MSc Scientific Computing Dissertation
Benchmarking a Raspberry Pi 4 Cluster

John Duffy

September 2020

Contents

Part I

Project Report

Chapter 1

Introduction

1.1 Arm

Since the release of the Acorn Computers Arm1 in 1985, as a second coprocessor for the BBC Micro, through to powering today's fastest supercomputer, the Japanese 8 million core Fugaku supercomputer, Arm has steadily grown to become a dominant force in the microprocessor industry, with more than 170+ billion Arm-based processors shipped to date.

Famed for power efficiency, which directly equates to battery life, Arm-based processors dominate the mobile device market for phones and tablets. And market segments which have almost exclusively been based upon x86 processors from Intel or AMD are also increasingly turning to Arm-based processors. Microsoft's current flagship laptop, the Surface Pro X, released in October 2019, is based on a Microsoft designed Arm-based processor. And Apple announced in June 2020 a roadmap to transition all Apple devices to Apple designed Arm-based processors within 2 years.

When Acorn engineers designed the Arm1, and subsequently the Arm2 for the Acorn Archimedes personal computer, low power consumptions was not the primary design criteria. Their focus was on simplicity of design. Influenced by research projects at Stanford University and the University of California, Berkeley, their focus was on producing a Reduced Instruction Set Computer (RISC). In comparison to a contemporary Complicated Instruction Set Computer (CISC), the simplicity of a RISC design required fewer transistors, which directly translated to a lower power consumption. The RISC design permitted the Arm2 to outperform the Intel 80286, a contemporary CISC design, whilst using less power.

Figure 1.1: The Raspberry Pi 4 Model B.

1.2 Raspberry Pi

The Raspberry Pi Foundation, founded in 2009, is a UK based charity whose aim is to "promote the study of computer science and related topics, especially at school level, and to put the fun back into learning computing". Through it's subsidiary, Raspberry Pi (Trading) Ltd, it provides low-cost, high-performance single-board computers called Raspberry Pi's, and free software.

At the heart of every Raspberry Pi is a Broadcom "System on a Chip" (SoC). The SoC integrates Arm processor cores with video, audio and Input/Output (IO). The IO includes USB, Ethernet, and General Purpose IO (GPIO) pins for interfacing with devices such as sensors and motors. The SoC is mounted on small form factor circuit board which hosts the memory chip, and video, audio, and IO connectors. A MicroSD card is used to boot the operating system and for permanent storage.

Initially released in 2012 as the Raspberry Pi 1, each subsequent model has seen improvements in SoC processor core count or performance, clock speed, connectivity and available memory.

The Raspberry Pi 1 has a single-core 32-bit ARM1176JZF-S based SoC clocked at 700 MHz and 256 MB of RAM. The RAM was increased to 512 MB in 2016.

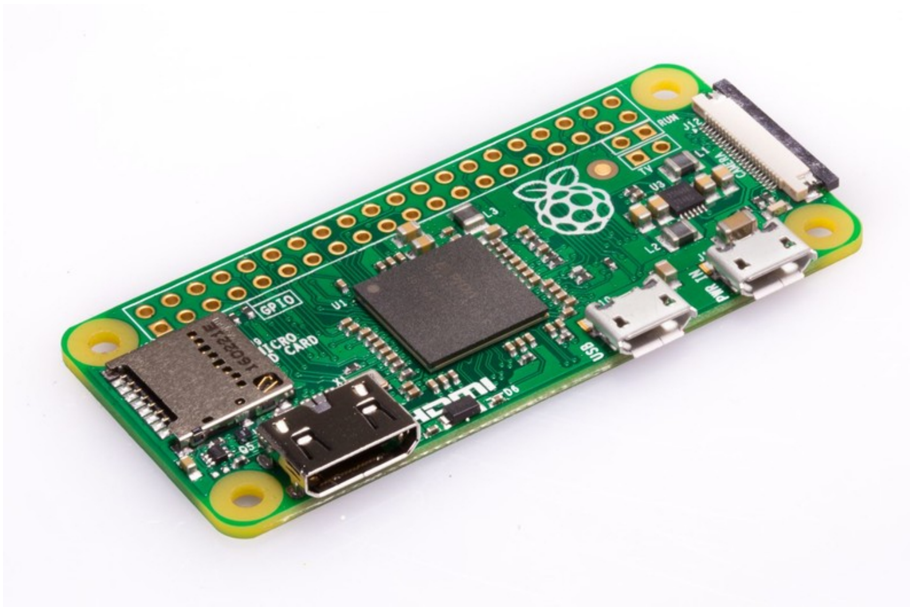


Figure 1.2: The Raspberry Pi Zero.

The Raspberry Pi 2, released in 2015, introduced a quad-core 32-bit Arm Cortex-A7 based SoC clocked at 900 MHz and 1 GB of RAM.

In 2016, the Raspberry Pi 3 was released with a quad-core 64-bit Arm Cortex-A53 based SoC clocked at 1.2 GHz, together with 1 GB of RAM.

The most recent addition to the range, in 2019, is the Raspberry Pi 4, sporting a quad-core 64-bit Cortex-A-72 based SoC clocked at 1.5 GHz. This model is available with 1, 2, 4 and 8 GB of RAM. This model with 4 GB of RAM was used for this project.

Since 2012 the official operating system for all Raspberry Pi models has been Raspbian, a Linux operating system based on Debian. Raspbian has recently been renamed Raspberry Pi OS. To support the aims of the Foundation, a number of educational software packages are bundled with Raspberry Pi OS. These include a ”non-commercial use” version of Mathematica, and a graphical programming environment aimed a young children called Scratch.

Python is the official programming language, due to its popularity and ease of use, and the inclusion of an easy to use Python IDE has been a Foundation priority. This is currently Thonny.

Even though the Raspberry Pi 3 introduced a 64-bit processor, Raspberry Pi OS has remained a 32-bit operating system. However, to complement the intro-

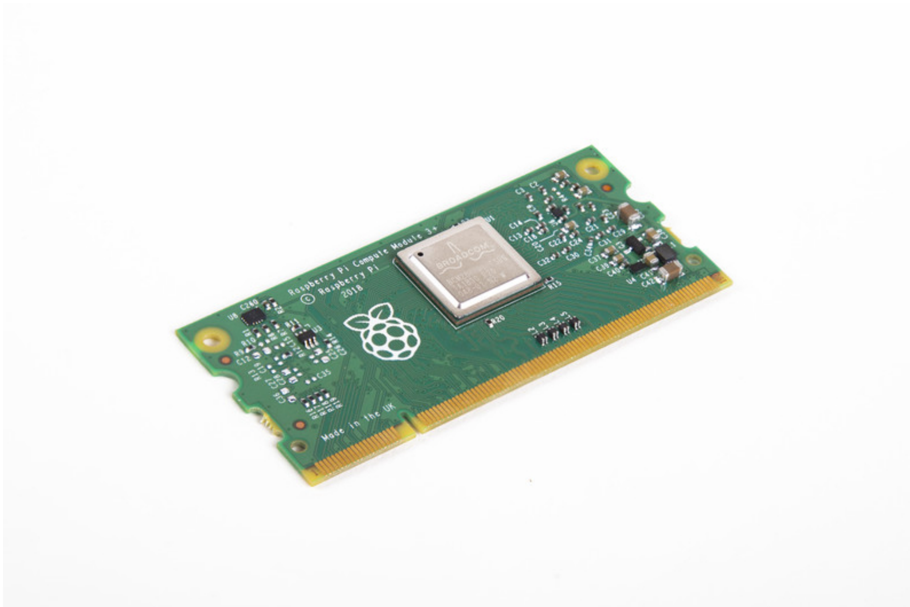


Figure 1.3: The Raspberry Pi Compute Module 3+ (CM3+).

duction of the Raspberry Pi 4 with 8 GB of RAM, a 64-bit version is currently in public beta testing.

Raspberry Pi OS is not the only operating system available for the Raspberry Pi. The Raspberry Pi website provides downloads for Raspberry Pi OS, and also NOOBS (New Out of the Box Software), together with a MicroSD card OS image burning tool called Raspberry Pi Imager. NOOBS and Raspberry Pi Imager make it easy to install operating systems such as Ubuntu, RISC OS, Windows 10 IoT Core, and more. Ubuntu 20.04 LTS 64-bit, the operating system used for this project, is available for download from the Ubuntu website, and is also available as an install option within Raspberry Pi Imager.

Since the release of the Raspberry Pi 1, the Raspberry Pi has been available in a number of model variants and circuit board formats. The Model B of each release is the most powerful variant, intended for desktop use. The Model A is a simpler and cheaper variant intended for embedded projects. The models B+ and A+ designate an improvement to the current release hardware. The Raspberry Pi Zero is a tiny, inexpensive variant without most of the external connectors, designed for low power, possibly battery powered, embedded projects. The Raspberry Pi Compute Module is a stripped down version of the Raspberry Pi without any external connectors. This model is aimed at industrial applications and fits in a standard DDR2 SODIMM connector.

1.3 Aims

1.3.1 Benchmark Performance

The main aim of this project is to benchmark the performance of an 8 node Raspberry Pi 4 Model B cluster using standard HPC benchmarks. These benchmarks include High Performance Linpack (HPL), HPC Challenge (HPCC) and High Performance Conjugate Gradient (HPCG).

A pure OpenMPI topology was benchmarked, together with a hybrid OpenMPI/OpenMP topology.

1.3.2 Performance Optimisations

Having determined a Baseline performance benchmark, opportunities for performance optimisations were investigated for a single core, single node and the whole cluster. Network optimisation was also investigated, and proved to be significant factor in overall cluster performance.

1.3.3 Investigate Gflops/Watt

The Green500 List ranks computer systems by energy efficiency, Gflops/Watt. In June 2020, ranking Number 1, the most energy-efficient system was the MN-3 by Preferred Networks in Japan, which achieved a record 21.1 Gigaflops/Watt. Ranking 200 was Archer at the University of Edinburgh, which achieved 0.497 Gflops/Watt.

The final aim of this project was to investigate where the Aerin cluster might fair in relation to the Green500 List.

1.4 Typography

This has been a very hands-on computing project with lots of Linux command line use. To enable a reader to replicate the cluster build and results, the Linux commands, output and file listings are included in the dissertation text, and colour coded as follows to aid readability.

This is a computer name.

`node1`

This is a command to type.

```
$ cat /proc/softirqs
```

This is the command output.

```
                    CPU0       CPU1       CPU2       CPU3
          HI:          1          0          0          1
       TIMER:    3835342    3454143    3431155    3431023
      NET_TX:      36635          0          0          0
      NET_RX:     509189        146        105        121
       BLOCK:      95326       4367       4311       4256
    IRQ_POLL:          0          0          0          0
     TASKLET:       4900          3          4         25
       SCHED:     444569     267214     218701     189120
     HRTIMER:         67          0          0          0
         RCU:     604466     281455     260784     277699
```

This is a long command to type.

```
$ mpirun -bind-to-socket -host node1:1,node2:1,node3:1 -np 3 -x OMP_NUM_THREADS=4 xhpl
```

Yes, in the command above, it is possible to reduce typing by putting the host data in a `hostfile`, but this illustrates the typography.

This is a file listing.

Listing 1.1: /etc/hosts

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1 localhost
255.255.255.255 broadcasthost
::1             localhost
192.168.0.1 node1
192.168.0.2 node2
192.168.0.3 node3
192.168.0.4 node4
192.168.0.5 node5
192.168.0.6 node6
192.168.0.7 node7
192.168.0.8 node8
192.168.0.9 node9
```

And this is something to take note of.

```
This is a 'gotcha', or
This differs from a similar build procedure, or
This is a 'hack' to be fixed permanently later, or
Don't do this at home, or,
Something similar
```

1.5 Project GitHub Repositories

The project code and benchmark results are hosted in the following GitHub repository.

`https://github.com/johnduffymsc/picluster`

This dissertation TeX and PDF files, and the Jupyter Notebook used to generate the plots, are hosted in the following GitHub repository.

`https://github.com/johnduffymsc/dissertation`

Chapter 2

ARM Architectures for HP

Chapter 3

The Aerin Cluster

This chapter describes the components of the Aerin Cluster, and includes some advice and lessons learned. Detailed build instructions are included in Part II Chapter 7.

Photo...

The Aerin Cluster consists of the following hardware and software components.

3.1 Hardware

- 8 x Raspberry Pi 4 Model B compute nodes, `node1` to `node8`
- 1 x Raspberry Pi 4 Model B build node, `node9`
- 9 x Official Raspberry Pi 4 power supplies
- 9 x Class 10 A1 MicroSD cards
- 9 x Heatsinks with integrated fans
- 1 x Netgear FVS318G 8 Port Gigabit Router/Firewall
- 1 x Netgear GS316 16 Port Gigabit Switch (with Jumbo Frame Support)
- Cat 7 cabling

3.1.1 Raspberry Pi's

The 9 x Raspberry Pi 4 used in the cluster are the 4GB RAM version. Recently, an 8GB RAM version became available. This which would be the preferred version for a future cluster.

The compute nodes of the cluster are `node1` to `node8`. These are used to run the benchmarks.

Some benchmarks require a substantial amount of time to run, so it is helpful to have a dedicated build node for compiling software, developing scripts, etc, while the benchmarks run on compute nodes. This build node is `node9`.

It is convenient to have one of the compute nodes designated the "master" node (this is a convenience and not a requirement). This is `node1`. If any software needs to compiled locally to the compute nodes, and not on the build node, then the "master" node is used to do this. This node is also used to mirror the GitHub repository and to run the various Pi Cluster Tools.

3.1.2 Power Supplies

The Raspberry Pi 4 is sensitive to voltage drops, especially whilst booting. So it was decided to purchase 9 Official Raspberry Pi 4 power supplies, rather than a USB hub with multiple power outlets which may not have been able to maintain output voltage whilst booting 9 nodes. The 9 power supplies do take up some space, so a future development would be to investigate a suitably rated USB hub.

3.1.3 MicroSD Cards

MicroSD cards are available in a number of speed classes and "use" categories. The recommended minimum specification for the Raspberry Pi 4 is Class 10 A1. The "10" refers to a 10 MB/s write speed. The "A" refers to the "Application" category, which supports at least 1500 read operations and 500 write operations per second.

3.1.4 Heatsinks

Cooling is a major consideration when building any cluster, even an 8 node Raspberry Pi cluster. The Raspberry Pi 4 throttles back the clock speed at approximately 85°C, which would not only have had a negative impact on benchmark results, but also on repeatability. So, it was very important to select suitable cooling. After some investigation, it was decided to purchase heatsinks with integrated fans. These proved to be very successful, with no greater than 65°C observed at any time, even with 100% CPU utilisation for many hours.

3.1.5 Network Considerations

The MTU is the network packet payload size in bytes, i.e. the size of your data that is transmitted in a single network packet. It is actually 28 bytes less than this due to network protocol overhead. We shall see later how a larger MTU can improve network efficiency and improve benchmark performance.

A Jumbo Frame is any MTU greater than 1500 bytes. There is no standard maximum size for a Jumbo Frame, but the norm seems to be 9000 bytes. Not all network devices support Jumbo Frames, and a change of MTU size from the default 1500 bytes has to be supported by all devices on the network (although some devices are smart enough to accommodate multiple MTU's).

As we shall see, the Raspberry Pi 4 has very good Ethernet networking capabilities. The theoretical maximum bandwidth of a Gigabit Ethernet connection is 1 Gbit/s. With the default MTU (Maximum Transmission Unit) of 1500 bytes, the Raspberry Pi 4 can achieve 930 Mbit/s. This is 93% of the theoretical maximum bandwidth. Increasing the MTU to 9000 bytes increases the achievable, and measurable, bandwidth to 980 Mbit/s. This is, effectively, full Gigabit speed. It is important we make full use of this with adequate network equipment.

It would be tempting to use any old router/firewall, switch, and cabling found lurking around in some dusty cupboard. This would be a mistake, and potentially cripple the cluster network. Courtesy of Ebay, I acquired a professional grade router/firewall and switch for less than £30 each. And the switch supports Jumbo Frames up to 9000 bytes, which we will be making use of.

3.1.6 Router/Firewall

The router/firewall acts a cluster interface to the outside world. The firewall wall only permits certain network packets access to the cluster through holes in the wall, in our case only `ssh` packets. One side of the firewall is the cluster LAN (Local Area Network). The other side of the firewall is the WAN (Wide Area Network).

In my home environment the WAN is connected to my ADSL router via an ethernet cable. This permits the compute nodes on the LAN to connect to the internet and download updates. When relocated to UCL, the WAN would be connected to the internal UCL network.

The router exposes a single IP address for the cluster to the WAN. Access from the outside world to the cluster is through this single IP address via `ssh`, which is routed to `node1`.

The router also acts as DHCP (Dynamic Host Configuration Protocol) server for the compute node LAN. Compute node hostnames, such as `node1` etc, are configured by a boot script which determines the node hostname from the last octet of the node IP address, served by the DHCP server based on the MAC address. This ensures that each compute node is always assigned the same LAN IP address and hostname across reboots.

It sounds more complicated than it actually is, and is easily configured through the router/firewall web-based setup. More details are in Part II Chapter 7.

3.1.7 Network Switch

The switch acts as an extension to the number of ports on the compute node LAN. And because it supports Jumbo Frames it can accommodate an MTU increase to 9000 bytes localised to the compute nodes.

My initial build of the Aerin Cluster only used the 8 port router/firewall. Only having 8 ports quickly became tiresome, so a 5 port switch was added so that I could directly connect `node9` and my `macbook` to the compute nodes without having to `ssh` through the firewall. Later, when it became apparent that the 5 port switch didn't support Jumbo Frames, this was replaced with the current 16 port switch. I did anticipate having to replace the router/firewall because it doesn't support Jumbo Frames, but the switch is sufficiently smart to route 9000 byte packets between the compute nodes, and fragment any packets to/from the outside world, through the router/firewall, into 1500 byte packets.

3.1.8 Cabling

Cat 5 network cables only support 100 Mbit/s, and without any electrical shielding. Cat 5e supports 1 Gbit/s without shielding. Cat 6 supports 1 Gbit/s, possibly with shielding depending on the cable. Cat 6a and Cat 7 support 10 Gbit/s with electrical shielding. Therefore, to ensure maximum use of the network capabilities of the Raspberry Pi 4, a minimum of Cat 5e cabling must be used. Because the Aerin Cluster cable lengths are relatively short, and therefore inexpensive, I opted to use CAT 7 cabling. My advice would be to do the same for any future clusters. Any performance limiting factor is then not the cabling.

If you need to use old cable, check the labelling!

3.2 Software

3.2.1 Operating System

The operating system used for the Aerin Cluster is Ubuntu 20.04 LTS 64-bit Pre-Installed Server for the Raspberry Pi 4. This can be downloaded from the Ubuntu website or installed via Raspberry Pi Imager.

3.2.2 `cloud-init`

The `cloud-init` system was originally developed by Ubuntu to simplify the instantiation of operating system images in cloud computing environments, such as Amazon's AWS and Microsoft's Azure. It is now an industry standard.

It is also very useful for automating the installation of the same operating system on a number of computers using a single installation image. This dramatically simplifies building a cluster.

The idea is that a `user-data` file is added to the `boot` directory of an installation image. When a node boots using the image, this file is read and the configuration/actions specified in this file are automatically applied/run as the operating system is installed.

For the Aerin Cluster the following configuration/actions were applied to each node:

- Add the user `john` to the system and set the initial password
- Add `john's` public key
- Update the `apt` data cache
- Upgrade the system
- Install specified software packages
- Create a `/etc/hosts` file
- Set the hostname based on the IP address

All of this is done from a single image and `user-data` file. The time invested in getting the `user-data` file right pays off handsomely, especially when the cluster may need to be rebuild from scratch a number of times.

The main software packages used for benchmarking installed by `cloud-init` are:

- build-essential
- openmpi-bin
- libopenblas0-serial
- libopenblas0-openmp
- libblis3-serial
- libblis3-openmp

This installs essential software build tools, such as C/C++ compilers, `make`, etc, OpenMPI binary and development files, and the OpenBLAS and BLIS libraries in both serial and OpenMP versions.

The package names for the OpenBLAS and BLIS libraries are somewhat cryptic and can cause confusion. For example, the BLIS packages libblis64-serial and libblis64-openmp are not the 64-bit packages we would expect to install on a 64-bit operating system. The "64" refers to the integer size for the BLAS library. The packages we need are the libblis3-serial and libblis3-openmp versions, which are still 64-bit packages.

3.2.3 Benchmark Software

The HPL, HPCC and HPCG benchmark software was all compiled locally from source. The instructions for how to do this are in Part II Chapter 7.

3.2.4 BLAS Library Management

On the Aerin Cluster we have two different BLAS libraries installed, OpenBLAS and BLIS, both in serial and OpenMP versions. It is obviously critical to have the same BLAS library configured as the "the BLAS library in use" on each node at the same time.

Debian/Ubuntu have a very clever mechanism for setting a particular version of a library to be the "the BLAS library in use". The is called the "alternatives" mechanism, and is not just used for BLAS libraries, there are lots of software packages with “alternatives”.

Each “alternative” has a name, in the case of the BLAS libraries it is called `libblas.so.3`. What is really clever is that you can build software, such as HPL, to link against `libblas.so.3`, and then change then change what this “alternative” points to without having to rebuild the software.

For example, to set the serial version of OpenBLAS to “the BLAS library in use” we update the “alternative” with the following command:

```
$ sudo update-alternatives --set libblas.so.3-aarch64-linux-
    ↪ gnu /usr/lib/aarch64-linux-gnu/openblas-serial/libblas
    ↪ .so.3
```

Alternatively, there is an interactive version of the command which allows you to select a BLAS library from a list of options:

```
$ sudo update-alternatives --config libblas.so.3-aarch64-
    ↪ linux-gnu
```

The “alternatives” mechanism is very clever, but when you need to set the BLAS library on 8 nodes quite frequently this is a lot of typing and also error prone. So to make life easier I wrote two BLAS library management wrapper scripts described in the section below.

3.2.5 Pi Cluster Tools

Even a cluster of only 8 nodes requires quite a bit of effort, and typing, to keep the system up to date and to ensure the same BLAS library is running on each node. Logging in to each node individually to do this is a chore, and more importantly it is very error prone.

To get around this problem, a number of `bash` scripts were written as Pi Cluster Tools. Each script loops over a list of node names and uses `ssh` to run a command remotely on each node in turn.

The following scripts are included as Pi Cluster Tools.

- upgrade
- reboot
- shutdown
- do

- libblas-query
- libblas-set

Listings of the scripts are included in Part II Chapter 17.

To run a particular tool, for example `upgrade`, type the following command which will upgrade all of the nodes sequentially.

```
$ ~/picluster/tools/upgrade
```

The `do` command "does" the same command on each node. Note the required quotation marks.

```
$ ~/picluster/tools/do "mkdir -p ~/picluster/hpl/hpl-2.3/bin
    ↪ /picluster"
```

Probably the most useful of the Pi Cluster Tools are the two BLAS library tools.

`libblas-query` queries the "the BLAS library in use" on each node. This is extremely useful for ensuring the same library is in use on each node.

For example.

```
$ ~/picluster/tools/libblas-query
```

```
node8... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node7... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node6... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node5... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node4... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node3... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node2... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
node1... /usr/lib/aarch64-linux-gnu/blis-openmp/libblas.so.3
```

`libblas-set` takes a single argument, openblas-serial, openblas-openmp, blis-serial, or blis-openmp, and then uses the "alternatives" mechanism to set the "BLAS library in use" on each node.

For example.

```
$ ~/picluster/tools/libblas-set openblas-serial
```

```
node8... done
node7... done
node6... done
node5... done
```

```
node4... done
node3... done
node2... done
node1... done
```

```
Pi Cluster Tools are not production quality, yet!
```

Chapter 4

HPC Benchmarks

4.1 Landscape

High Performance Linpack (HPL) is the industry standard HPC benchmark and has been for ??? years. It is used by Top500 and Green500 lists described in the following section. However, it has been criticised for producing a single number, and not being a true measure of real-world application performance. This has led to the creation of complementary benchmarks, namely HPC Challenge (HPCC) and High Performance Conjugate Gradients (HPCG). These benchmarks measure whole system performance, including processing power, memory bandwidth and network speed, in relation to standard HPC algorithms such as FFT and CG.

A more detailed description of each benchmark follows.

4.2 Lists

Top500...

Green500...

HPCG...

4.3 High Performance Linpack (HPL)

Reference Paper...

https://www.netlib.org/benchmark/hpl/...

Describe algorithm...

Terminology R_{peak}, R_{max}..., problem size...

Describe methodology for determining main parameters NB, N, P and Q...

N formula...

Reference http://hpl-calculator.sourceforge.net

4.3.1 HPL.dat

Describe HPL.dat parameters...

Listing 4.1: Example HPL.dat

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
0            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
26208        Ns
1            # of NBs
32           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
2            # of process grids (P x Q)
1 2          Ps
8 4          Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
```

```
64            swapping threshold
0             L1 in (0=transposed,1=no-transposed) form
0             U  in (0=transposed,1=no-transposed) form
1             Equilibration (0=no,1=yes)
8             memory alignment in double (> 0)
```

A detailed description of each line of this file is ...

4.3.2 HPL.out

Describe HPL.out...

It is very easy to use `grep`to find the lines in HPL.out containing the results. And to then conduct a general numeric sort, first by P and then by Gflops, to find Rmax for each P and Q pair, squeezing repeated white space down to a single space for readability.

```
$ grep WR HPL.out | sort -g -k 4 -k 7 | tr -s ' ' > HPL.out.sorted
```

Listing 4.2: Example HPL.out.sorted

```
WR00C2R2 26208 32 1 8 802.01 1.4965e+01
WR00R2C2 26208 32 1 8 799.75 1.5007e+01
WR00L2L2 26208 32 1 8 796.04 1.5077e+01
WR00C2C2 26208 32 1 8 794.65 1.5103e+01
WR00L2C2 26208 32 1 8 793.86 1.5118e+01
WR00C2L2 26208 32 1 8 793.67 1.5122e+01
WR00R2L2 26208 32 1 8 793.48 1.5126e+01
WR00R2R2 26208 32 1 8 790.26 1.5187e+01
WR00L2R2 26208 32 1 8 789.16 1.5208e+01
WR00R2L4 26208 32 1 8 774.49 1.5497e+01
WR00C2R4 26208 32 1 8 773.52 1.5516e+01
WR00L2L4 26208 32 1 8 770.20 1.5583e+01
WR00R2C4 26208 32 1 8 767.92 1.5629e+01
WR00L2C4 26208 32 1 8 763.10 1.5728e+01
WR00L2R4 26208 32 1 8 762.43 1.5742e+01
WR00R2R4 26208 32 1 8 761.92 1.5752e+01
WR00C2C4 26208 32 1 8 761.58 1.5759e+01
WR00C2L4 26208 32 1 8 757.87 1.5836e+01
WR00R2R2 26208 32 2 4 728.78 1.6468e+01
WR00R2C2 26208 32 2 4 728.21 1.6481e+01
WR00R2L2 26208 32 2 4 726.55 1.6519e+01
WR00C2R2 26208 32 2 4 722.38 1.6614e+01
WR00L2C2 26208 32 2 4 721.63 1.6632e+01
WR00L2L2 26208 32 2 4 721.54 1.6634e+01
WR00C2C2 26208 32 2 4 721.25 1.6640e+01
```

```
WR00C2L2  26208  32  2  4  720.82  1.6650e+01
WR00L2R2  26208  32  2  4  720.80  1.6651e+01
WR00L2R4  26208  32  2  4  692.09  1.7341e+01
WR00R2C4  26208  32  2  4  690.37  1.7385e+01
WR00C2L4  26208  32  2  4  686.69  1.7478e+01
WR00C2C4  26208  32  2  4  686.23  1.7489e+01
WR00C2R4  26208  32  2  4  686.08  1.7493e+01
WR00L2L4  26208  32  2  4  686.02  1.7495e+01
WR00L2C4  26208  32  2  4  685.88  1.7498e+01
WR00R2L4  26208  32  2  4  685.76  1.7502e+01
WR00R2R4  26208  32  2  4  684.45  1.7535e+01
```

4.3.3 Running xhpl

To run xhpl on the Aerin Cluster...

To run xhpl using the serial version of OpenBLAS...

```
$ ~/picluster/tools/picluster-set-libblas-openblas-serial
```

or, with the serial version of BLIS...

```
$ ~/picluster/tools/picluster-set-libblas-blis-serial
```

```
cd ~/picluster/hpl/hpl-2.3/bin/serial
mpirun -np 4 xhpl
```

4.4 HPC Challenge (HPCC)

HPCC...

4.5 High Performance Conjugate Gradients (HPCG)

HPCG...

4.6 BLAS Libraries

4.6.1 GotoBLAS

4.6.2 OpenBLAS

4.6.3 BLIS

4.7 OpenMPI Topology

The...

4.8 Hybrid OpenMPI/OpenMP Topology

The network...

Chapter 5

Benchmarking the Aerin Cluster

5.1 Theoretical Maximum Performance (Gflops)

The Raspberry Pi 4 Model B is based on the Broadcom BCM2711 System on a Chip (SoC). The BCM2711 contains 4 Arm Cortex-A72 cores clocked at 1.5 GHz.

Each core implements the 64-bit Armv8-A Instruction Set Architecture (ISA). This instruction set includes Advanced SIMD instructions which operate on a single 128-bit SIMD pipeline. This 128-bit pipeline can conduct two 64-bit double precision floating point operations (Flops) per clock cycle.

A *fused multiply-add* (FMA) instruction implements a multiplication followed by an add in a single instruction. The main purpose of FMA instructions is to improve result accuracy by conducting a single rounding operation on completion of both the multiplication and the add operations. A single FMA instruction counts as two Flops.

The theoretical maximum performance of a single Aerin Cluster node, R_{peak}, is therefore:

$$R_{peak} = 4 \text{ cores} \times 1.5 \text{ GHz} \times 2 \text{ doubles} \times 2 \text{ FMA} \tag{5.1}$$

$$= 24 \text{ Gflops} \tag{5.2}$$

This is only achievable continuously if every instruction in a program is an FMA instruction, which obviously cannot be the case, since data has to be loaded from memory and stored back into memory. Nevertheless, this is the standard measure of theoretical maximum performance.

The theoretical maximum performance of the Aerin Cluster as a whole is therefore:

$$R_{peak} = 8 \text{ nodes} \times 24 \text{ Gflops} \tag{5.3}$$
$$= 192 \text{ Gflops} \tag{5.4}$$

For the High Performance Linpack benchmark, to achieve 100% performance requires a problem size that utilises 100% of memory. Because the operating system requires memory, is it not possible to use 100% for benchmarks.

The Linux `dmesg` command prints out the kernel boot messages, which can be searched using `grep` to determine how memory is utilised on the system:

```
$ dmesg | grep Memory
```

```
[    0.000000] Memory: 3783876K/4050944K available (11772K
   ↪ kernel code, 1236K rwdata, 4244K rodata, 6144K init,
   ↪ 1072K bss, 201532K reserved, 65536K cma-reserved)
```

As can be seen, 37838776k of memory is available, which equates to 90% of the 4 GB (4194304k) on each node. It would be optimistic to expect to use every byte of this 90%, and using any more than this would result in swap space being used which would negatively impact benchmark results.

So, for the HPL baseline benchmarks, 80% of memory was chosen for the problem size. This is the amount suggested as an initial *good guess* in the HPL Frequently Asked Questions.

The above necessarily results in the baseline benchmarks only being able to achieve 80% of R_{peak} at best, 4.8 Gflops for a single core, 19.2 Gflops for a single node, and 153.6 Gflops for the 8 node cluster. These values are indicated on the HPL baseline result plots.

TODO: REFERENCE (https://www.netlib.org/benchmark/hpl/faqs.html).

5.2 HPL Baseline

Ubuntu 20.04 LTS 64-bit packages, without any tweaks...

80% of memory

Methodology...

1 core... to investigate single core performance... caveats... use 1GB of memory...

1 node... to investigate inter-core performance...

2 nodes... to investigate inter-core and inter-node performance...

1..8 nodes ... to investigate over scaling of performance with node count... with optimal N, NB, P and Q parameters determined from 2 node investigation... caveats...

5.2.1 1 Core HPL Baseline

The purpose of this baseline is to determine the performance of a single core running a single `xhpl` process, with the single core having exclusive access to the shared L2 cache.

As discussed in the previous section, the HPL problem size is restricted to 80% of available memory. In the case of this baseline, this is 80% of a single node's 4 GB.

Using values of block size NB from 32 to 256, in increments of 8, and using formula ?? to ensure the problem size N is an integer multiple of NB, results in the table below of NB and N combinations.

NB	N	NB	N	NB	N	NB	N	NB	N
32	18528	80	18480	128	18432	176	18480	224	18368
40	18520	88	18480	136	18496	184	18400	232	18328
48	18528	96	18528	144	18432	192	18432	240	18480
56	18536	104	18512	152	18392	200	18400	248	18352
64	18496	112	18480	160	18400	208	18512	256	18432
72	18504	120	18480	168	18480	216	18360	-	-

The HPL input file HPL.dat is populated with these NB and N combinations as follows, in this example using an NB of 32 and an N of 18528:

```
1             # of problems sizes (N)
18528         Ns
1             # of NBs
32            NBs
```

For this baseline a single `xhpl` process is run on both the pure OpenMPI and Hybrid OpenMPI/OpenMP topologies. In both of these cases HPL.dat is populated with processor grid parameters P and Q as follows:

```
1            # of process grids (P x Q)
1            Ps
1            Qs
```

This baseline is run on a pure OpenMPI topology with the following command:

```
$ mpirun --bind-to core -host node1:1 -np 1 xhpl
```

Explain bind to core...

This baseline is run on a hybrid OpenMPI/OpenMP topology with the following command:

```
$ mpirun --bind-to socket -host node1:1 -np 1 -x
    ↪ OMP_NUM_THREADS=1 xhpl
```

Explain bind to socket...

The results are plotted in Figure ??.

Observations

As expected, there is no noticeable performance difference between a pure OpenMPI and hybrid OpenMPI/OpenMP topology for a single `xhpl` process running on a single core.

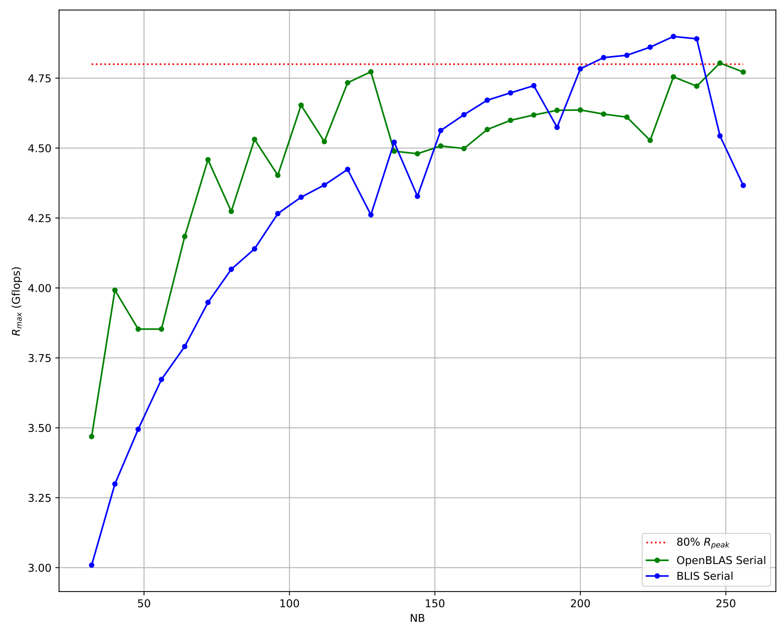
Both topologies attain 80% R_{peak} for a single core.

Discussion about OpenBLAS and BLIS internal kernel blocking...

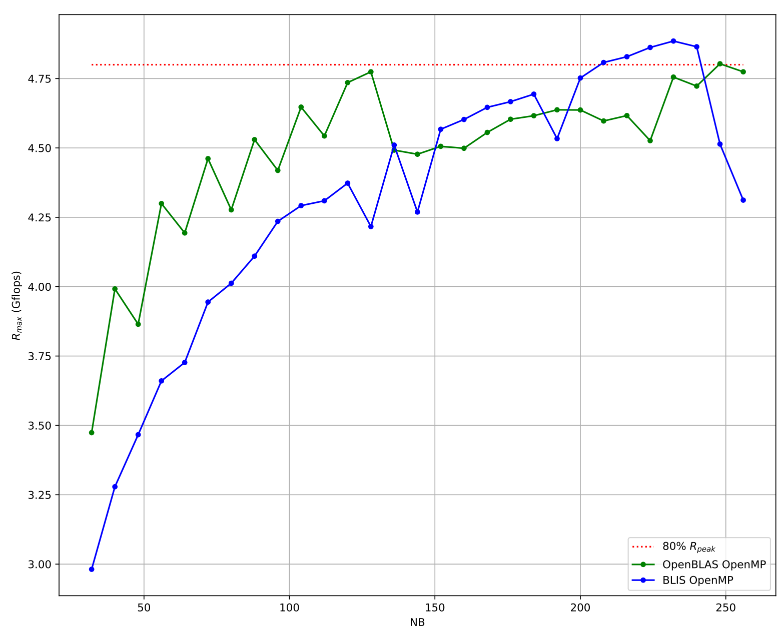
5.2.2 1 Node HPL Baseline

The purpose of this baseline is to determine the performance of the 4 cores of a single node. In this case each core has to share access to the L2 cache, which will result in more main memory accesses. It is therefore anticipated that this will result in a performance reduction, per core, compared to the single core case.

As per the single core benchmark, the HPL problem size is restricted to 80% of available memory. Again, in the case, this is 80% of a single node's 4 GB.



(a) Pure OpenMPI



(b) Hybrid OpenMPI/OpenMP

Figure 5.1: 1 Core R_{max} vs NB using 80% memory.

This results in the same table of NB and N combinations as the single core benchmark.

NB	N	NB	N	NB	N	NB	N	NB	N
32	18528	80	18480	128	18432	176	18480	224	18368
40	18520	88	18480	136	18496	184	18400	232	18328
48	18528	96	18528	144	18432	192	18432	240	18480
56	18536	104	18512	152	18392	200	18400	248	18352
64	18496	112	18480	160	18400	208	18512	256	18432
72	18504	120	18480	168	18480	216	18360	-	-

For the pure OpenMPI topology, 4 `xhpl` processes are run, one on each core. In this case HPL.dat is populated with processor grid parameters P and Q as follows:

```
1            # of process grids (P x Q)
1            Ps
4            Qs
```

And the pure OpenMPI topology baseline is run with the following command:

```
$ mpirun --bind-to core -host node1:4 -np 4 xhpl
```

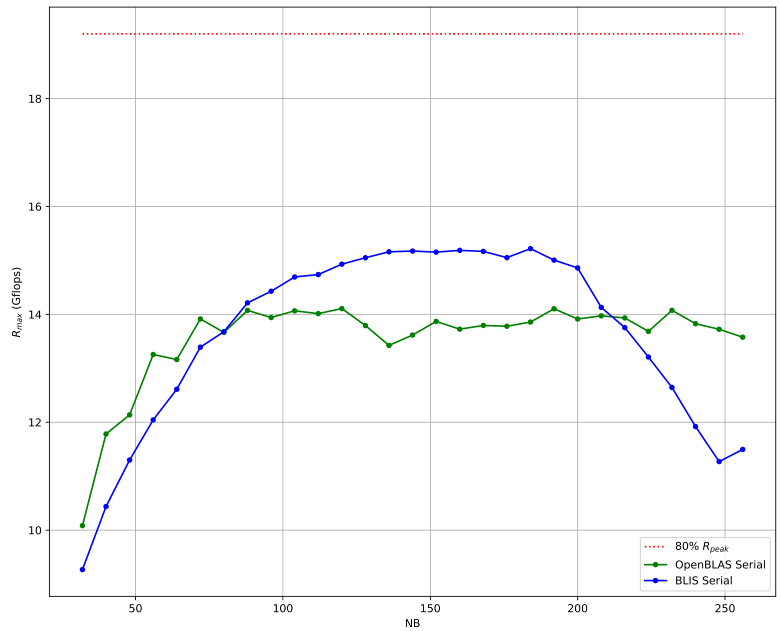
For the hybrid OpenMPI/OpenMP topology, a single `xhpl` process is run on the node. In this case the HPL.dat P and Q processor grid parameters are populated as follows:

```
1            # of process grids (P x Q)
1            Ps
1            Qs
```

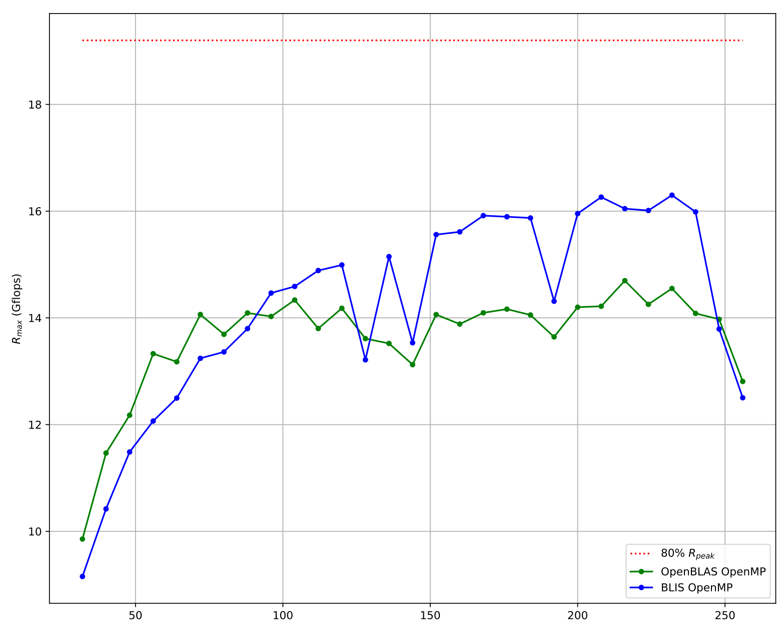
With 4 cores available to the multi-threaded BLAS library, the hybrid OpenMPI/OpenMP topology baseline is run with the following command:

```
$ mpirun --bind-to socket -host node1:1 -np 1 -x
    ↪ OMP_NUM_THREADS=4 xhpl
```

The results are plotted in Figure ??



(a) Pure OpenMPI



(b) Hybrid OpenMPI/OpenMP

Figure 5.2: 1 Node R_{max} vs NB using 80% memory.

Observations

As anticipated, there is indeed a reduction in performance per core, 80% R_{peak} in no longer attained.

Pure OpenMPI topology attains a R_{max} of ?? with an NB of ??.

The hybrid OpenMPI/OpenMP topology attains a R_{max} of ?? with an NB of ??.

5.2.3 2 Node HPL Baseline

The purpose of this baseline is to determine the performance of 2 nodes. Now, each core not only has to share access to the L2 cache, but the cache may be refreshed with data less frequently due to network delays and competition between the nodes for network resources. It is therefore anticipated that this will result in a performance reduction, per node, compared to the single node case.

For this baseline the HPL problem size is restricted to 80% of 2 nodes combined memory, 80% of 8 GB. This results in NB and N combinations as tabulated below:

NB	N	NB	N	NB	N	NB	N	NB	N
32	26208	80	26160	128	26112	176	26048	224	26208
40	26200	88	26136	136	26112	184	26128	232	25984
48	26208	96	26208	144	26208	192	26112	240	26160
56	26208	104	26208	152	26144	200	26200	248	26040
64	26176	112	26208	160	26080	208	26208	256	26112
72	26208	120	26160	168	26208	216	26136	-	-

For the pure OpenMPI topology, 8 `xhpl` processes are run, one on each core of each of the 2 nodes. Now it is possible to have 2 processor grid shapes, 1 x 8 and 2 x 4. In this case HPL.dat is populated with processor grid parameters P and Q as follows:

```
2            # of process grids (P x Q)
1 2          Ps
8 4          Qs
```

The pure OpenMPI topology baseline is run with the following command:

```
$ mpirun --bind-to core -host node1:4,node2:4 -np 8 xhpl
```

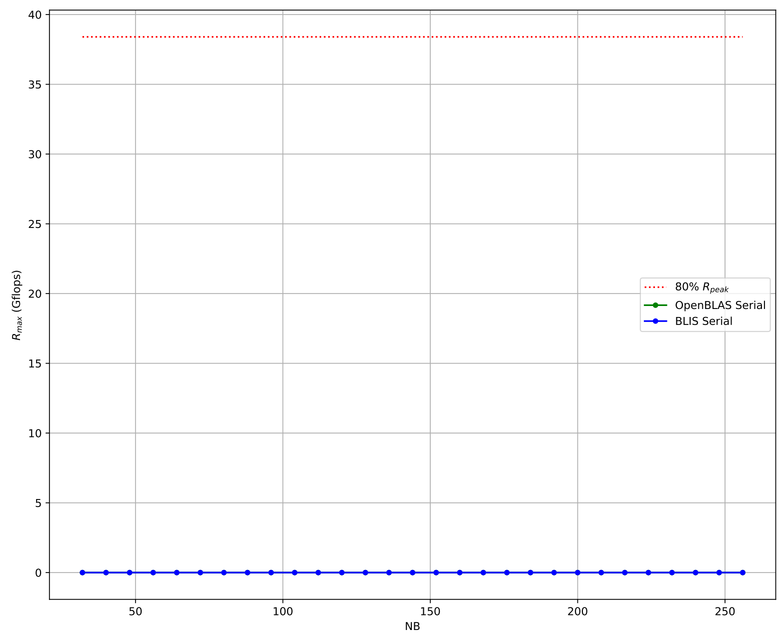
For the hybrid OpenMPI/OpenMP topology, a single `xhpl` process is run on each node. This results in single processor grid shape of 1 x 2, and the HPL.dat P and Q processor grid parameters are populated as follows:

```
1            # of process grids (P x Q)
1            Ps
2            Qs
```

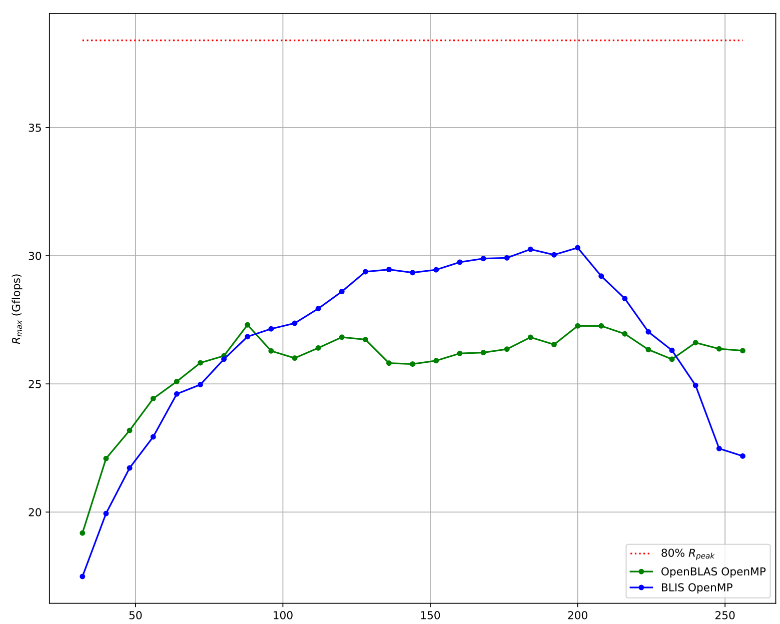
With the BLAS library utilising the 4 cores on each node, the hybrid OpenMPI/OpenMP topology baseline is run with the following command:

```
$ mpirun --bind-to socket -host node1:1,node2:1 -np 2 -x
    ↪ OMP_NUM_THREADS=4 xhpl
```

The results are plotted in Figure ??



(a) Pure OpenMPI



(b) Hybrid OpenMPI/OpenMP

Figure 5.3: 2 Node R_{max} vs NB using 80% memory.

Observations

5.2.4 Cluster HPL Baseline

This cluster baseline uses the optimum values of NB from the 2 Node Baseline. For each of the 4 BLAS library combinations, OpenBLAS serial, OpenBLAS OpenMP, BLIS serial, and BLIS OpenMP, with the corresponding value of N for 80% of memory for the particular node count is used, as tabulated below.

		Nodes					
BLAS	NB	3	4	5	6	7	8
OpenBLAS Serial	000	32000	36000	41000	45000	48000	52000
OpenBLAS OpenMP	000	32000	36000	41000	45000	48000	52000
BLIS Serial	000	32000	36000	41000	45000	48000	52000
BLIS OpenMP	000	32000	36000	41000	45000	48000	52000

These NB and N combinations are used to populate HPL.dat, as per the example below for the 3 node OpenBLAS Serial case.

```
1             # of problems sizes (N)
32000         Ns
1             # of NBs
000           NBs
```

For the 3 node pure OpenMPI baseline, the following HPL.dat processor grid shapes are used:

```
3             # of process grids (P x Q)
1 2 3         Ps
12 6 4        Qs
```

And the 3 node pure OpenMPI baseline is run with the following command:

```
$ mpirun --bind-to core -host node1:4,node2:4,node3:4 -np 12
    ↪ xhpl
```

For the 3 node hybrid OpenMPI/OpenMP baseline, the following HPL.dat processor grid shapes are used:

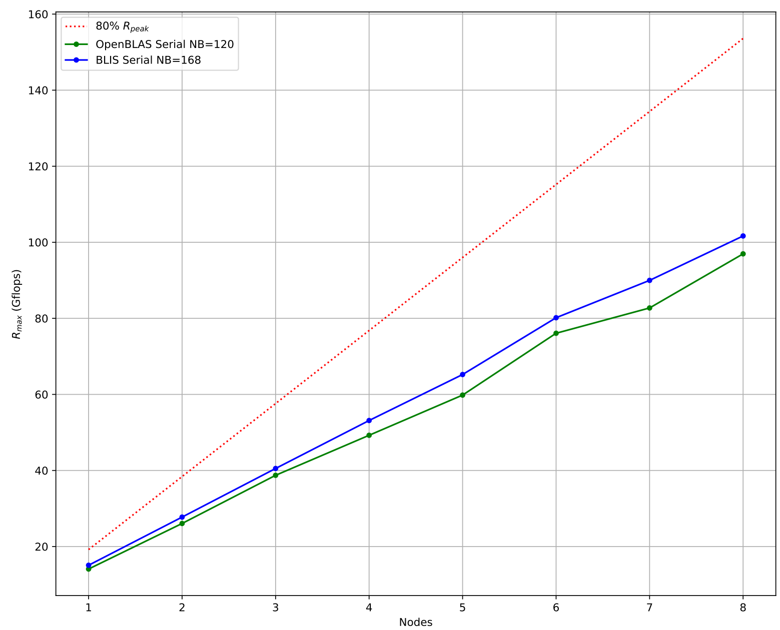
```
1             # of process grids (P x Q)
1             Ps
3             Qs
```

And the 3 node hybrid OpenMPI/OpenMP baseline is run with the following command:

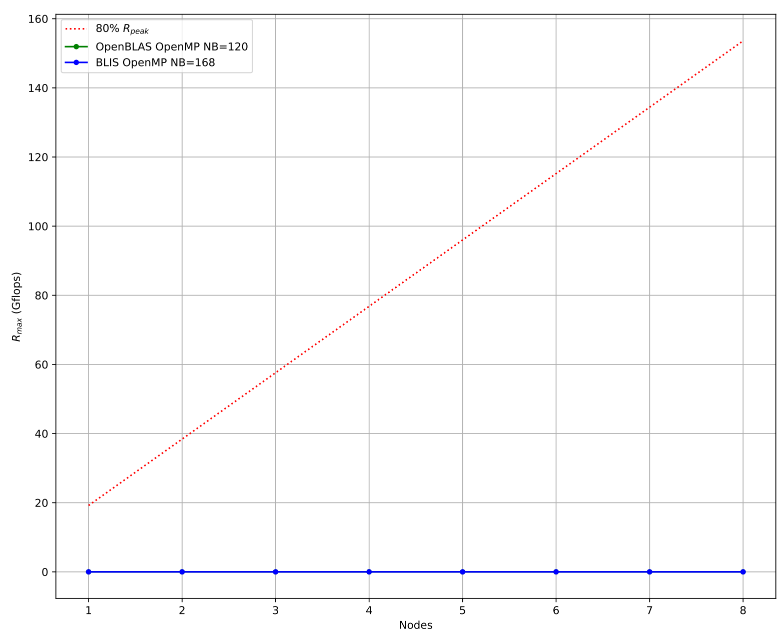
```
$ mpirun --bind-to socket -host node1:1,node2:1,node3:1 -np
    ↪ 3 -x OMP_NUM_THREADS=4 xhpl
```

For the 4, 5, 6, 7 and 8 node pure OpenMPI and hybrid OpenMPI/OpenMP baselines, HPL.dat is populated with the processor grid parameters P and Q in a similar manner. Likewise, the baselines are run in a similar manner.

The baseline results are presented in Figure ??.



(a) Pure OpenMPI



(b) Hybrid OpenMPI/OpenMP

Figure 5.4: R_{max} vs Nodes using 80% memory.

5.2.5 Observations

Best NB...

PxQ discussion... 1x8 vs 2x4... ethernet comment...

Iperf...

htop...

top...

perf...

cache misses...

software interrupts...

Suggests... improve network efficiency?

5.3 Optimisations

5.3.1 reclaim memory

A closer look at the memory use above indicates that 65536k of memory is being used as *cma-reserved*. This Contiguous Memory Allocator (CMA) memory is reserved at boot time for certain kernel drivers, in particular some video drivers. Since the Aerin Cluster is not using video, it may be possible to reclaim some of this memory to increase the amount available for the benchmark problem size.

The `/proc` filesystem enables access to kernel data structures at run time. Running the following command it is possible to see how the *cma-reserved* memory is being utilised:

```
$ cat /proc/meminfo | grep Cma
```

```
CmaTotal:           65536 kB
CmaFree:            63732 kB
```

As can be seen, the majority of *cma-reserved* memory is not being used. So, although this is a relatively small amount of memory on a single node, it is approximately 0.5 GB across all 8 nodes. This is worth trying to reclaim for the benchmark problem size via a rebuild of the kernel, something that is investigated later.

5.3.2 Single Core Optimisation

Rebuild libopenblas0-serial

Better BLAS library...

The Debian Science Wiki suggests...

So, following the instructions in /usr/local/share/

Details are in Appendix ?...

Poking around in the OpenBLAS source code, I noticed...

cpuid_arm64.c

in function void get_cpuconfig(void)

Listing 5.1: cpuid_arm64.c

```
...
  case CPU_CORTEXA57:
  case CPU_CORTEXA72:
  case CPU_CORTEXA73:
    // Common minimum settings for these Arm cores
    // Can change a lot, but we need to be conservative
    // TODO: detect info from /sys if possible
    printf("#define %s\n", cpuname[d]);
    printf("#define L1_CODE_SIZE 49152\n");
    printf("#define L1_CODE_LINESIZE 64\n");
    printf("#define L1_CODE_ASSOCIATIVE 3\n");
    printf("#define L1_DATA_SIZE 32768\n");
    printf("#define L1_DATA_LINESIZE 64\n");
    printf("#define L1_DATA_ASSOCIATIVE 2\n");
    printf("#define L2_SIZE 524288\n");
    printf("#define L2_LINESIZE 64\n");
    printf("#define L2_ASSOCIATIVE 16\n");
    printf("#define DTB_DEFAULT_ENTRIES 64\n");
    printf("#define DTB_SIZE 4096\n");
    break;
...
```

REFERENCE: Arm...

The following two lines are incorrect for the Arm Cortex-A72:

```
    printf("#define L2_SIZE 524288\n");
    printf("#define DTB_DEFAULT_ENTRIES 64\n");
```

To reflect the 1MB of L2 cache of the BCM??????, and the 32 entry L1 Data TLB, they should be:

```
    printf("#define L2_SIZE 1048576\n");
    printf("#define DTB_DEFAULT_ENTRIES 32\n");
```

Having changed these to the correct values, the build process now accurately reflects the 1MB of L2 cache on line 18 of 0-serial/config.h from which the libopenblas0-serial package is built:

Listing 5.2: 0-serial/config.h

```
1  #define OS_LINUX 1
2  #define ARCH_ARM64 1
3  #define C_GCC 1
4  #define __64BIT__ 1
5  #define PTHREAD_CREATE_FUNC pthread_create
6  #define BUNDERSCORE _
7  #define NEEDBUNDERSCORE 1
8  #define ARMV8
9  #define HAVE_NEON
10 #define HAVE_VFPV4
11 #define CORTEXA72
12 #define L1_CODE_SIZE 49152
13 #define L1_CODE_LINESIZE 64
14 #define L1_CODE_ASSOCIATIVE 3
15 #define L1_DATA_SIZE 32768
16 #define L1_DATA_LINESIZE 64
17 #define L1_DATA_ASSOCIATIVE 2
18 #define L2_SIZE 1048576
19 #define L2_LINESIZE 64
20 #define L2_ASSOCIATIVE 16
21 #define DTB_DEFAULT_ENTRIES 64
22 #define DTB_SIZE 4096
23 #define NUM_CORES 4
24 #define CHAR_CORENAME "CORTEXA72"
25 #define GEMM_MULTITHREAD_THRESHOLD 4
```

On completion of the build process, and after uninstalling the original libopenblas0-serial package and installing the new one...

Discussion...

Rebuild libblis3-serial

5.3.3 Single Node Optimisation

Kernel Preemption Model

The Linux kernel has 3 Preemption Models...

1... 2... The default 3...

As per the Help in the Kernel Configuration...

Listing 5.3: Kernel Configuration Preemption Model Help

```
CONFIG_PREEMPT_NONE:

This is the traditional Linux preemption model, geared
    ↪ towards
throughput. It will still provide good latencies most of the
time, but there are no guarantees and occasional longer
    ↪ delays
are possible.

Select this option if you are building a kernel for a server
    ↪  or
scientific/computation system, or if you want to maximize
    ↪ the
raw processing power of the kernel, irrespective of
    ↪ scheduling
latencies.
```

So, kernel rebuilt with CONFIG_PREEMPT_NONE=y

See Appendix ? on how to rebuild the kernel...

Installed on each node...

So, although this optimisation applies to single node, the benefits of applying this optimisation may not be apparent until the kernel has to juggle networking etc...

RESULTS...

Recieve Queues

```
$ sudo perf record mpirun -allow-run-as-root -np 4 xhpl
```

Running xhpl on 8 nodes using OpenBLAS...

```
$ mpirun -host node1:4 ... node8:4 -np 32 xhpl
```

SHORTLY AFTER PROGRAM START...

On node1,... where we initiated...

top...

```
top - 20:33:15 up 8 days,  6:02,  1 user,  load average:
    ↪ 4.02, 4.03, 4.00
Tasks: 140 total,   5 running, 135 sleeping,   0 stopped,
    ↪ 0 zombie
%Cpu(s): 72.5 us, 21.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0
    ↪ hi,  5.8 si,  0.0 st
MiB Mem :   3793.3 total,    330.1 free,   3034.9 used,
    ↪ 428.3 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.
    ↪ 698.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM
        ↪      TIME+ COMMAND
  34884 john      20   0  932964 732156   7980 R 100.3  18.8
      ↪  106:40.29 xhpl
  34881 john      20   0  933692 732272   7916 R 100.0  18.9
      ↪  107:29.75 xhpl
  34883 john      20   0  932932 731720   8136 R  99.3  18.8
      ↪  107:33.25 xhpl
  34882 john      20   0  932932 731784   8208 R  97.7  18.8
      ↪  107:33.64 xhpl
```

SOFTIRQS...

NODE 2 - 2 NODES ONLY TO SEE EFFECT...

IPERF!!!

On node8, running the top command...

```
$ top
```

We can see...

```
top - 18:58:44 up 8 days,  4:29,  1 user,  load average:
    ↪ 4.00, 3.75, 2.35
Tasks: 133 total,   5 running, 128 sleeping,   0 stopped,
    ↪ 0 zombie
```

```
%Cpu(s): 50.7 us, 47.8 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0
    ↪ hi,  1.4 si,  0.0 st
MiB Mem :   3793.3 total,    392.7 free,   2832.6 used,
    ↪ 568.0 buff/cache
MiB Swap:       0.0 total,      0.0 free,      0.0 used.
    ↪ 901.1 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM
        ↪      TIME+ COMMAND
  23928 john      20   0  883880 682456   8200 R 100.0  17.6
      ↪   13:14.17 xhpl
  23927 john      20   0  883988 682432   7932 R  99.7  17.6
      ↪   13:12.58 xhpl
  23930 john      20   0  883912 682664   7832 R  99.7  17.6
      ↪   13:17.01 xhpl
  23929 john      20   0  883880 682640   8376 R  99.3  17.6
      ↪   13:16.25 xhpl
```

Indicates that only 50.7% of CPU time is being utilised by user programs (us), Linpack/OpenMPI...

I hypothesise that the 1.4% of software interrupts (si) is responsible 47.8% of CPU time in the kernel (sy) servicing these interupts...

Lets have a look at the software interrupts on the system...

```
$ watch -n 1 cat /proc/softirqs
```

```
Every 1.0s: cat /proc/softirqs

                    CPU0       CPU1       CPU2       CPU3
          HI:          0          1          0          1
       TIMER:  122234556   86872295   85904119   85646345
      NET_TX:  222717797     228381     147690     144396
      NET_RX: 1505715680       1132       1294       1048
       BLOCK:      63160      11906      13148      11223
    IRQ_POLL:          0          0          0          0
     TASKLET:   58902273         33          2          6
       SCHED:    3239933    3988327    2243001    2084571
     HRTIMER:       8116         55         53         50
         RCU:    6277982    4069531    4080009    3994395
```

As can be seen...

1. the majority of software interrupts are being generated by network receive (NET_RX) activity, followed by network transmit activity (NET_TX)...

2. these interrupts are being almost exclusively handled by CPU0...

What is there to be done?...

1. Reduce the numbers of interrupts...

1.1 Each packet produces an interrupt - interrupt coalesing...

1.2 Reduce the number of packets - increase MTU...

2.1 Share the interrupt servicing activity evenly across the CPUs...

5.3.4 Network Optimisation

On node2 start the Iperf server...

```
$ iperf -s
```

On node1 start the Iperf client...

```
$ iperf -c
```

ping tests of MTU...

iperf network speed...

Jumbo Frames

Requires a network switch capable of Jumbo frames...

```
$ ip link show eth0
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
    ↪  state UP mode DEFAULT group default qlen 1000
    link/ether dc:a6:32:60:7b:cd brd ff:ff:ff:ff:ff:ff
```

```
$ ping -c 1 -s 1500 -M do node2
```

```
PING node2 (192.168.0.2) 1500(1528) bytes of data.
ping: local error: message too long, mtu=1500
```

```
$ ping -c 1 -s 1472 -M do node2
```

```
PING node2 (192.168.0.2) 1472(1500) bytes of data.
1480 bytes from node2 (192.168.0.2): icmp_seq=1 ttl=64 time
    ↪ =0.392 ms
```

Trying to set the MTU to 9000 bytes...

```
$ sudo ip link set eth0 mtu 9000
```

... results with...

```
Error: mtu greater than device maximum.
```

In fact, attempting to set the MTU to anything greater than 1500 bytes...

```
$ sudo ip link set eth0 mtu 1501
```

... results with...

```
Error: mtu greater than device maximum.
```

Need to build a kernel with Jumbo frame support...

See Appendix ?...

```
$ ip link show eth0
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq
   ↪  state UP mode DEFAULT group default qlen 1000
    link/ether dc:a6:32:60:7b:cd brd ff:ff:ff:ff:ff:ff
```

```
$ ping -c 1 -s 9000 -M do node2
```

```
PING node2 (192.168.0.2) 9000(9028) bytes of data.
ping: local error: message too long, mtu=9000
```

```
$ ping -c 1 -s 8972 -M do node2
```

```
PING node2 (192.168.0.2) 8972(9000) bytes of data.
8980 bytes from node2 (192.168.0.2): icmp_seq=1 ttl=64 time
   ↪ =0.847 ms
```

On `node2` create the `Iperf` server...

```
$ iperf -s
```

On `node1` create and run the `Iperf` client...

```
$ iperf -i 1 -c node2
```

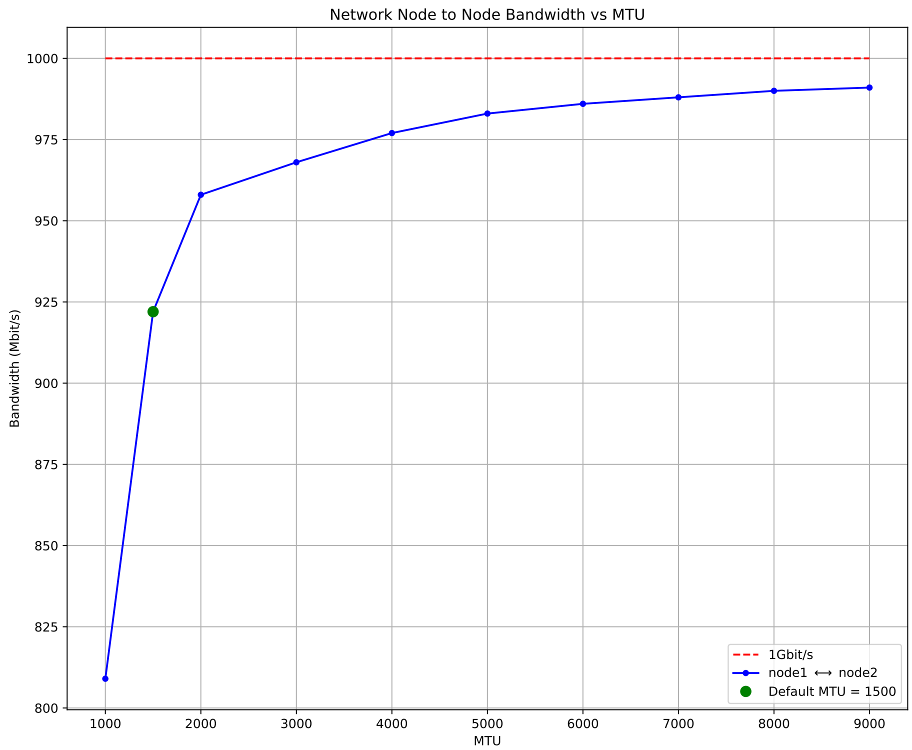


Figure 5.5: Network Node to Node Bandwidth vs MTU.

```
------------------------------------------------------------
Client connecting to node2, TCP port 5001
TCP window size:  682 KByte (default)
------------------------------------------------------------
[  3] local 192.168.0.1 port 46216 connected with
    ↪ 192.168.0.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.15 GBytes   991 Mbits/sec
```

5.3.5 Kernel TCP Parameters Tuning

REFERENCE...

https://www.open-mpi.org/faq/?category=tcp

Listing 5.4: /etc/sysctl.d/picluster.conf

```
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
net.core.netdev_max_backlog = 30000
net.core.rmem_default = 16777216
net.core.wmem_default = 16777216
net.ipv4.tcp_mem= 16777216 16777216 16777216
net.ipv4.route.flush = 1
```

```
sudo sysctl --system
```

or

```
sudo shutdown -r now
```

```
Aug 11 03:35:40 node5 kernel: [19256.425779] bcmgenet fd580000.ethernet eth0: bcmgenet_xmit: tx ring 1 full when queue 2 awake
```

Chapter 6

Summary

Part II

Build Instructions

Chapter 7

The Aerin Cluster

7.1 Introduction

This appendix is intended to be a complete and self contained guide for building a Raspberry Pi Cluster. With the caveat that the cluster has the bare minimum software/functionality necessary to compile and run the High Performance Linpack (HPL) benchmark, namely the build-essential package, two BLAS libraries (OpenBLAS and BLIS), and Open-MPI. A number of performance measurement tools are also installed, such as perf and iperf. The latest version of HPL is downloaded and built from source.

It would be a relatively simple task to add... SLIRM or...

The cluster consists of the following components...

8 x Raspberry Pi 4 Model B 4GB compute nodes, node1 to node8 1 x software development and build node, node9 9 x Official Raspberry Pi 4 Model B power supplies 9 x 32GB Class 10 MicroSD cards 1 x *workstation*, in my case my MacBook Pro, macbook 1 x 8 port Gigabit Router/Firewall 1 x 16 port Gigabit switch with Jumbo Frame support

Items

Photo

7.2 Preliminary Tasks

7.2.1 Update Raspberry Pi EE-PROMs

7.2.2 Obtain Raspberry Pi MAC Addresses

7.2.3 Generate User Key Pair

On macbook (no passphrase):

```
$ ssh-genkey -t rsa -C john
```

This will create two files... in ...

7.2.4 Amend macbook /etc/hosts

On `macbook`, using your favourite editor, add the following to /etc/hosts:

```
192.168.0.1 node1
192.168.0.2 node2
192.168.0.3 node3
192.168.0.4 node4
192.168.0.5 node5
192.168.0.6 node6
192.168.0.7 node7
192.168.0.8 node8
192.168.0.9 node9
```

This enables...

```
$ ssh john@node1
```

or, the abbreviated...

```
$ ssh node1
```

provided the user name on the macbook is the same as the Linux user created by cloud-init.

7.2.5 Router/Firewall Configuration

Local network behind firewall/switch: 192.168.0.254

WAN address LAN address

Firewall/Switch (Netgear FVS318G)

Describe DHCP reservations mapping IP to MAC addresses.

Describe ssh access

Add relevant PDFs.

7.3 Ubuntu 20.04 64-bit LTS Installation

The idea is to have a single (modified) Ubuntu 20.04 image which can be used to install Ubuntu 20.04 on all of the nodes...

7.3.1 Create the Installation Image

The instructions below are for MacOS but should be straightforward to adjust for other operating systems.

On `macbook`...

Download the Raspberry Pi 4 Ubuntu 20.04 LTS 64-bit pre-installed server image from the Ubuntu website.

Double click the compressed the `.xz` file to extract the `.img` file.

Double click the `.img` file to mount the image in the `macbook` filesystem as:

`/Volumes/system-boot`

We now need to edit the `user-data` file which stores the `cloud-init` configuration. The `user-data` file used to create the Aerin Cluster is at Listing 7.

Listing 7.1: /Volumes/system-boot/user-data

```
#cloud-config

# This is the user-data configuration file for cloud-init. By default this sets
# up an initial user called "ubuntu" with password "ubuntu", which must be
# changed at first login. However, many additional actions can be initiated on
# first boot from this file. The cloud-init documentation has more details:
#
# https://cloudinit.readthedocs.io/

# On first boot, set the (default) ubuntu user's password to "ubuntu" and
# expire user passwords
chpasswd:
  expire: true
  list:
  - ubuntu:ubuntu
  - john:john
```

```

# Enable password authentication with the SSH daemon
ssh_pwauth: true

## Add users and groups to the system, and import keys with the ssh-import-id
groups:
- john: [john]

users:
- default
- name: john
  gecos: John Duffy
  primary_group: john
  sudo: ALL=(ALL) NOPASSWD:ALL
  shell: /bin/bash
  ssh_authorized_keys:
  - ssh-rsa ...= john

## Update apt database and upgrade packages on first boot
package_update: true
package_upgrade: true

## Install additional packages on first boot
packages:
- git
- tree
- unzip
- iperf
- net-tools
- linux-tools-common
- linux-tools-raspi
- build-essential
- gfortran
- gdb
- fakeroot
- devscripts
- openmpi-bin
- libblis3-serial
- libblis3-openmp
- libopenblas0-serial
- libopenblas0-openmp

## Write arbitrary files to the file-system (including binaries!)
write_files:
- path: /etc/hosts
  content: |
    127.0.0.1 localhost
    192.168.0.1 node1
```

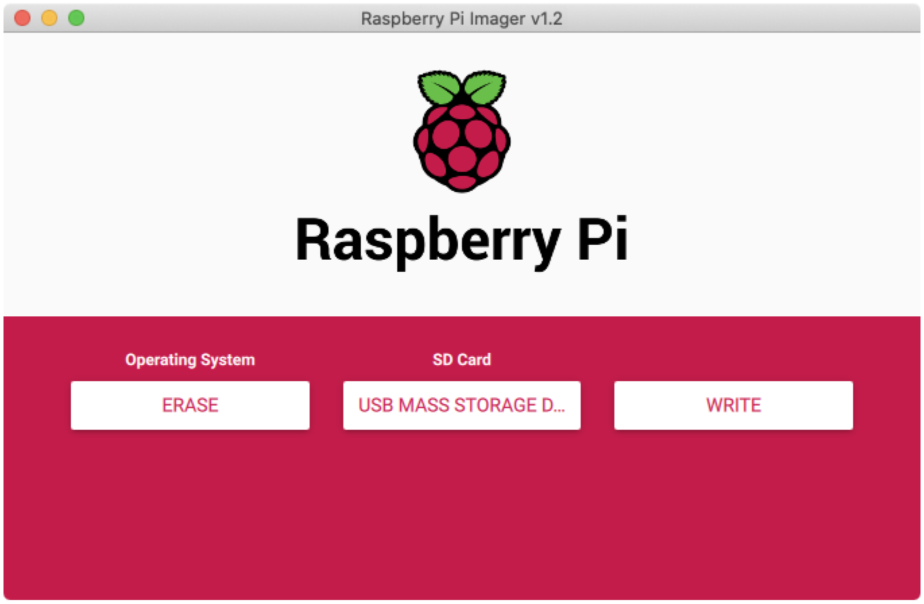


Figure 7.1: Using Raspberry Pi Imager to erase and format a MicroSD card.

```
    192.168.0.2 node2
    192.168.0.3 node3
    192.168.0.4 node4
    192.168.0.5 node5
    192.168.0.6 node6
    192.168.0.7 node7
    192.168.0.8 node8
    192.168.0.9 node9
  permissions: '0644'
  owner: root:root

## Run arbitrary commands at rc.local like time
runcmd:
- hostnamectl set-hostname --static node$(hostname -i | cut -d ' ' -f 1 | cut -d '.' -f 4)
- reboot
```

Eject/unmount the `.img` file.

Use Raspberry Pi Imager to erase...

Then use the Raspberry Pi Imager to write preinstalled server image to the MicroSD card...

When complete, remove the MicroSD card from the card reader, place it the

Figure 7.2: Using Raspberry Pi Imager to write the server image to a MicroSD card.

Raspberry Pi and plug in the power cable.

The cloud-init configuration process will now start. The Raspberry Pi will acquire its IP address from the router, setup users, update apt, upgrade the system, download software packages, set the hostname (based on the IP address), and finally the system will reboot.

7.4 Post-Installation Tasks

7.4.1 Enable No Password Access

This is required for Open-MPI...

Our public key was installed on each node by cloud-init. So, we can ssh into each node without a password, and use the abbreviated ssh node1, instead of ssh john@node1 (assuming john is the user name on the workstation).

We need to copy our private key to node1 (only node1)...

```
$ scp ~/.ssh/id_rsa node1:~/.ssh
```

Then to enable access to nodes node2 to node9 without a password from node1, we need to import the ... keys into the node1 knownhosts file...

This is easily done. From `macbook`...

```
$ ssh node1
```

And then from node1, for `node2` to `node9`...

```
$ ssh node2
```

This will generate will generate a message similar to...

```
The authenticity of host 'node2 (192.168.0.2)' can't be
    ↪ established.
ECDSA key fingerprint is SHA256:5VgsnN2nPvpfbJmALh3aJd0eT/
    ↪ NvDXqN8TCreQyNaFA.
Are you sure you want to continue connecting (yes/no/[
    ↪ fingerprint])?
```

Respond yes to this, which imports the host key into the `~/.ssh/knownhosts` file of `node1`.

And then exit from the connected node...

```
$ exit
```

Repeat the above for `node2` to `node9`.

The above is only required to be done once (unless the host keys on `node2` to `node9` change).

7.4.2 Uninstall unattended-upgrades

The `unattended-upgrades` package is installed automatically...

This can potentially interferer with long running benchmarks...

Remove...

From `macbook`:

```
$ ssh node1
$ ~/picluster/tools/do "sudo apt remove unattended-upgrades"
```

Don't forget to upgrade your cluster regularly at convenient times with...

```
$ ssh node1
$ ~/picluster/tools/upgrade
```

7.4.3 Add Ubuntu Source Repositories

We are going to be rebuilding some packages from source...

```
$ ssh node1
$ sudo touch /etc/apt/sources/list.d/picluster.list
$ sudo vim /etc/apt/sources/list.d/picluster.list
```

... and add the following source repositories...

Listing 7.2: /etc/apt/sources.list.d/picluster.list

```
deb-src http://archive.ubuntu.com/ubuntu focal main universe
deb-src http://archive.ubuntu.com/ubuntu focal-updates main universe
```

... and then update the repository cache...

```
$ sudo apt update
```

7.4.4 Create a Project Repository

Xpand upon...

```
$ ssh node1
$ mkdir picluster
```

```
$ cd picluster
$ git init
```

Ensure you do push your repository to a remote repository at regular intervals...

7.4.5 Select BLAS Library

The `cloud-init` process will have installed four BLAS libraries, namely...

`libopenblas0-serial`

`libopenblas0-openmp`

`libblis0-serial`

`libblis0-openmp`

To query the BLAS library currently in use on each node we can use one of our Pi Cluster tools...

```
$ ~/picluster/tools/libblas-query
```

```
node8... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node7... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node6... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node5... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node4... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node3... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node2... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
node1... /usr/lib/aarch64-linux-gnu/openblas-openmp/libblas.
    ↪ so.3
```

To select an alternative library we can use another of our Pi Cluster tools...

```
$ ~/picluster/tools/libblas-set blis-serial
```

```
node8... done
node7... done
node6... done
```

```
node5... done
node4... done
node3... done
node2... done
node1... done
```

```
$ ~/picluster/tools/libblas-query
```

```
node8... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node7... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node6... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node5... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node4... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node3... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node2... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
node1... /usr/lib/aarch64-linux-gnu/blis-serial/libblas.so.3
```

Chapter 8

Install High-Performance Linpack (HPL)

Download and install the latest version of HPL on `node1`...

```
$ ssh node1
$ cd ~/picluster
$ mkdir hpl
$ cd hpl
$ wget https://www.netlib.org/benchmark/hpl/hpl-2.3.tar.gz
$ gunzip hpl-2.3.tar.gz
$ tar xvf hpl-2.3.tar
$ rm hpl-2.3.tar
$ cd hpl-2.3
```

Create a `Make.picluster` file...

```
$ cd setup
$ bash make_generic
$ cp Make.UNKNOWN ../Make.picluster
$ cd ..
```

Amend `Make.picluster` as per listing ???.

Listing 8.1: /picluster/hpl/hpl-2.3/Make.picluster

```
#
#  -- High Performance Computing Linpack Benchmark (HPL)
#     HPL - 2.3 - December 2, 2018
#     Antoine P. Petitet
#     University of Tennessee, Knoxville
```

```
#     Innovative Computing Laboratory
#     (C) Copyright 2000-2008 All Rights Reserved
#
#  -- Copyright notice and Licensing terms:
#
#  Redistribution  and  use in  source and binary forms, with or without
#  modification, are  permitted provided  that the following   conditions
#  are met:
#
#  1. Redistributions  of  source  code  must retain the above copyright
#  notice, this list of conditions and the following disclaimer.
#
#  2. Redistributions in binary form must reproduce  the above copyright
#  notice, this list of conditions,  and the following disclaimer in the
#  documentation and/or other materials provided with the distribution.
#
#  3. All  advertising  materials  mentioning  features  or  use of this
#  software must display the following acknowledgement:
#  This  product  includes  software  developed  at  the University  of
#  Tennessee, Knoxville, Innovative Computing Laboratory.
#
#  4. The name of the  University,  the name of the Laboratory,  or the
#  names  of  its  contributors  may  not  be used to endorse or promote
#  products  derived   from   this  software  without specific  written
#  permission.
#
#  -- Disclaimer:
#
#  THIS  SOFTWARE  IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,  BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND  FITNESS FOR
#  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE UNIVERSITY
#  OR  CONTRIBUTORS  BE  LIABLE FOR ANY  DIRECT,  INDIRECT,  INCIDENTAL,
```

```
#  SPECIAL,  EXEMPLARY,  OR  CONSEQUENTIAL DAMAGES  ( INCLUDING,  BUT NOT
#  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#  DATA OR PROFITS; OR BUSINESS INTERRUPTION)  HOWEVER CAUSED AND ON ANY
#  THEORY OF LIABILITY, WHETHER IN CONTRACT,  STRICT LIABILITY,  OR TORT
#  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
# ######################################################################
#
# ----------------------------------------------------------------------
# - shell --------------------------------------------------------------
# ----------------------------------------------------------------------
#
SHELL        = /usr/bin/bash
#
CD           = cd
CP           = cp
LN_S         = ln -s
MKDIR        = mkdir -p
RM           = rm -f
TOUCH        = touch
#
# ----------------------------------------------------------------------
# - Platform identifier ------------------------------------------------
# ----------------------------------------------------------------------
#
ARCH         = picluster
#
# ----------------------------------------------------------------------
# - HPL Directory Structure / HPL library
```

```
------------------------------
# ----------------------------------------------------------------------
#
TOPdir       = $(HOME)/picluster/hpl/hpl-2.3
INCdir       = $(TOPdir)/include
BINdir       = $(TOPdir)/bin/$(ARCH)
LIBdir       = $(TOPdir)/lib/$(ARCH)
#
HPLlib       = $(LIBdir)/libhpl.a
#
# ----------------------------------------------------------------------
# - Message Passing library (MPI) --------------------------------------
# ----------------------------------------------------------------------
# MPinc tells the  C  compiler where to find the Message Passing library
# header files,  MPlib  is defined  to be the name of  the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
MPdir        = /usr/lib/aarch64-linux-gnu/openmpi
MPinc        = $(MPdir)/include
MPlib        = $(MPdir)/lib/libmpi.so
#
# ----------------------------------------------------------------------
# - Linear Algebra library (BLAS or VSIPL) -----------------------------
# ----------------------------------------------------------------------
# LAinc tells the  C  compiler where to find the Linear Algebra  library
# header files,  LAlib  is defined  to be the name of  the library to be
# used. The variable LAdir is only used for defining LAinc and LAlib.
#
LAdir        = /usr/lib/aarch64-linux-gnu
LAinc        =
LAlib        = $(LAdir)/libblas.so.3
#
```

```
# ----------------------------------------------------------------------------------------
# - F77 / C interface --------------------------------------------------
# ----------------------------------------------------------------------------------------
# You can skip this section  if and only if  you are not planning to use
# a  BLAS  library featuring a Fortran 77 interface. Otherwise,  it  is
# necessary  to  fill out the  F2CDEFS  variable  with  the  appropriate
# options.  **One and only one**  option should be chosen in  **each** of
# the 3 following categories:
#
# 1) name space (How C calls a Fortran 77 routine)
#
# -DAdd_              : all lower case and a suffixed underscore  (Suns,
#                       Intel, ...),                             [default]
# -DNoChange          : all lower case (IBM RS6000),
# -DUpCase            : all upper case (Cray),
# -DAdd__             : the FORTRAN compiler in use is f2c.
#
# 2) C and Fortran 77 integer mapping
#
# -DF77_INTEGER=int   : Fortran 77 INTEGER is a C int,         [default]
# -DF77_INTEGER=long  : Fortran 77 INTEGER is a C long,
# -DF77_INTEGER=short : Fortran 77 INTEGER is a C short.
#
# 3) Fortran 77 string handling
#
# -DStringSunStyle    : The string address is passed at the string loca-
#                       tion on the stack, and the string length is then
#                       passed as  an  F77_INTEGER  after  all  explicit
#                       stack arguments,                       [default]
# -DStringStructPtr   : The address  of  a  structure  is  passed  by  a
#                       Fortran 77  string,  and the structure is of the
#                       form: struct {char *cp; F77_INTEGER
```

```
len;},
# -DStringStructVal   : A structure is passed by value for each  Fortran
#                       77 string,  and  the  structure is of the form:
#                       struct {char *cp; F77_INTEGER len;},
# -DStringCrayStyle   : Special option for  Cray  machines, which  uses
#                       Cray  fcd  (fortran  character descriptor)  for
#                       interoperation.
#
F2CDEFS      = -DAdd_ -DF77_INTEGER=int -DStringSunStyle
#
# ----------------------------------------------------------------------
# - HPL includes / libraries / specifics -------------------------------
# ----------------------------------------------------------------------
#
HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) -I$(MPinc)
HPL_LIBS     = $(HPLlib) $(LAlib) $(MPlib)
#
# - Compile time options -----------------------------------------------
#
# -DHPL_COPY_L           force the copy of the panel L before bcast;
# -DHPL_CALL_CBLAS       call the cblas interface;
# -DHPL_CALL_VSIPL       call the vsip  library;
# -DHPL_DETAILED_TIMING  enable detailed timers;
#
# By default HPL will:
#    *) not copy L before broadcast,
#    *) call the BLAS Fortran 77 interface,
#    *) not display detailed timing information.
#
# HPL_OPTS   =
#
# ----------------------------------------------------------------------
#
HPL_DEFS     = $(F2CDEFS) $(HPL_OPTS) $(HPL_INCLUDES)
#
# ----------------------------------------------------------------------
```

```
# - Compilers / linkers - Optimization flags ---------------------------
#
# ----------------------------------------------------------------------
#
CC           = mpicc
CCNOOPT      = $(HPL_DEFS)
CCFLAGS      = $(HPL_DEFS) -O3 -march=armv8-a -mtune=cortex-a72
#
LINKER       = $(CC)
LINKFLAGS    = $(CCFLAGS)
#
ARCHIVER     = ar
ARFLAGS      = r
RANLIB       = echo
#
# ----------------------------------------------------------------------
```

Build HPL...

```
$ make arch=picluster
```

This creates the executable `xhpl` and input file `HPL.dat` in `bin/picluster`

The `xhpl` executable has to exist in the same location on each node, so copy `xhpl` to `node2` to `node8` (only `xhpl`, and not `HPL.dat`)...

```
$ cd bin/picluster
$ ~/picluster/tools/do "mkdir -p picluster/hpl/hpl-2.3/bin/picluster"
$ scp xhpl node2:~picluster/hpl/hpl-2.3/bin/picluster
$ scp xhpl node3:~picluster/hpl/hpl-2.3/bin/picluster
$ scp xhpl node4:~picluster/hpl/hpl-2.3/bin/picluster
$ scp xhpl node5:~picluster/hpl/hpl-2.3/bin/picluster
$ scp xhpl node6:~picluster/hpl/hpl-2.3/bin/picluster
$ scp xhpl node7:~picluster/hpl/hpl-2.3/bin/picluster
$ scp xhpl node8:~picluster/hpl/hpl-2.3/bin/picluster
```

Chapter 9

Install HPC Challenge (HPCC)

These instructions are derived from the README.txt file in the top level directory of the HPCC source code.

Download and install the latest version of HPCC on `node1`...

```
$ ssh node1
$ cd ~/picluster
$ mkdir hpcc
$ cd hpcc
$ wget http://icl.cs.utk.edu/projectsfiles/hpcc/download/
    ↪ hpcc-1.5.0.tar.gz
$ gunzip hpcc-1.5.0.tar.gz
$ tar xvf hpcc-1.5.0.tar
$ rm hpcc-1.5.0.tar
$ cd hpcc-1.5.0
```

Copy the HPL build script `Make.picluster` to the `hpl` directory...

```
$ cd hpl
$ cp ~/picluster/hpl/hpl-2.3/Make.picluster .
```

Make the following changes to `Make.picluster`. These differ from the build instructions for HPL.

Change the `TOPdir` variable to `../../..`

Listing 9.1: Make.picluster

```
TOPdir = ../../..
```

Add the `math` library explicitly, `-lm`, for the linker...

Listing 9.2: Make.picluster

```
LAlib = $(LAdir)/libblas.so.3 -lm
```

Add the constant `OMPI_OMIT_MPI1_COMPAT_DECLS` to `CCFLAGS`, otherwise the compilation fails...

Listing 9.3: Make.picluster

```
CCFLAGS = $(HPL_DEFS) -O3 -march=armv8-a -mtune=cortex-a72 -
    ↪ DOMPI_OMIT_MPI1_COMPAT_DECLS
```

Now move back up into the top level directory...

```
$ cd ..
```

Build HPCC...

```
$ make arch=picluster
```

Copy the `hpcc` executable to all of the nodes...

```
$ ~/picluster/tools/do "mkdir -p ~/picluster/hpcc/hpcc
    ↪ -1.5.0"
$ scp hpcc node2:~/picluster/hpcc/hpcc-1.5.0
$ scp hpcc node3:~/picluster/hpcc/hpcc-1.5.0
$ scp hpcc node4:~/picluster/hpcc/hpcc-1.5.0
$ scp hpcc node5:~/picluster/hpcc/hpcc-1.5.0
$ scp hpcc node6:~/picluster/hpcc/hpcc-1.5.0
$ scp hpcc node7:~/picluster/hpcc/hpcc-1.5.0
$ scp hpcc node8:~/picluster/hpcc/hpcc-1.5.0
```

Create the input file `hpccinf.txt`...

```
$ cp _hpccinf.txt hpccinf.txt
```

And amend as necessary. An input file which uses 80% of the total cluster memory is at Listing ???.

To run HPCC across all 8 nodes...

```
$ mpirun -host node1:4,node2:4,...node7:4,node8:4 -np 32
    ↪ hpcc
```

The output will be in the file `hpccoutf.txt`.

Chapter 10

Install High Performance Conjugate Gradients (HPCG)

Chapter 11

Ubuntu Kernel Build Procedure

This procedure is derived from the Ubuntu Wiki BuildYourOwnKernel document...

Make sure you have made the source code repositories available as per...

Create a kernel build directory with the correct directory permissions to prevent source download warnings.

```
$ ssh node1
$ mkdir -p ~/picluster/build/kernel
$ sudo chown _apt:root ~/picluster/build/kernel
$ cd ~/picluster/build/kernel
```

Install the kernel build dependencies...

```
$ sudo apt-get build-dep linux linux-image-$(uname -r)
```

Download the kernel source...

```
$ sudo apt-get source linux-image-$(uname -r)
$ cd linux-raspi-5.4.0
```

This bit is a fix for the subsequent `editconfigs` step of the build procedure...

```
$ cd debian.raspi/etc
$ sudo cp kernelconfig kernelconfig.original
$ sudo vim kernelconfig
```

And make the following change...

Listing 11.1: diff kernelconfig kernelconfig.original

```
5c5
<   archs="arm64"
---
>   archs="armhf arm64"
```

Then move back up to the kernel source top level directory...

```
$ cd ../..
```

Prepare the build scripts...

```
$ sudo chmod a+x debian/rules
$ sudo chmod a+x debian/scripts/*
$ sudo chmod a+x debian/scripts/misc/*
```

SOURCE CHANGES AND/OR verb—editconfigs— AT THIS POINT

```
$ sudo apt install libncurses-dev
$ sudo LANG=C fakeroot debian/rules clean
$ sudo LANG=C fakeroot debian/rules editconfigs
```

Tweak the kernel name for identification...

```
$ cd debian.raspi
$ sudo cp changelog changelog.original
$ sudo vim changelog
```

And make the following change, where `+picluster0` is our kernel identifier...

Listing 11.2: diff changelog changelog.original

```
1c1
< linux-raspi (5.4.0-1015.15+picluster0) focal; urgency=
    ↪ medium
---
> linux-raspi (5.4.0-1015.15) focal; urgency=medium
```

Move up to the top level kernel source directory...

```
$ cd ..
```

And build the kernel...

```
$ sudo LANG=C fakeroot debian/rules clean
$ sudo LANG=C fakeroot debian/rules binary-arch
cd ..
```

Install the new kernel...

```
$ sudo dpkg -i linux*picluster0*.deb
$ sudo shutdown -r now
```

Another build procedure fix...

After each kernel build delete the `linux-libc-dev` directory...

```
$ cd ~/picluster/build/kernel/linux-raspi-5.4.0/debian
$ rm -rf linux-libc-dev
$ cd ..
```

Chapter 12

Build Kernel with No Pre-Emption Scheduler

Chapter 13

Build Kernel with Jumbo Frames Support

Standard MTU is 1500 bytes...

Maximum payload size is 1472 bytes...

NB of 184 (x 8 bytes for Double Precision) = 1472 bytes...

NB > 184 => packet fragmentation => reduced network efficiency...

This causes drop of in performance???...

Max MTU on Raspberry Pi 4 Model B is set at build time to 1500...

Not configurable above 1500...

TODO: EXAMPLE OF ERROR MSG...

Need to build the kernel with higher MTU...

Make the required changes to the source... as per REFERENCE

```
cd linux-raspi-5.4.0

sudo vim include/linux/if_vlan.h...
    #define VLAN_ETH_DATA_LEN   9000
    #define VLAN_ETH_FRAME_LEN  9018

sudo vim include/uapi/linux/if_ether.h...
```

```
    #define ETH_DATA_LEN        9000
    #define ETH_FRAME_LEN       9014

sudo vim drivers/net/ethernet/broadcom/genet/bcmgenet.c...
    #define RX_BUF_LENGTH       10240
```

Add a Jumbo Frames identifier, "+jf", to the new kernel name...

```
sudo vim debian.raspi/changelog...
    linux (5.4.0-1013.13+jf) focal; urgency=medium
```

Chapter 14

Rebuild OpenBLAS

```
$ ssh node1
$ mkdir -p build/openblas
$ chown -R _apt:root build
$ cd build/openblas
$ sudo apt-get source openblas
$ sudo apt-get build-dep openblas
$ cd openblas-0.3.8+ds
```

Edit cpuid_arm64.c...

```
$ sudo cp cpuid_arm64.c cpuid_arm64.c.original
$ sudo vim cpuid_arm64.c
```

```
$ diff cpuid_arm64.c cpuid_arm64.c.original
```

```
275c275
<       printf("#define L2_SIZE 1048576\n");
---
>       printf("#define L2_SIZE 524288\n");
278c278
<       printf("#define DTB_DEFAULT_ENTRIES 32\n");
---
>       printf("#define DTB_DEFAULT_ENTRIES 64\n");
```

And, then following the instructions in debian/README.Debian

```
$ DEB_BUILD_OPTIONS=custom dpkg-buildpackage -uc -b
```

Once the build is complete..

```
cd ..
$ sudo apt remove libopenblas0-serial
$ sudo dpkg -i libopenblas0-serial\_0.3.8+ds-1\_arm64.deb
```

Ensure the correct BLAS library is being used...

```
$ sudo update-alternatives --config libblas.so.3-aarch64-
    ↪ linux-gnu
```

copy to other nodes remove old... install new...

If more than one BLAS library is installed, check update-alternatives!!!

ssh node2 .. node8

```
$ ssh node2 sudo apt remove libblas0-serial
$ scp libopenblas0-serial\_0.3.8+ds-1\_arm64.deb node2:~
$ ssh sudo dpkg -i libopenblas0-serial\_0.3.8+ds-1\_arm64.
    ↪ deb
$ ssh sudo update-alternatives --config libblas.so.3-aarch64
    ↪ -linux-gnu
```

Chapter 15

Rebuild BLIS

```
$ ssh node1
$ mkdir -p picluster/build/blis
$ cd picluster/build/blis
$ apt-get source blis
$ sudo apt-get build-dep blis
$ cd blis-0.6.1
```

Chapter 16

Build OpenMPI from Source

Do all of this on node1...

```
$ ssh node1
```

We want to avoid collisions with multiple OpenMPI installations, so remove original installed version...

```
$ sudo apt remove openmpi-common
$ sudo apt remove openmpi-bin
$ sudo apt autoremove
```

OpenMPI requires the libevent-dev package...

```
$ sudo apt install libevent-dev
```

Create a build directory, and download and, and and following BLAH, BLAH build OpenMPI...

```
$ mkdir -p ~/picluster/build/openmpi
$ cd ~/picluster/build/openmpi
$ wget https://download.open-mpi.org/release/open-mpi/v4.0/
    ↪ openmpi-4.0.4.tar.gz
$ gunzip openmpi-4.0.4.tar.gz
$ tar xvf openmpi-4.0.4.tar
$ rm openmpi-4.0.4.tar
$ cd openmpi-4.0.4
$ mkdir build
```

```
$ cd build
$ ../configure CFLAGS="-O3 -march=armv8-a -mtune=cortex=a72"
$ make all
$ sudo make install
$ sudo ldconfig
```

OpenMPI will installed to /usr/local

EXTRACT FROM HPL.dat

TODO: HOW TO COPY TO ALL NODES!

Chapter 17

Aerin Cluster Tools

Listing 17.1: picluster/tools/upgrade

```
#!/usr/bin/bash

NODES=9

for (( i=$NODES; i>0; i-- ))
do
  echo "Upgrading node$i..."
  ssh node$i sudo apt update
  ssh node$i sudo apt full-upgrade --yes
  ssh node$i sudo apt autoremove --yes
  ssh node$i sudo shutdown -r now
done
```

Listing 17.2: picluster/tools/reboot

```
#!/usr/bin/bash

NODES=8

for (( i=$NODES; i>0; i-- ))
do
  echo "Rebooting node$i..."
  ssh node$i sudo shutdown -r now
done
```

Listing 17.3: picluster/tools/shutdown

```
#!/usr/bin/bash

NODES=8
```

```
for (( i=$NODES; i>0; i-- ))
do
  echo "Shutting down node$i..."
  ssh node$i sudo shutdown -h now
done
```

Listing 17.4: picluster/tools/libblas-query

```
#!/usr/bin/bash

NODES=8

for (( i=$NODES; i>0; i-- ))
do
  printf "node$i... "
  ssh node$i update-alternatives --query libblas.so.3-aarch64-linux-gnu \
    | grep Value: \
    | gawk '{print $2}'
done
```

Listing 17.5: picluster/tools/libblas-set

```
#!/usr/bin/bash

NODES=8

case $1 in
  "openblas-serial" | "openblas-openmp" | "blis-serial" | "blis-openmp")
    for (( i=$NODES; i>0; i-- ))
    do
      printf "node$i... "
      ssh node$i sudo update-alternatives --quiet --set \
        libblas.so.3-aarch64-linux-gnu \
        /usr/lib/aarch64-linux-gnu/$1/libblas.so.3
      printf "done\n"
    done
    exit
    ;;
esac

echo "Usage: libblas-set {openblas-serial|openblas-openmp|blis-serial|blis-openmp}"
```

Chapter 18

Arm Performance Libraries

This does not work, yet! HPL will compile and link to Arm Performance Libraries, but raises an illegal instruction error at runtime.

At the time of writing, Arm Performance Libraries release 20.2.0 requires a minimum Instruction Set Architecture (ISA) of armv8.1-a. Unfortunately, the Raspberry Pi's Cortex-A72 ISA is armv8.0-a. An Arm representative has indicated on the Arm HPC Forum that the next release of the libraries will support the armv8.0-a ISA.

This Chapter is included for future reference.

The Arm Performance Libraries website states:

"Arm Performance Libraries provides optimised standard core math libraries for high-performance computing applications on Arm processors. This free version of the libraries provides optimised libraries for Arm® Neoverse™ N1-based Armv8 AArch64 implementations that are compatible with various versions of GCC. You do not require a license for this version of the libraries."

To install Arm Performance Libraries, firstly downloaded Arm Performance Libraries 20.2.0 with GCC 9.3 for Ubuntu 16.04+ from the Arm website.

Then follow these instructions.

```
$ ssh node1
```

Install the required `environment_modules` package.

```
$ sudo apt install environment-modules
```

Then extract and install Arm Performance Libraries.

The default installation directory is /opt/arm.

```
$ mkdir ~/picluster/armpl
$ cd ~/picluster/armpl
$ tar xvf arm-performance-libraries_20.2_Ubuntu-16.04_gcc
    ↪ -9.3.tar
$ rm arm-performance-libraries_20.2_Ubuntu-16.04_gcc-9.3.tar
$ sudo ./arm-performance-libraries_20.2_Ubuntu-16.04.sh
```

Copy the `Make.picluster` configuration file.

```
$ cd ~/picluster/hpl/hpl-2.3
$ cp Make.picluster Make.picluster-armpl
```

Make the following changes to `Make.picluster-armpl`.

Listing 18.1: Make.picluster-armpl

```
LAdir        = /opt/arm/armpl_20.2_gcc-9.3
LAinc        =
LAlib        = -L$(LAdir)/lib -larmpl -lgfortran -lamath -lm
```

Build HPL.

```
$ make arch=picluster-armpl
```