieve our goal of developing parallel computing applications. Two of the most important areas are as follows:

- Hadoop MapReduce environment
- Hadoop Distributed File System (HDFS)

In this chapter both subjects will be touched upon during the installation and setup process and *Chapter 5, MapReduce Applications with Hadoop and Java*, provides an in-depth look at the HDFS and MapReduce.

One of the perquisites for installing and using Hadoop is Java, so we will now install this programming language onto Raspbian.

Installing Java

Java, as you may know, is an OOP language that has its syntactical roots in the C and C++ programming languages. It is also the language we will be using to interact with the Hadoop framework.

We will start by installing Java onto our Master Raspberry Pi. Make sure you have logged into this machine.

The latest version of the **Java Development Kit** (**JDK**) can be found on the Oracle website at the following URL:

`http://www.oracle.com/technetwork/java/javase/downloads/index.html`

You should note that some versions of Java (including hard-float versus soft-float) might not be compatible with your Raspberry Pi.

You can always refer to the *eLinux.org* RPi Java JDK installation guide for updates available at the following link:

`http://elinux.org/RPi_Java_JDK_Installation`

Remember you can update your Raspberry Pi's package list by running apt-get update.

To start the Java installation, run the following command:

```
sudo apt-get install openjdk-7-jdk
```

You will now see the process running in your terminal window.

Once it is completed you will have the JDK ready to go for Java development. The JDK includes important tools such as the **JRE** (**Java Run-time Environment**) and the Java compiler.

To test your installation, run the following command in the terminal to view Java's help content:

```
java -h
```

You can run the preceding command at any time in order to find out more information on the Java software suite.

With Java installed we can move on to setting up Apache Hadoop.

Installing Apache Hadoop

In order to install Hadoop, we will need to locate the `tar.gz` file that contains the most recent stable release from the Apache website.

Before downloading this file you should create a directory on your Raspberry Pi to place the file in and to store your Hadoop projects.

Under your `/home/pi` directory, create the `hadoop` folder using the following command:

```
mkdir hadoop
```

Next navigate into this directory using the `cd` command:

```
cd hadoop
```

Now that we have a place to store our code, we can grab the latest version of Hadoop at the following link:

```
http://www.apache.org/dyn/closer.cgi/hadoop/common
```

We will download the `tar.gz` file you selected from the download website using **wget.** The following command illustrates this process:

```
wget http://apache.osuosl.org/hadoop/common/hadoop-1.2.1/
  hadoop-1.2.1.tar.gz
```

Remember to replace the URL with the mirror you selected from the download page and the version number (in our example 1.2.1) with the one you have chosen.

Once the file has finished downloading you can extract its contents. From the command line type:

```
tar xvfz hadoop-1.2.1.tar.gz
```

You should now see a directory with Hadoop located in it, for example `hadoop-1.2.1`.

These very simple steps have downloaded and extracted a copy of Hadoop Common onto our Raspberry Pi's file system. However, before we can start using it we need to amend some of the configuration files that govern how it will work.

Hadoop configuration

The configuration settings for Hadoop are stored in two types of XML documents: read-only default configurations and site-specific configurations. You will learn more about both of these types of files over the next two chapters.

If you are interested, you can also read more on the Apache website at the following URL:

```
http://hadoop.apache.org/docs/stable/cluster_setup.html#Configuration
```

In addition to the Hadoop XML documents, you will also be editing a shell script. The shell scripts referenced in this book are generally used for tasks such as starting and stopping servers.

Before we can amend our configuration files, you will need to locate the open JDK directory where Java is installed on your Raspberry Pi. This path is required in order to update the user profile.

To do this, navigate to the following `jvm` (Java Virtual Machine) directory:

```
cd  /usr/lib/jvm
```

As the Raspberry Pi uses an ARM processor, we will need the ARM-specific version of Java. When you list the contents of the directory, you should expect to see:

```
java-7-openjdk-armhf
```

Make a note of this directory as you will need to edit your `.profile` file and include this information.

Navigate to your home directory and open the `.profile` file in your text editor of choice for example:

```
vim ~/.profile
```

At the bottom of the file we need to add some export statements and update our `PATH` variable. Paste the following statements at the end of the file:

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-armhf
export HADOOP_INSTALL=/home/pi/hadoop/hadoop-1.2.1
export PATH=$PATH:$HADOOP_INSTALL/bin
```

These export commands update our environment to include the `bin` directory of the version of Hadoop we installed in the `PATH` variable.

Save the file and exit. We can now reload our profile so that Raspbian picks up our changes using the `source` command as follows:

```
source .profile
```

To check if our profile successfully reloaded you can try the `hadoop version` command, for example:

```
hadoop version
```

The output in the terminal should be information on the Hadoop version you installed.

As you can see Hadoop is successfully extracted and accessible; therefore our next task is to create the directories that it uses for storing data. Start by navigating to the `hadoop` directory.

Hadoop needs to have a directory where is can write intermediate output from MapReduce tasks and temporary data for the **HDFS** (**Hadoop Distributed File System**) client. The root directory where this data is stored is named `tmp`.

Create a folder with the name `tmp` inside the `hadoop` directory and then navigate into it:

```
mkdir tmp
cd tmp
```

Within this directory you can create subdirectories; for example, for specific data types.

The next directory created will be the `hdfs` directory. Once it has been created, you should navigate to this:

```
mkdir hdfs
cd hdfs
```

Inside this directory we will need to create two more sub-directories, `data` and `name`. Use the following command to create both at once:

```
mkdir data name
```

The `name` directory will be used for storing the **file system image** (**fsimage**), something you will learn more about later in this book.

The `data` directory is used for HDFS block storage. HDFS is also covered in more detail in *Chapter 5*, *MapReduce Applications with Hadoop and Java*.

With these in place, there are several XML configuration files that have to be edited to include settings for our environment.

The files are stored in the `conf` directory under the version number of Hadoop you installed. In the following example you should change `hadoop-1.2.1` to the version on your machine:

```
cd ~/hadoop/hadoop-1.2.1/conf
```

This directory contains site-specific configuration files, such as `mapred-site.xml`, that we will shortly be editing.

Located in the `conf` directory you will also find a file named `core-site.xml` that we will be editing next. This file contains information to override the default settings for core Hadoop properties and is used to set specific information such as the location of the temporary directory used by the HDFS.

```
vim core-site.xml
```

We will now add the following XML configuration (in the `core-site.xml` file) between the `<configuration>` tags:

```
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/pi/hadoop/tmp</value>
  <description>Temporary directories root.</description>
</property>
```

The `property` tag contains the path to the `tmp` directory we created on our file system. Make sure you update the contents of the `<value>` tag to reflect your path structure if it is different from the example.

Below the temp directory property, paste the following XML. Here we set the IP address and the port of our Master Raspberry Pi in the `<value>` tags.

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://192.168.1.84:54310</value>
  <description>The default file systems name.
   Contains the Master Raspberry Pi's IP and the port number.
  </description>
</property>
```

That completes the edits to the `core-site.xml` file; save your changes and exit. With this step complete we can modify the `mapred-site.xml` file.

The `mapred-site.xml` file is responsible for overriding the default values for MapReduce, for example allowing us to set the IP address that the MapReduce job tracker runs on.

```
vim mapred-site.xml
```

To the `<configuration>` tag we can now add the paths to the `data` and `name` properties as follows:

```
<property>
  <name>mapred.job.tracker</name>
  <value>192.168.1.84:54311</value>
  <description>The Master Raspberry Pi's IP and port that the
  MapReduce job tracker runs on.</description>
</property>
```

The preceding code sets the Map Reduce Job Tracker (responsible for farming out MapReduce jobs) to run on port 54311 of our Raspberry Pi's IP address. Save this and exit.

Finally with your text editor open the `hdfs-site.xml` file. This file contains important information such as the paths to the `name` and `data` directories we created previously.

```
vim hdfs-site.xml
```

To the `<configuration>` tag we can now add the paths to the data and name properties as follows:

```
<property>
  <name>dfs.data.dir</name>
  <value>/home/pi/hadoop/hdfs/data/</value>
</property>
```

```
<property>
  <name>dfs.name.dir</name>
  <value>/home/pi/hadoop/hdfs/name/</value>
</property>
```

Save these changes and exit your text editor.

With our configuration edits complete, we can go on to modify the `hadoop-env.sh` shell script in our text editor, for example:

```
vim hadoop-env.sh
```

Once open, edit the following `export` command to the open JDK directory you noted earlier:

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-armhf/
```

Next you can format the file system. This process prepares the target directory to be used by Hadoop to store data produced by the MapReduce jobs you will be writing later in this book.

Make sure your user (pi) owns the directory `sudo chown pi:pi -R hadoop`.

The command to begin the formatting process is as follows:

```
./home/pi/hadoop/hadoop-1.2.1/bin/hadoop namenode –format
```

Accept any prompts you get when re-formatting. Once this process has finished running we can test the Hadoop server.

Testing our Hadoop server

Congratulations! You have installed Hadoop and configured it to run on your Raspberry Pi. It is now time to test the various Hadoop services.

Next you can execute the `start-all` shell script:

```
/home/pi/hadoop/hadoop-1.2.1/bin/start-all.sh
```

This will start up a number of processes on your Raspberry Pi related to Hadoop. In your terminal you will see output similar to the following:

```
starting namenode, logging to /home/pi/hadoop/
  hadoop-1.2.1/libexec/../logs/hadoop-pi-namenode-raspberrypi.out
localhost: starting datanode, logging to /home/pi/hadoop/
  hadoop-1.2.1/libexec/../logs/hadoop-pi-datanode-raspberrypi.out
localhost: starting secondarynamenode, logging
  to /home/pi/hadoop/hadoop-1.2.1/libexec/../logs/hadoop-
  pi-secondarynamenode-raspberrypi.out
starting jobtracker, logging to /home/pi/hadoop/hadoop-
  1.2.1/libexec/../logs/hadoop-pi-jobtracker-raspberrypi.out
localhost: starting tasktracker, logging to /home/pi/hadoop/
  hadoop-1.2.1/libexec/../logs/hadoop-pi-tasktracker-raspberrypi.out
```

This indicates that our setup was successful. To stop the processes, run the following shell script:

```
/home/pi/hadoop/hadoop-1.2.1/bin/stop-all.sh
```

We have one simple configuration task left on our Master Raspberry Pi before we consider the Slave machine. Update the following `masters` conf file to indicate this is the Master machine:

```
vim /home/pi/hadoop/hadoop-1.2.1/conf/masters
```

To this file add the IP address of this Raspberry Pi.

Next open the `/etc/hosts` file in your text editor and append the IP address and the name of the machine to it, for example:

```
   192.168.1.84 raspberrypi
```

We are now ready to start work on our second machine. Make sure this is plugged in and powered up.

Setting up our second Raspberry Pi

The process for setting up our second Raspberry Pi is very similar to the previous steps we explored, although a lot quicker as we can reuse some of our configuration files and edit them as needed.

To start with, we will grab the IP address of our Slave Raspberry Pi. There is a file on the Master Raspberry Pi we will update that will need to include this.

From within the `conf` directory on the master machine, open the `slaves` file in your text editor:

```
vim /home/pi/hadoop/hadoop-1.2.1/conf/slaves
```

To this file add both the Slave and Master machines IP addresses on a new line each as follows:

```
192.168.1.84
192.168.1.85
```

Save the file and exit.

That's the final portion of configuration we needed to update, to support our second Raspberry Pi on the Master device.

To save some time in the setup process, we can copy our `.profile` file from the Master Raspberry Pi to the Slave Raspberry Pi using the following command:

```
scp ~/.profile pi@192.168.1.85:.profile
```

We can also create the `hadoop` directory on the second machine and copy the tar.gz Hadoop file we downloaded before using the following two commands:

```
ssh 192.168.1.85 "mkdir hadoop"
```

```
scp /home/pi/hadoop/hadoop-1.2.1.tar.gz
  pi@192.168.1.85:/home/pi/hadoop/hadoop-1.2.1.tar.gz
```

Log directly into the second Raspberry Pi:

```
ssh pi@192.168.1.85
```

We now need to add the Master and Slave IP addresses and the device names to the `/etc/hosts` file. Open the `/etc/hosts` file in your text editor and append the following statements to it:

```
192.168.1.84 raspberrypi
192.168.1.85 raspberrypi2
```

Save this file and exit.

As with the Master Raspberry Pi we also need to install Java. As before you can run `apt-get` to grab the JDK and install it, using `apt-get`:

```
sudo apt-get install openjdk-7-jdk
```

When Java has finished installing we are going to extract the files from the ZIP file we copied to this machine from the Master Raspberry Pi using the `tar` command:

```
tar xvfz hadoop-1.2.1.tar.gz
```

Once complete, you can run `hadoop version` again to see if it has installed successfully.

If everything looks good then recreate the folder structure inside the `hadoop` directory as we did on the Master Raspberry Pi. This should consist of the `tmp`, `hdfs`, `hdfs/name`, and `hdfs/data` directories.

Rather than copying and pasting the XML into the various configuration files again, it is easier to copy them from the Master Raspberry Pi and then edit the values.

Navigate to the following `conf` directory:

```
cd /home/pi/hadoop/hadoop-1.2.1/conf
```

Next run the following `scp` commands:

```
scp pi@raspberrypi:/home/pi/hadoop/hadoop-1.2.1/conf/*site.xml
scp pi@raspberrypi:/home/pi/hadoop/hadoop-1.2.1/conf/hadoop-env.sh
```

These two commands will copy the three site files, `core-site.xml`, `mapred-site.xml`, `hdfs-site.xml`, and also the `hadoop-env.sh` file we added the JDK path to.

The `core-site.xml` file contains the IP address of the Master Raspberry Pi. This will need to be changed to the address of the Slave machine. Open up the `core-site.xml` file using the following command:

```
vim core-site.xml
```

In the configuration, change the `<value>` tag associated with the **fs.default.name** property to reflect your Slave Raspberry Pi's IP, for example:

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://192.168.1.85:54310</value>
  <description> The default file systems name.
  Contains the Slave Raspberry Pi's IP and the port
  number.</description>
</property>
```

Following this we need to edit the `mapred-site.xml` file to also include the IP address of the Slave Raspberry Pi. With your editor, open the file:

```
vim mapred-site.xml
```

Next update the `<value>` tag with the new IP address as shown in the following XML:

```
  <property>
    <name>mapred.job.tracker</name>
    <value>192.168.1.84:54311</value>
    <description> The Slave Raspberry Pi's IP and port that the
    MapReduce job tracker runs on. </description>
  </property>
```

The other files that were copied do not need to be amended. With the configuration updated we can now move on to formatting the file system. As with the Master Raspberry Pi you can run the `hadoop` command to kick off the process:

```
/home/pi/hadoop/hadoop-1.2.1/bin/hadoop namenode –format
```

Once the formatting is completed we can return to the Master Raspberry Pi, using the `exit` command, and test whether the machines can work in unison.

```
exit
```

From the Master Raspberry Pi's command line, execute the following shell script to start the distributed file system:

```
/home/pi/hadoop/hadoop-1.2.1/bin/start-dfs.sh
```

If everything is configured successfully you will start to see an output similar to the following in the terminal:

```
starting namenode, logging to /home/pi/hadoop/hadoop-
  1.2.1/libexec/../logs/hadoop-pi-namenode-raspberrypi.out
192.168.1.85: starting datanode, logging to /home/pi/hadoop/
  hadoop-1.2.1/libexec/../logs/hadoop-pi-datanode-raspberrypi2.out
192.168.1.84: starting datanode, logging to /home/pi/hadoop/
  hadoop-1.2.1/libexec/../logs/hadoop-pi-datanode-raspberrypi.out
192.168.1.84: starting secondarynamenode, logging to
  /home/pi/hadoop/hadoop-1.2.1/libexec/../logs/hadoop-pi-
  secondarynamenode-raspberrypi.out
```

After this we can then run the `start-mapred` script to start the MapReduce daemons, jobtracker, and tasktrackers:

```
/home/pi/hadoop/hadoop-1.2.1/bin/start-mapred.sh
```

The output displayed on your screen should be like:

```
192.168.1.85: starting tasktracker, logging to
  /home/pi/hadoop/hadoop-1.2.1/libexec/../logs/hadoop-pi-
  tasktracker-raspberrypi2.out
192.168.1.84: starting tasktracker, logging to
  /home/pi/hadoop/hadoop-1.2.1/libexec/../logs/hadoop-pi-
  tasktracker-raspberrypi.out
```

If everything ran successfully then congratulations the process is complete!

Now Hadoop has been set up, we can start exploring the technology at its core—the MapReduce framework.

Summary

In this chapter we introduced you to Apache Hadoop. You then installed Java and set up your first Raspberry Pi with the Hadoop software. Once this was done, you completed the setup of your second Raspberry Pi by installing Java and Hadoop.

With our environment ready to go, we can now explore MapReduce Java-based applications. The coming chapter will introduce you to this very topic and fill in some more detail on Hadoop Common.

5 MapReduce Applications with Hadoop and Java

In the previous chapter we setup Hadoop across two Raspberry Pis. In this chapter we will delve into MapReduce, the core paradigm of Hadoop and run our first MapReduce application.

We will also explore some technologies used in setting up a Hadoop, cluster and cover features such as HDFS in more detail.

MapReduce

MapReduce is a programming approach that allows systems to process large datasets in parallel.

The key concept is that of using two functions, `Map` and `Reduce`, that are combined to produce a desired result.

Its genesis can be found in functional programming and has been available in languages such as LISP for several decades. Google has been a driver for bringing it out of the functional programming paradigm into the **OOP** (**Object Orientated Programming**) world. Its contributions include publishing a seminal paper on the subject in 2004, and being granted a patent on the technology.

So how does MapReduce work? The Map function takes a data set and then operates on the data, returning another data set as an output. This output is then fed to the `Reduce` function, which subsequently operates on the data set once again and returns a smaller data set as an output.

So let's look at an example of how the `Map` function operates. The pseudo code function `CtoF` in the following code takes a list of temperatures in Celsius and then converts the values into Fahrenheit, giving us another list as an output:

```
Map CtoF [4, 3, 2, 1] -> [39.2, 37.4, 35.6, 33.8]
```

Here we can see the values from the initial input have been converted and then a list with the new values has been returned as an output.

Now we have an output from the `Map` function we can look at the `Reduce` function.

The `Reduce` function takes the output of the preceding `Map` function and operates on the data passed to it. For example, let's assume that we now wish to find the highest temperature (`max`) in the outputted data set; we could use a function as follows to achieve this:

```
Reduce max [39.2, 37.4, 35.6, 33.8] -> [39.2]
```

So the `Reduce` function has taken the set of data, output by our `Map` function, and returned a single value—the largest is of course 39.2.

Using this model we could take a large dataset, break it into separate chunks, and pass each of these chunks to a `Map` function. Finally each output can be fed into the `Reduce` function to give us our answer. Let's take a look at how this would work.

In the following scenario we have three datasets, each consisting of the temperature data in Celsius for a given European city. The `Map` function will convert the temperature from Celsius to Fahrenheit and the `Reduce` function will show us the highest temperature.

```
Dataset 1 = [(London, 10), (Madrid, 28), (Helsinki, 5)]
Dataset 2 = [(London, 11), (Madrid, 26), (Helsinki, 3)]
Dataset 3 = [(London, 8),  (Madrid, 24), (Helsinki, 1)]
```

Since we have three datasets, we can pass each dataset to a separate `Map` function (for example, each function could be on a separate Raspberry Pi).

Each `Map` function would then convert the temperatures located on the dataset to Fahrenheit, giving us the following output:

```
Output dataset 1 = [(London, 50),   (Madrid, 82.4), (Helsinki, 41)]
Output dataset 2 = [(London, 51.8), (Madrid, 78.8), (Helsinki, 37.4)]
Output dataset 3 = [(London, 46.4), (Madrid, 75.2), (Helsinki, 33.8)]
```

These outputs are then fed back to a single `Reduce` function, which would collect the data and return the highest temperature for each city.

Based on the preceding data this would be as follows:

```
Final dataset = [(London, 51.8),(Madrid,82.4),(Helsinki,41)]
```

The explained MapReduce concept is at the core of how Hadoop operates. Using Java we can write MapReduce-based programs that will then be passed to the various Hadoop nodes—in our case Raspberry Pi's.

MapReduce in Hadoop

In order to use Hadoop to run our MapReduce applications, there are key terminologies used in the technology we should understand.

These are briefly described as follows:

- **NameNode**: The NameNode is responsible for keeping the directory tree of all the files stored in the system and tracking where the file data is stored in the cluster.
- **JobTracker**: The JobTracker passes out MapReduce tasks to each Raspberry Pi in our cluster.
- **DataNode**: The DataNodes in our cluster use the HDFS to store replicated data.
- **TaskTracker**: A node in our Raspberry Pi cluster that accepts tasks.
- **Default configuration**: A default configuration file provides the base that the website-specific files overwrite/augment. An example is the core-default.xml file.
- **Site-specific configuration**: You will be familiar with this from the previous chapter. This configuration contains specifics about our own development environment, such as our Raspberry Pi's IP address.

A comprehensive guide to Hadoop's terms and phrases can be found on the Hadoop Wiki at `http://wiki.apache.org/hadoop/`.

One area that requires closer examination though is the Hadoop Distributed File System.

HDFS – The Hadoop distributed file system

The **HDFS** (**Hadoop Distributed File System**) is a scalable and distributed file system written in the Java programming language. Each node in the cluster, in our instance a Raspberry Pi, has a NameNode but may not have a DataNode present.

It is the clustering however of DataNodes that forms the distributed file system. This file system then communicates between nodes over TCP/IP and uses **Remote Procedure Calls** (**RPC**) to execute functions between each Raspberry Pi.

The data that is stored is replicated across nodes and the number of replications in Hadoop is by default set to three. This value can be changed as extra Raspberry Pis are added to your cluster by updating the `dfs.replication` parameter in your website configuration file. The replicated data is stored in the location you set in the `dfs.data.dir` parameter in the previous chapter.

The **Hadoop FileSystem** shell is the tool we use to interact with HDFS for adding and removing files and directories we wish to process. We shall be exploring the shell shortly, as a means to add data to our file system.

First, let's start up our Hadoop-related processes so we can experiment with the HDFS and prepare some directories for running our upcoming MapReduce application against.

We can start the HDFS via the `start-dfs.sh` shell script available at `/home/pi/hadoop/hadoop-1.2.1/bin/start-dfs.sh`.

Following this we can then fire off the `start-mapred.sh` script available at `/home/pi/hadoop/hadoop-1.2.1/bin/start-mapred.sh`.

Next, we can use the following `jps` command to check if all of the services are running as expected:

```
jps
```

You should now see output similar to the following:

```
12680 Jps

5996 SecondaryNameNode

6206 TaskTracker

6110 JobTracker

5784 NameNode

5888 DataNode
```

With our services running, we can now start creating directories with the FS shell. To invoke the shell you use the following command, where `<arguments>` is the specific shell command you wish to run:

```
hadoop fs <arguments>
```

A list of shell command arguments can be found on the Apache Hadoop website:

```
http://hadoop.apache.org/docs/stable/file_system_shell.html
```

For the MapReduce application we will be running, we need an input directory to store files. To create our input directory we use the following command:

```
hadoop fs -mkdir input
```

We now need to download text input files from the following BitBucket website:

```
https://bitbucket.org/IoTAtHome/hadoop-wordcount-files
```

Add these to the file system using the `–put` argument as follows:

```
hadoop fs -put *.txt input
```

Our files are now in place, ready for processing. Next we shall move on to creating a WordCount application that can run a MapReduce job on these text files and return a count of the instances of each word across the files.

The WordCount MapReduce program

The following code is a typical example of a MapReduce program for Hadoop. Its purpose is to take a number of input files and return a count of each word located in them.

We will be running this application in order to illustrate how the files we added to the FS can be processed.

Add the preceding code into a new file called `WordCount.java` created in `vim /home/pi/hadoop/apps/WordCount.java`:

```
package org.myorg;
import java.io.IOException;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
```

The preceding code includes the various libraries we will need to use in our application. After this let's add the WordCount class:

```
public class WordCount {

  public static class Map extends MapReduceBase implements
  Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output, Reporter
      reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokens = new StringTokenizer(line);
      while (tokens.hasMoreTokens()) {
        word.set(tokens.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output, Reporter
      reporter) throws IOException {
        int total = 0;
        while (values.hasNext()) {
          total += values.next().get();
        }
        output.collect(key, new IntWritable(total));
      }
    }
```

Inside the `WordCount` class definition, you will see the code responsible for using the Hadoop `Map` and `Reduce` functionality. This will be used to perform the tasks we explained earlier when looking at the MapReduce paradigm.

Following this we can add our main function as follows:

```
  public static void main(String[] args) throws Exception {

    String input = args[0];
    String output = args[1];
    String startupmsg = String.format("Word Count job started
    getting files from %s outputting to %s ", input, output);

    System.out.println(startupmsg);

    JobConf configuration = new JobConf(WordCount.class);
    configuration.setJobName("wordcount");

    configuration.setOutputKeyClass(Text.class);
    configuration.setOutputValueClass(IntWritable.class);

    configuration.setMapperClass(Map.class);
    configuration.setCombinerClass(Reduce.class);
    configuration.setReducerClass(Reduce.class);

    configuration.setInputFormat(TextInputFormat.class);
    configuration.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(configuration,
      new Path(input));
    FileOutputFormat.setOutputPath(configuration,
      new Path(output));

    JobClient.runJob(configuration);
  }
}
```

Our `main()` function here is the entry point of the program. This function uses a number of libraries from Hadoop in order to process the text files we downloaded from the Web. We pass in the input directory as a command-line argument, which the application will read from. The second command-line argument we pass is the output directory. Our application will create this for us.

When the application launches it will indicate the input and output directories in the terminal.

With the code added, we can save and exit the file, ready to try compiling and running the program.

Testing our application

Testing our application is a relatively simple task.

Let's start with compiling the code. All the following commands should be run from inside the following directory:

```
/home/pi/hadoop/apps
```

Inside of this `apps` folder create a new directory called `wordcount_classes`; this will be where we store our outputted JAR file. You can use the following `mkdir` command:

```
mkdir wordcount_classes
```

The tool we use for compiling Java code is `javac`. This handy command-line utility will take a number of inputs and compile our Java code, outputting an application we can run.

The `javac` command takes several flags and parameters. The first flag is `–classpath`. This flag points to the Hadoop core JAR file. This is required at the compilation time in order to generate our application. Following this we include the `–d` flag. The `–d` flag sets the destination directory for our class file.

Finally we reference the Java file we wish to compile, in this case `WordCount.java`, as follows:

```
javac -classpath /home/pi/hadoop/hadoop-1.2.1/hadoop-core-1.2.1.jar
  -d wordcount_classes WordCount.java
```

Next we run the following command to produce a JAR file. A JAR file is used to combine several Java class files into a single ZIP style file, in our case the output from the previous command.

```
jar -cvf /home/pi/hadoop/apps/wordcount.jar -C wordcount_classes/
```

The `c` flag in the previous command indicates that we wish to create a file. The `f` flag indicates that we want to write the data to the file rather than the terminal window. The `v` flag tells the JAR application to display verbose output.

Following these three flags we add the path and the name of the JAR file we want to create. Finally we include the `–C` flag. This changes the directory into `wordcount_classes` and the period (`.`) tells the command to grab everything in it.

After running the command you will see the output `wordcount.jar` file.

Using this we can execute our application as follows:

```
hadoop jar wordcount.jar org.myorg.WordCount input output
```

The application will now start and generate a number of output files including the word count.

These processed files can be found in the output directory using the FS shell as follows:

```
hadoop fs -ls output
```

We can copy the processed files back to Raspbian's file system using the `get` command.

First create a directory under `apps` to store the output using the `mkdir` command:

```
mkdir /home/pi/hadoop/apps/output
```

Then run the following hadoop command:

```
hadoop fs -get /user/pi/output/* /home/pi/hadoop/apps/output
```

The files have now been copied to the app directory. You can open one of the output files, for example `vim /home/pi/hadoop/apps/output/part-00000`.

Inside this file you will see a count of the instances of each word; the extract of the following output demonstrates this:

```
Goodbye 2
Hello   3
Hi      3
```

If you want to run the application again, delete the output folder using the `-rmr` command:

```
hadoop fs -rmr output
```

We have now demonstrated how we can use MapReduce for taking files and processing them via a Java application on multiple devices. As with MPI, this provides us with another tool for parallel computing tasks.

Summary

In this chapter we learned about the MapReduce paradigm. We also explored the Hadoop file system and tried out an application that processed a number of text documents to return a word count.

We shall now move on to writing our own MapReduce application for calculating π using a process known as a Monte Carlo simulator.

6 Calculate Pi with Hadoop and MPI

Now since we have set up Hadoop, written, and run our first application in it, we can look at the concept of Monte Carlo simulators and how we can calculate Pi (Π) using Hadoop and MPI. This brings together and compares the two technologies we have explored in Chapters 2 through 6.

Monte Carlo simulators

A Monte Carlo simulator, also known as Monte Carlo methods, is a type of computational method found in a variety of fields ranging from physics to finance.

Monte Carlo simulators use randomized sampling repeatedly in order to obtain a result for a particular mathematical question.

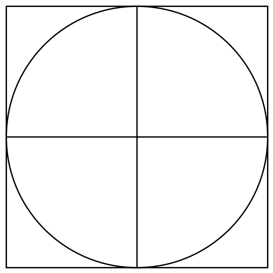
The name is derived from the city of Monte Carlo in Monaco. The origin of the name comes from Manhattan project participants *Stanislaw Ulam* and *John Von Neumann* in reference to a relative of Ulam who had a taste for gambling.

Calculating Π is an example of a problem especially suited to this type of algorithm and an early example of this is Buffon's needle. You can read more about the history of this experiment at Wolfram MathWorld:

`http://mathworld.wolfram.com/BuffonsNeedleProblem.html`

In order to calculate Π we can also use another method that involves a diagram displaying a circle located in a square divided into four quarters.

In this diagram we are interested in the top-right quarter of the circle. A simulator can generate numbers that corresponds to co-ordinates in this top-right portion bounded by the box. A number that falls inside the circle is recorded as **H**: a hit. Values that fall outside of the quarter circle are discarded. Finally we take the total number of co-ordinates generated and record it as **T**: the total. The diagram explaining this is as follows:



This method, akin to throwing darts, can be used to give us an estimate of Π. The more "throws" we have the closer we will come to calculating an accurate representation of the number.

Therefore the equation for calculating Pi this way can be summarized as:

*Pi = 4*H/T*

As you may have guessed this type of calculation is a prime candidate for parallel computing as it can be divided up between multiple cores and nodes and executed multiple times on each machine.

With this in mind we will now take a look at calculating Π with MapReduce by examining the example that comes bundled with Hadoop.

A Hadoop application to calculate Pi

Hadoop comes packaged with a number of example applications. We are of course interested in calculating Π program in particular.

The source code for this application can be downloaded from Apache's website at the following URL:

`https://svn.apache.org/repos/asf/hadoop/common/trunk/hadoop-mapreduce-project/hadoop-mapreduce-examples/src/main/java/org/apache/hadoop/examples/QuasiMonteCarlo.java`

The JAR file containing the compiled class can be found on your machine at:

```
/home/pi/hadoop/hadoop-1.2.1/hadoop-examples-1.2.1.jar
```

Let's navigate to this directory:

```
cd ~/hadoop/hadoop-1.2.1
```

We are now going to run the example. The program takes two inputs: the number of maps and the number of samples. Try running the following demonstration:

```
hadoop jar hadoop-examples-1.2.1.jar pi 2 4
```

You should now see something similar to:

```
Number of Maps  = 2
Samples per Map = 4
Wrote input for Map #0
Wrote input for Map #1
Starting Job
…
Job Finished in 273.167 seconds
Estimated value of Pi is 3.50000000000000000000
```

Here we can see that the value is fairly inaccurate compared to the first two decimal digits of Π, that is, 3.14. This is because we only had a small sample size and two map tasks.

By altering the number of maps and samples we can get a more accurate number. You can try experimenting with these values by changing the command line inputs.

For example, changing the values to `2` and `50` gives us an estimate of `3.2`, which is as we can see getting closer to a more accurate estimate.

This application is using the MapReduce framework to distribute the calculation of Pi before reducing the multiple results to a single estimate.

Now you have tried calculating Π via Hadoop; let's return to MPI, as you may remember we ran a similar exercise in *Chapter 3, Parallel Computing – MPI on the Raspberry Pi*.

Pi with C language and MPI

We have seen that we can calculate Π with Hadoop. We can now try a similar application in C. The program we will now write will generate results similar to what we saw with the example program included with MPICH and will also use a MapReduce-style approach.

Create a new file at the following location to store your code in:

```
~/mpich3/code/monte_carlo_pi.c
```

Open this file and add the following code:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

double insidecircle(int throws);

#define GAMES 20
#define THROWS 100
```

The previous block of code includes the necessary header files and defines a function and two constants. The function `insidercircle()` will be responsible for calculating Π.

The first constant is the number of `GAMES`, that is, attempts at calculating Π. The second defines the number of `THROWS` in each game. Now add the following code to the end of the file:

```
int main (int argc, char *argv[]) {

double jobaverage, calcpi, totalpi;
int rpi, totalrpi, i;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rpi);
MPI_Comm_size(MPI_COMM_WORLD,&totalrpi);
srand((int)time(NULL));
jobaverage = 0;

  for (i = 0; i < GAMES; i++) {

  calcpi = insidecircle(THROWS);
  MPI_Reduce(&calcpi, &totalpi, 1, MPI_DOUBLE, MPI_SUM, 0,
  MPI_COMM_WORLD);
```

```
        if (rpi == 0) {
          jobaverage = ((jobaverage * i) +
                       (totalpi/totalrpi))/(i + 1);
          printf("Game: %d calculated the value of pi as: %10.8f\n",
                  i,jobaverage);
        }
     }

      if (rpi == 0) {
        printf ("\nJob complete \n");
      }

      MPI_Finalize();
      return 0;
    }
```

The `main()` function in our code is used for making MPI calls, seeding the random number generator, and then kicking off the process of generating Π. We introduce here the `MPI_Reduce` function as well, which you can read more about at the **mpitutorial** site:

`http://mpitutorial.com/mpi-reduce-and-allreduce/`

Our function uses a `for` loop, based upon the number of games defined in the `GAMES` constant to call the `insidecircle()` function, which is tasked with the actual calculation of Π. Once complete it outputs the estimate of Π for that game.

Let's now add the `insidercircle()` function, which computes an estimate of Π, based upon the equation we examined earlier in this chapter:

```
    double insidecircle(int throws)
    {
      double xposcoord, yposcoord, pi, rnum;
      int hits, i;
        hits = 0;

        for (i = 1; i <= throws; i++)  {

        rnum = (double) rand()/((double) RAND_MAX);
        xposcoord = (rnum * 2.0) - 1.0;
        rnum = (double) rand()/((double) RAND_MAX);
        yposcoord = (rnum * 2.0) - 1.0;

        if ((xposcoord*xposcoord)+(yposcoord*yposcoord) <= 1.0){
                hits++;
        }
      }
      pi = 4.0 * (double)hits/(double)throws;
      return(pi);
    }
```

In the preceding code we can see we pass in the number of `throws` to the function. This will determine how many darts we throw at the circle.

Following this we then calculate whether the dart landed inside the section of the circle or outside it. For every dart that lands inside the circle we increment the `hits` variable by one.

Once we have exhausted the number of throws, we calculate Π using our previous equation and return this value to the `main()` function. When the `main()` function has completed all the games it exits.

Save the file you have added the code to and then exit. We are now going to test our application.

Using the `mpicc` compiler run the following command:

```
mpicc -lm -g -o monte_carlo_pi monte_carlo_pi.c
```

We now need to copy the compiled file to our second Raspberry Pi:

```
scp monte_carlo_pi pi@raspberrypi2:/home/pi/mpich3/code
```

Navigate back to the root of your user account and run `mpiexec`, passing in your monte_carlo_pi application as a parameter:

```
mpiexec -f pifile -n 2 ./mpich3/code/monte_carlo_pi
```

You should now see an output similar to the following:

```
Game: 0 calculated the value of pi as: 2.88000000
...
Game: 19 calculated the value of pi as: 3.10400000
```

Congratulations, your MPI-based application is now calculating Π. You can try amending the values stored in `GAMES` and `THROWS`, in order to garner a more precise estimate.

Summary

In this chapter, we brought together the technologies we have studied so far and compared them by looking at how they both solve the problem through parallel computing.

This problem was calculating Π using a Monte Carlo style simulator.

In case of the MPI, we wrote a small application in C, which gave us some more exposure to programming parallel applications.

Now that you have a taste for how these two technologies can be used, you have the context to explore both Hadoop and MPI in more detail including editing the C program and writing your own Java application.

In the next chapter, we shall be looking at further tasks which we can perform with our Raspberry Pi cluster.

7
Going Further

In this chapter, we will bring together what we have learned throughout this book and look at some next-step projects that build upon the work from the previous chapters. The projects located here include booting up your Raspberry Pi to use a USB HDD, creating a Lego enclosure, and writing an MPI-based application using Fortran.

Let's start by looking at the USB HDD project.

Booting from an external USB HDD

Booting the Raspberry Pi up to use a USB HDD will give you extra storage capacity and also provide you with a device that is faster and more robust when it comes to accessing data repeatedly. This is especially useful when running large MapReduce applications with Hadoop. The USB HDD should contain external power, if possible, as the max recommended current on the USB ports is 100 mA.

Follow the listed steps to set up your USB HDD and configure your SD card:

1. Plug in the USB device to one of your Raspberry Pi's USB port's; remember to leave the keyboard attached.
2. Using the Linux instructions from *Chapter 2*, *Setting Up your Raspberry Pi Software and Hardware for Parallel Computing*, grab the disk name and unmount the USB device. You can use the instruction at the following eLinux page to format your USB drive:

   `http://elinux.org/RPi_Easy_SD_Card_Setup`

> You can use `ls /dev/ | grep "sd"` to list connected devices and then to mount the device use `sudo mount /dev/sda1 ~/usb`, where `sda1` is your USB device. This will confirm the USB device is working.

3. After formatting the USB device mount it to a directory locally, for example:

   ```
   sudo mount /dev/sda1 ~/usb
   ```

4. Copy the image of your SD card root file system to the USB drive using the `dd` command, for example:

   ```
   sudo dd bs=1M if=/dev/mmcblk0p1 of=/dev/sda1
   ```

5. We can now edit/create the current SD card's `cmdline.txt` file as follows:

   ```
   vim /boot/cmdline.txt
   ```

6. If you created a new file, change the file to the following line:

   ```
   dwc_otg.lpm_enable=0 console=ttyAMA0,115200
     kgdboc=ttyAMA0,115200 console=tty1 root=/dev/sda1
     rootfstype=ext4 elevator=deadline rootwait
   ```

7. Run the partition resize tool using the following command:

   ```
   sudo resize2fs /dev/sda1
   ```

8. Once it ends, edit the `fstab` file to include the following code:

   ```
   vim usb/etc/fstab

   proc              /proc      proc    defaults   0      0
   /dev/mmcblk0p1    /boot      vfat    defaults   0      2
   /dev/sda1         /ext4      defaults,noatime   0      1
   ```

9. Save the changes and reboot your Raspberry Pi.

Your Raspberry Pi is now set up to run from the extra USB device. The boot loader will stay on the SD card; however your data is now stored on the HDD.

You should find a number of performance gains using this method, and the HDD has a longer life than the SD card. For example, you can now remove the root file system from the SD card and just use the USB device for storing items such as your HDFS.

Let's now take a look at building a case for our Raspberry Pi.

Building a Lego enclosure

Building a case for your device helps to protect it from getting damaged. One fun, popular, and cheap method of doing this is to use Lego bricks.

For those of you who don't have them, these can be purchased from most toy stores, Amazon, or can be ordered directly from the following Lego website:

`http://shop.lego.com/en-US/#shopxlink`

Lego also allows custom brick ordering; for example, you can order all red, green, or white bricks for your case at the following link:

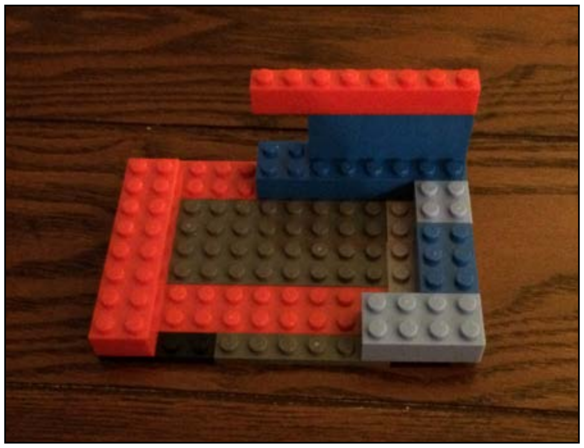
`http://shop.lego.com/en-US/Customized-Items-ByCategory`

For each Raspberry Pi case you will need the following:

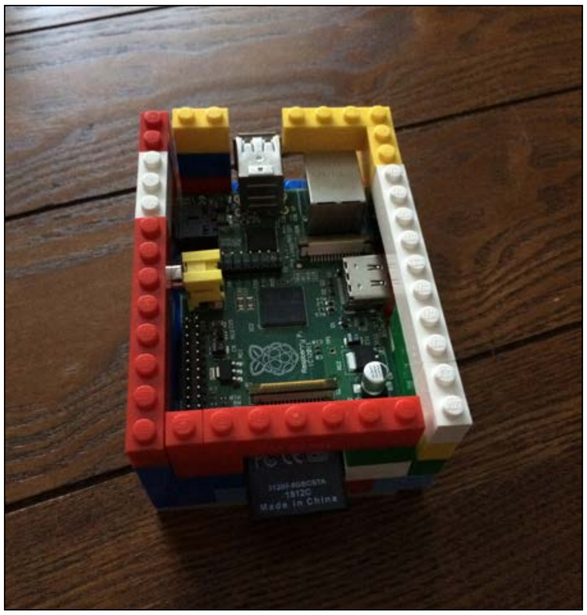
- Flat Lego panels to make a 13 x 9 stud rectangle.
- Bricks to go around the edges of this rectangle. These should be two-studs wide.
- Four flat tiles. These have no studs on top and cover an area of 1 x 2 studs when connected.
- Pieces to make a wall, three bricks high and one stud wide, to ring the 13 x 9 dimensions.

So let's get started with building our Lego enclosure. The steps for this are as follows:

1. Make a single layer base from the flat Lego piece 13 x 9 studs.
2. Following this, add a second layer around the edge with the two-stud wide blocks.



3. Add the flat tiles to prevent the bottom of the RPi catching on the studs.
4. Build up the walls leaving gaps for the RCA, HDMI, and Ethernet ports.
5. Once complete, the finished case should look like the following figure:



The top of the case does not contain a lid. This is optional and will help to prevent anything from falling onto your Raspberry Pi. However, by not including a lid you can stack cases upon each other, with the base of the top case acting as the lid of the bottom.

If you wish to make a lid, you can use the specs for the base of the Raspberry Pi case and attach this to the top of your Lego case walls.

This fairly simple design is of course very easy to modify and build upon. Please refer to the *Appendix* for further links and ideas.

If reading about Fortran earlier in the book intrigued you, then you may be interested in trying out the MPI Fortran application in the following section.

Experimenting with MPI and Fortran

The following Fortran program is an adaptation of the `hello_rpi.c` application from *Chapter 3, Parallel Computing – MPI on the Raspberry Pi.*

Create a new file called `hello_rpi.f` in the Fortran directory that you created earlier as follows:

```
vim /home/pi/Fortran/hello_rpi.f
```

Next add the following code to the file:

```
      program hellorpi
      include 'mpif.h'

      integer rpi, totalrpi, ierr

      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, rpi, ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, totalrpi, ierr)

      print *, 'This is Raspberry Pi ',(rpi+1),' of ',totalrpi

      call MPI_FINALIZE(ierr)
      end
```

You can see this program is very similar to the C one.

To start with, we include the `mpif.h` header. Following this we declare a number of integer variables to store the current Raspberry Pi executing the program, the total number of RPi's in the cluster, and an additional variable for an error code.

After this are the three MPI calls we made in the `hello_rpi.c` application.

A difference between the C and Fortran program is that we also include an extra parameter for our MPI calls—`ierr`. If you wish to expand the preceding application you can check the value of `ierr` and then halt the application or alert the user if there is a problem.

Finally, we print the Raspberry Pi ID to the screen.

Save the file and exit. We are now going to compile the code with the Fortran 77 compiler. To do this, run the following command:

```
mpif77 -o hello_pi hello_pi.f
```

Once the file has been compiled we need to copy it to the other Raspberry Pi using the following command:

```
scp hello_pi pi@raspberrypi2:/home/pi/Fortran
```

Now that the file is on both machines we can navigate back to our home directory and run the application using `mpiexec` as follows:

```
mpiexec -f pifile -n 2 ./Fortran/hello_pi
```

You should now see the following output:

```
This is Raspberry Pi           1  of           2
This is Raspberry Pi           2  of           2
```

Congratulations, you have now written another MPI-based program, this time in Fortran.

For more Fortran based MPI programs check out the following link:

```
http://people.sc.fsu.edu/~jburkardt/f_src/mpi/mpi.html
```

Power for multiple devices

In this section we take a brief look at some alternatives to using standard micro-USB phone-style chargers or desktop-mounted USB ports for powering your Raspberry Pi.

Each of the following suggestions should provide you with some ideas for future projects and provide some links to further reading. Remember, when working with mains electricity, to always make sure you take the necessary safety precautions and refer to a licensed professional.

USB wall plates

One alternative to the standard USB hubs is to install a wall mounted USB plate.

For running a small number of Raspberry Pis you can eliminate the need to use a USB hub or to use a desk-mounted multi adapter with USB ports.

Wall plates are directly wired into your mains electric as with a standard electricity socket.

These can be found at most home DIY stores. It is recommended that you have them installed by a certified electrician due to the risk of electric shock posed by working with mains electricity. Some plugs offer up to 4 USB ports so they are good for smaller projects.

Battery power

There are a number of projects dedicated to powering the Raspberry Pi via batteries. Batteries can be very useful if you wish to display your project at another location for a short amount of time. They also have the benefit of allowing you to move your project to another country that uses a different mains electricity system without needing to purchase a new set of wall warts. This of course is especially useful if, as we suggested previously, you wish to display the cluster.

The downside to battery power is that it only last a few hours and you will need some experience soldering and modifying electronics to get the most out of a battery-powered system. When modifying electronics such as the Raspberry Pi, there is a risk of damaging the board. You should therefore only attempt these projects if you are comfortable with the level of technical experience required.

The following URL provides a guide to powering your Raspberry Pi via battery, modifying your Raspberry Pi, and how to set it up:

`http://www.daveakerman.com/?page_id=1294`

Using a PC power supply

For electronically-minded people, it is possible to build your own USB-based hub that relies on a PC power supply. Once again, safety precautions should be taken when attempting projects such as this to avert the risk of an electric shock.

The Technology Toolshed site provides a comprehensive guide to building out a scalable USB-based power unit that can be expanded over time as you add more nodes to your cluster.

The guide can be found at the following URL:

`http://techtoolshed.blogspot.com/2013/01/power-requirements-of-raspberry-pi.html`

Power over Ethernet

Another option is to power your Raspberry Pi's over an Ethernet connection. You may be familiar with **VoIP** (**Voice over IP**) phones, which draw their power directly from the Ethernet cables plugged into them. Since your Raspberry Pi is already running Ethernet cables into a networking switch, you can optionally also power them this way, thus eliminating the need for a separate power unit.

The following website provides you with a guide to the hardware you can use to **PoE**-enable your Raspberry Pi. Please note that when attempting projects such as this you should take the utmost precautions to ensure that any devices you construct are safe, including adding fuses to protect your Raspberry Pis.

`http://www.xtronix.co.uk/raspberry-pi-poe.htm`

When using USB or power over Ethernet, patch panels can also be a useful addition to your setup. Once you start to increase the number of Raspberry Pis in your cluster, a patch panel can help to keep wires tidy and allows you to chain Ethernet/USB cables together.

You can find a wide variety of patch panels with numerous ports on the following Amazon website:

`http://www.amazon.com/`

Summary

This chapter concludes the projects presented in this book. Here we learned how to use the Raspberry Pi with a USB HDD. We also explored building a case using Lego. As a follow up to this, the *Appendix* provides you with a number of resources for purchasing Raspberry Pi cases or 3D-printing custom cases.

Following building a case, we explored writing another MPI-based application, this time using the Fortran programming language.

Finally we provided some ideas for alternative power sources for your Raspberry Pi cluster and some links to resources that discuss those.

The projects from this chapter, combined with those from across the rest of this book, have provided you with the basic tools needed to explore the topic of Parallel Computing further. You should now be familiar with MPI and Hadoop. You have also gained a solid understanding of how to build a new node and how to add it to your cluster.

We now leave it to you, the reader, to take the next steps in this exciting area of computing. For further links and project ideas please check out the *Appendix*.

Appendix

In this Appendix, we provide links to further resources referenced during this book and and websites covering the topics presented throughout the chapters in a more in-depth manner.

Fortran and C/C++

The following links provide further information on Fortran and C programming languages:

- The Chartered Institute for IT—Fortran Specialist Group

  `http://www.fortran.bcs.org/resources.php`

- The Fortran Company

  `http://www.fortran.com/the-fortran-company-homepage/fortran-tools-libraries-and-application-software/`

- Fortran and MPI—University of Kansas

  `http://condor.cc.ku.edu/~grobe/docs/intro-MPI.shtml`

- Fortran 77 programming guide—University of Leicester

  `http://www.star.le.ac.uk/~cgp/prof77.html`

- C programming resources

  `http://www.cprogramming.com/`

  `http://en.wikibooks.org/wiki/C_Programming`

  `http://people.sc.fsu.edu/~jburkardt/c_src/mpi/mpi.html`

MPI, Hadoop, and parallel computing

Further resources on MPI are as follows:

- Java examples—Cardiff University

  `http://users.cs.cf.ac.uk/David.W.Walker/CM0323/code.html`

- MPI tutorials—Lawrence Livermore National Laboratory

  `https://computing.llnl.gov/tutorials/mpi/`

- Hadoop MapReduce tutorial—Apache Foundation

  `http://hadoop.apache.org/docs/stable/mapred_tutorial.html`

- Raspberry Pi Cloud blog

  `http://raspberrypicloud.wordpress.com/2013/04/25/getting-hadoop-to-run-on-the-raspberry-pi/`

- Beowulf clusters—Duke University

  `http://www.phy.duke.edu/~rgb/brahma/Resources/beowulf/papers/ICPP95/icpp95.html`

- Parallel Computing—Intel

  `http://www.intel.com/pressroom/kits/upcrc/ParallelComputing_backgrounder.pdf`

- Parallel Programming—Gribble Lab

  `http://gribblelab.org/CBootcamp/A2_Parallel_Programming_in_C.html`

- Virtualization—Red Hat

  `http://www.redhat.com/products/cloud-computing/virtualization/`

- Virtual Machines—Virtual Box

  `https://www.virtualbox.org/`

Raspberry Pi cases and clusters

- Raspberry Pi cases—Thingiverse

  `http://www.thingiverse.com/thing:30563`

- Raspberry Pi case design—Raspberry Pi official website

  `http://www.raspberrypi.org/archives/1354`

- Raspberry Pi cases—Adafruit

  `http://www.adafruit.com/`

- Raspberry Pi super computer—Southampton University

  `http://www.southampton.ac.uk/~sjc/raspberrypi/pi_supercomputer_southampton.htm`

Index