Bachelor's thesis

Tivi in English

2022

Vaihela Emil

Creation and optimization of HPC Clusters

TURKU AMK
TURKU UNIVERSITY OF
APPLIED SCIENCES

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Bachelor of Engineering, Information and Communications Technology

2022 | 46 pages

Vaihela Emil

Creation and optimization of HPC Clusters

This thesis aimed to investigate the contents of computer clustering and more specifically high-performance clustering. The need for computational power in various fields is developing rapidly. Simulations can be run in the field of medicine, used for molecular modelling finding cures for various diseases or simulating weather. Today’s supercomputers are incredibly expensive, huge facilities but the scalability of high-performance clusters allows smaller systems to be built.

The thesis will investigate what clusters are and how they work, introducing a few commonly used applications and software interfaces. An HPC cluster with consumer computers was built and optimized for benchmarking as well as using an NVIDIA DGX-1 workstation for reference. The clusters were benchmarked using the LINPACK benchmark.

The results showed that a cluster can be built by having basic skills in various IT fields but the cluster can be hard to optimize and reach good performance with consumer components. Further steps could be taken to improve the performance.

Keywords:

High-performance computing, Cluster, Slurm, MPI

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2022 | 46 sivua

Vaihela Emil

Kotitekoisen korkean suorituskyvyn klusterin rakentaminen ja optimointi

Tämän opinnäytetyön tarkoitus oli tutkia tietotekniikan klustereita, klusterointimalleja ja korkean suorituskyvyn järjestelmiä. Eri aloilla on viimeisien vuosien aikana huomattavasti lisääntynyt laskennallisen tehon tarve. Esimerkiksi lääkealalla simuloidaan molekyylien kehitystä eri sairauksissa ja kehitetään vasta-aineita. Tämä vaatii järeitä tietokoneklustereita, jotka ovat skaalattavissa miljardeja maksaviin tutkimuslaitoksiin.

Skaalattavuus on tärkeä ominaisuus klustereissa ja alkuun pääsee jo kahdella tietokoneella. Tässä opinnäytetyössä tarkastellaan klustereitten rajapintoja ja toimintamalleja lopuksi vertaillaan kuluttajatietokoneilla rakennetun klusterin ja NVIDIA DGX-1 ammattilaispalvelimen suorituskykyjä.

Kuluttajatietokoneilla rakennetun klusterin rakentaminen vaatii lukuisia IT-alan taitoja ja ymmärrystä. Suorituskyvyn optimoinnissa on haasteita kuluttajakomponenteilla mutta on hyviä tuloksia on mahdollista saavuttaa.

Asiasanat:

High-performance computing, Cluster, Slurm, MPI

Contents

List of abbreviations (or) symbols **8**

1 Introduction **9**

2 Computer Clusters **10**
2.1 Overview of Computer Clustering 10
2.2 History of Computer Clustering 11
2.2.1 The first supercomputers 11
2.2.2 The Cray Era 11
2.2.3 The Cluster Era 12
2.2.4 The Conventional Era 12
2.3 The GPU Era 12

3 Computer Cluster Hardware **14**
3.1 Components of a Computer Cluster 14
3.1.1 Cluster nodes 15
3.1.2 Connections 15

4 Computer Cluster Software **16**
4.1 Slurm 16
4.2 MPI 17
4.3 Ansible 19
4.4 Docker 19
4.5 Singularity 20

5 Methodology / Methods **21**
5.1 Building the Cluster 21
5.2 Configuring the cluster 25
5.3 DGX-1 27
5.4 Benchmarking the clusters 27
5.4.1 Setting up LINPACK on the cluster 28
5.4.2 Settings up LINPACK on DGX-1 28

5.5 Running LINPACK Benchmark 29

6 Results 30

6.1 HPL Results DGX-1 30

6.2 HPL Results Lab Cluster 33

7 Discussion 36

8 Conlusion 38

References 39

Appendices

Appendix 1. Finding CPU Cores / Rank for the DGX.

Appendix 2. Finding optimal NB for the DGX-1.

Figures

Figure I. HPC Cluster architecture (Iowa State University n.d.b) 14

Figure II. Slurm components (SchedMD 2021c). 17

Figure III. Point-to-Point Message Passing Interface (New Mexico State University 2021). 18

Figure IV. Docker Architecture (Docker n.d.). 20

Figure V. DeepOps inventory file. 23

Figure VI. Selecting roles for the listed nodes. 24

Figure VII. Netplan configuration file for the master node. 25

Figure VIII. Screenshot of the /etc/hosts file. 26

Tables

Table I. Hardware specifications for compute nodes. 21

Table II. Compute node versions. 26
Table III. NVIDIA DGX-1 Hardware. 27
Table IV. NVIDIA DGX-1 Software. 27
Table V. Finding CPU Cores / Rank for the DGX. 30
Table VI. Finding optimal NB for the DGX-1. 31
Table VII. Finding the optimal NB for the DGX-1. 32
Table VIII. Finding the optimal N value for DGX-1. 33
Table IX. Finding the optimal NB value for the home cluster. 34
Table X. Finding the optimal NB for 60000 N for the home cluster. 34
Table XI. Finding the optimal N value for maximizing FLOPS for the home cluster. 35

List of abbreviations (or) symbols

Abbreviation	Explanation of abbreviation (Source)
ARM	Advanced RISC Machines
CPU	Central Processing Unit
FLOPS	Floating point operations per second
GPGPU	General-purpose computing on graphics processing units
GPU	Graphics Processing Unit
HPC	High-Performance Computing
HT	Hyper-threading
IB	InfiniBand
MIMD	Multiple instruction, multiple data
MPI	Message Passing Interface
NFS	Network File System
OS	Operating system
SLURM	Simple Linux Utility for Resource Management
SSH	Secure Shell Protocol
WLAN	Wireless Local Area Network

1 Introduction

With the constant technological advancement in many fields of science, the demand for more computational power constantly grows. The evolution of Artificial Intelligence has been sped up by utilizing the massive amounts of data being generated by the public and private sectors. Tasks such as processing data, finding data patterns, and solving complicated problems by processing millions of gigabytes of data are impossible for humans. This is where high-performance computer clusters, also called “Supercomputers” appear. (Cremin Dash, and Huang, 2022; Fernández et al. 2019; Neumann & Kunkel 2020).

Modern supercomputers consist of thousands sometimes even a million smaller computers which communicate and act as a single unit. They are the size of a building and require high investment as well as upkeep to keep running. Most supercomputers are funded by governments, technology enterprises or different financiers. One of the peculiarities of these systems is scalability. Clusters can have a few powerful computers to do only a few tasks quickly or have a great number of slower computers do smaller tasks but at a really large scale. This provides the means to build a variety of computer clusters depending on the task at hand. (Palmer 2019; Skibba & Ramin 2021).

The purpose of this thesis is to investigate high-performance computing clusters, review different technological aspects of how a cluster is built and then finally build a home cluster. The home cluster will be benchmarked against a professional high-performance cluster following an evaluation of the whole process.

2 Computer Clusters

2.1 Overview of Computer Clustering

A computer cluster is a system that uses multiple processors in a single system and/or several systems to increase the data processing throughput. The computational power increases as the scale of the cluster grows. The easiest way to increase computational power is by adding more computers also called “nodes” to the cluster. There are no set sizes of clusters and while basic home clusters can consist of a few processor cores, the modern supercomputers have hundreds of thousands of cores. (Cook 2013). The TOP500 list has been tracking the performance of the supercomputers since 1993, benchmarking them and ranking them accordingly (TOP500 List 2022a).

There are several types of different nodes in a computer cluster. The “compute” nodes, also called “worker nodes ” are usually equipped with hardware to maximise computational power. Worker nodes are based on either Central Processing Units(CPU) or Graphics Processing Units (GPU). Most computing clusters are CPU-based, but there has been a surge in GPU-based clusters in recent years due to the increase in performance proportioned to the cost. Also, the development of different interfaces and hardware utilizing GPU cores to do non-graphical computational tasks have greatly affected the popularity of GPGPU-based systems. GPGPU (general purpose graphic processing unit) has become a popular solution for data processing in high-performance computing. (Rodriguez et al. 2020).

A Cluster can also be heterogeneous utilizing both CPU and GPU based hardware. These systems use advanced parallelization techniques to figure out which tasks can be performed efficiently on different kinds of cores (Karpusenko 2015). These heterogeneous systems have been a common trend in the past years and a direction for both rofessional and desktop computers. (Cook 2013; Ho 2015).

2.2 History of Computer Clustering

2.2.1 The first supercomputers

The history of computer clusters and supercomputing are tightly tied together with the development of early networking as the driving force behind it was to connect computers to increase resources and be able to communicate between computers. The first packet switching transmission techniques were invented during the cold war by RAND corporation in the late 1950s. (Baran 2002). The first use of the word “Supercomputing” dates back to March 1, 1929, when an article was published in the New York World was released. (Cruz FD 2001; New York World 1929).

2.2.2 The Cray Era

The first supercomputer by modern standards was built in 1965 by Seymour Cray, who is considered to be the “father of supercomputing”. Seymour Cray worked for Control Data Corporation (CDC). Cray model CDC 6600 used instruction level parallelism (ILP) to deliver messages across multiple units simultaneously. (Chen et al. 2009). CDC 6600 sold 100 units within a year for $8 million each. A few years later Cray developed the successor CDC 7600 which could manage 15 million instructions per second. In 1972, after running into problems with funding, in a market which was almost entirely controlled by IBM, Cray decided to start his own company called Cray Research, Inc. Four years later, in 1976, Cray 1 was released. Cray 1 could execute up to 240 million calculations per second and be mainly used in performing tasks for the U.S. Department of defense, meteorological institutes and various scientific and engineering faculties. (Hannan 2008). Cray 2 supported up to 8 CPUs and held the title for the fastest supercomputer until 1990 with a performance of 1.9 gigaFLOPS. Cray 2 also used a more common software, Unix System V, which meant that it was in the market for universities and corporations. (Anthony 2012).

2.2.3 The Cluster Era

The cluster era began in 1990 when performance was gained by using parallelism instead of a single bus to transfer data. The Japanese introduced several computers like the NEC SX-3 and Hitachi SR2201 with the latter having up to 2048 cores and a performance of 600 gigaFLOPS. (Anthony 2012). With the introduction of MIMD (Multiple instruction, multiple data) techniques the clustering took off. MIMD allowed to run independent separate programs simultaneously on different units. These units would communicate to exchange data or synchronize their operations. (Pfister 2008). This allowed the processing units in a system to compute different tasks and run different programs while still being in the same system and communicating with each other using the MPI (Message passing interface). By the end of the 20th century, ASCI Red held the position of the world's fastest supercomputer having 9000 computing nodes, 12 terabytes of disk storage and reaching peak performance of 1.80 teraFLOPS. (Mattson & Henry 1998).

2.2.4 The Conventional Era

During the following years, several different HPC clusters were built for simulations in different fields. Earth Simulator cluster was built in 2002 to simulate global climate in the atmosphere and the ocean. Capable of a peak performance of 40 TFlops with 640 compute nodes running on 5120 CPU cores. (JAMSTEC-Japan). IBM ASCI-Blue Gene was introduced in late 2004. Focusing on maximizing space and energy efficiency by having tiny processors. IBM was able to fit 1024 dual-processor nodes in a single rack yet the system had computational power ten times of the Earth simulator. (IBM).

2.3 The GPU Era

The GPU Era, also called the hybrid era started with the introduction of GPU- and clusters utilizing both GPU and CPU cores. Energy consumption concerns

gained importance at the beginning of the 21st century. GPU-based systems initially consumed significantly more power but the rise in computational power reduced the overall energy consumption. (Enos et al. 2010). The TOP500 list also introduced the Green500 list to measure the energy efficiency of clusters. (TheGreen500).This also bloomed the popularity of using ARM (Advanced RISC Machines)-based systems in HPC clusters. Having low energy consumption and cost can deliver competitive performance. (Rajovic et al. 2014). This was showcased by the Fugaku Supercomputer topping the TOP500 list in 2020 running on 7,630,848 ARM-based A64FX cores (TOP500 List 2022b).

3 Computer Cluster Hardware

3.1 Components of a Computer Cluster

Computer clusters consist of several components that make up a cluster. Specific components can vary depending on the design of the cluster. The core design is to have one or more compute nodes and one master node acting as a controller handling the jobs and login services. Single node clusters have all the roles being performed by the same node. A cluster can also have nodes for special purposes to isolate some operations on independent nodes for example data-transfer nodes as seen in Figure I. (Iowa State University a).

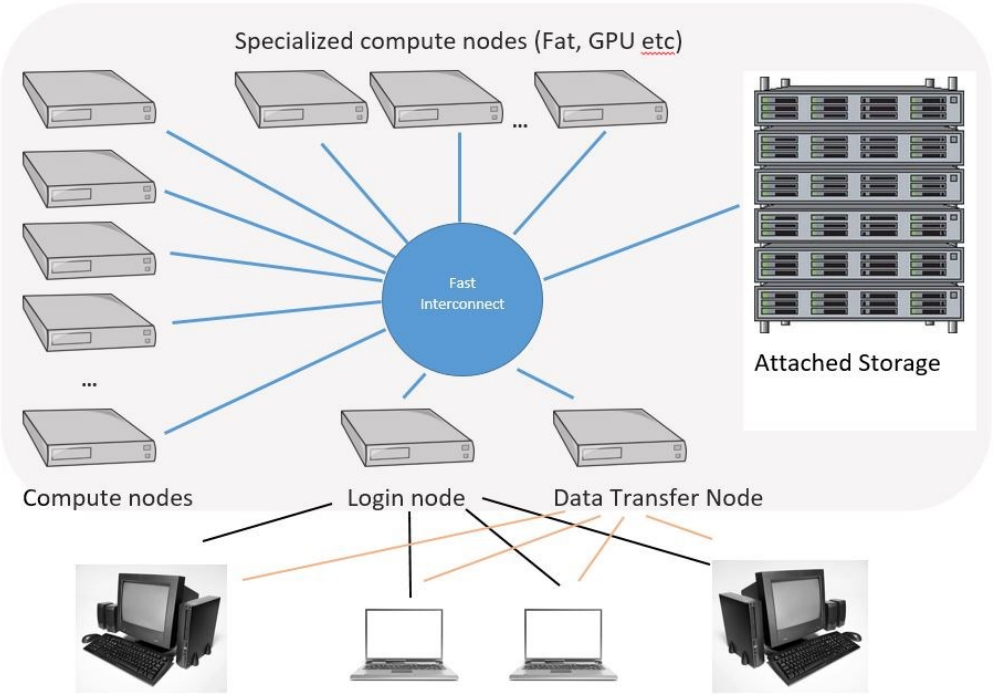

Figure I. HPC Cluster architecture (Iowa State University b).

The most common type of storage is to have a Network File System (NFS) for file sharing across the cluster. This makes it easier for all the nodes to access files without having to store data locally running into issues like file ownership and accessibility. (Iowa State University c).

3.1.1 Cluster nodes

A node is a single system in an HPC cluster. The node runs independently with its independent instance of an operating system (OS). Commonly variations of the Linux-based server OS are used on cluster nodes. Graphical user interfaces take up too much of the system resources and are highly unnecessary as no input is given to the node locally. (Iowa State University d).

In a cluster setup, the compute node doesn’t perform anything by itself but receives instructions from the master node to perform tasks through various scripts or commands. Alternatively, a master node (resource management node) can assign a user to a computer node with the requested resources to perform tasks through an SSH connection. (Linux New Media USA).

3.1.2 Connections

Most commonly clusters use variations of the Mellanox (NVIDIA) InfiniBand computer networking communications standard and Ethernet (NVIDIA Corporation 2020). InfiniBand (IB) is optimised for use in HPC clusters, having very high throughput and low latency (Mellanox Technologies 2018). Ethernet is more commonly used on smaller systems as the availability is high and cost is low (Linux New Media USA).

4 Computer Cluster Software

4.1 Slurm

SLURM (Simple Linux Utility for Resource Management) is a job scheduler used in many systems ranging from home clusters to the world's fastest supercomputers. The main task of a resource manager is to allocate users the resources they request for their jobs. While broad systems can have thousands of users log in and request resources for their jobs, it is impossible to manually keep track and schedule the use and availability of system resources across hundreds of nodes. Slurm automatically tracks nodes, their status, available system resources and the job queue assigning nodes that are available to compute the tasks waiting. Depending on the system equipment a manager can handle several hundred jobs a second. (Leonenkov & Zhumatiy 2015).

Slurm is built around its main process called slurmctld or Slurm controller daemon (Figure II). The daemon is responsible for allocating resources on a compute node when a user makes a request. The user can specify the number of resources needed by specifying duration, cores, memory, bandwidth etc. When the requested resources are available the user gains access to the computer node. The user requesting the resources can also submit a job containing an input script to the work queue which enables the user to not be present when resources are available. Every compute node also has a Slurm daemon process running on the system. This process enables commands to be run on the node anywhere in the system. (SchedMD 2021a; SchedMD 2021b).

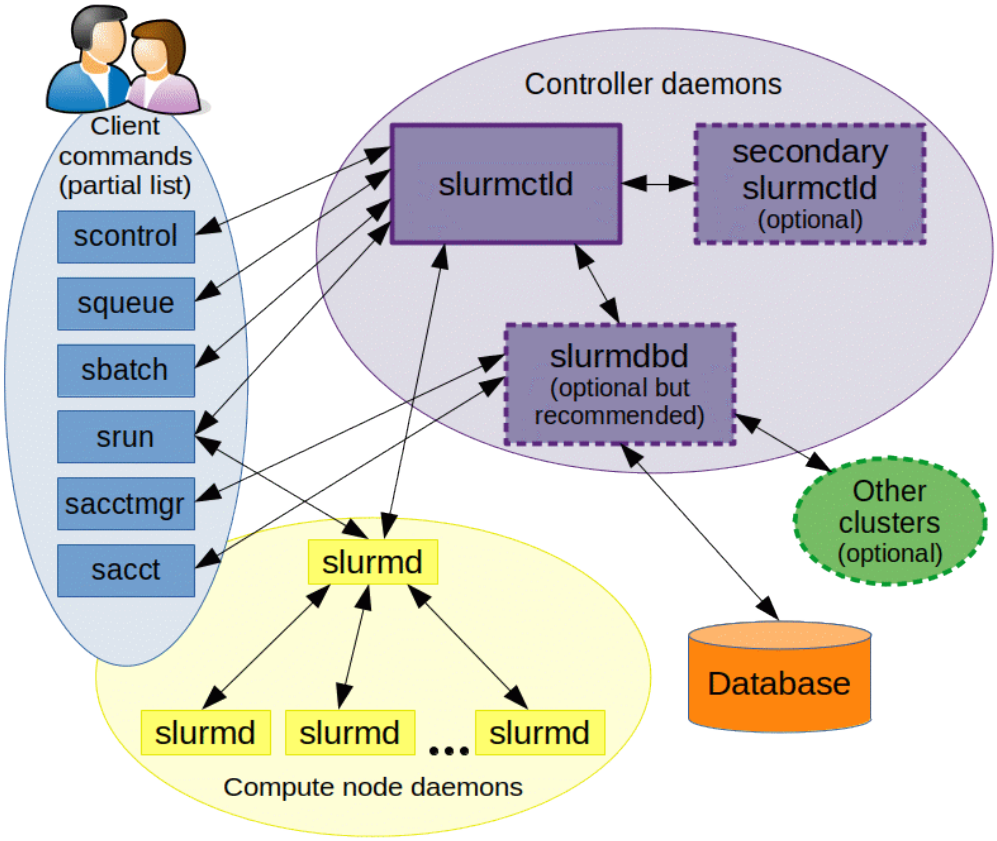

Figure II. Slurm components (SchedMD 2021c).

Slurm also has a wide variety of plug-in modules available. This makes SLURM highly configurable for various use-cases. Available modules enable configuration for software, different kinds of policies and architecture. (Reuther et al. 2018). The module plug-in system enables users to run environments with elevated privileges on their local user. (Słonina, Goździewski & Migaszewski 2015).

4.2 MPI

Parallel processing is a way for the cluster to utilize its computing power by dividing a given task into smaller subtasks. Subtasks can then be given out evenly to several nodes instead of having one node do the whole task. A similar way that multi-core processors work in modern consumer computers. (Buyya, Vecchiola & Selvi 2013).

In recent year, as the CPU and GPU based HPC systems have developed and the performance of these systems are skyrocketing the need for improved parallelization systems have been in great demand. (Sung et al. 2005; Wang, Li & Gao 2021; He et al. 2022). Especially GPU-based clusters, a single GPU has thousands of cores (NVIDIA Corporation 2019). The challenge with modern multicore systems is to utilize the available hardware. This is where MPI (message-passing interface) comes in. MPI enables cores in the system to pass messages between the nodes in a cluster and between processes enabling infinite scalability in a cluster. (Sterling, Anderson & Brodowicz 2018).

Point-to-point communication is the most used type of MPI method. The communication can either be synchronous or asynchronous. Synchronous, also called blocking point-to-point communication sends a message to another process and waits until the receiver has completely and correctly received the message before continuing work. Non-blocking (Asynchronous) point-to-point communication continues to work after the message has been sent and doesn't wait. (Sterling, Anderson & Brodowicz 2018).

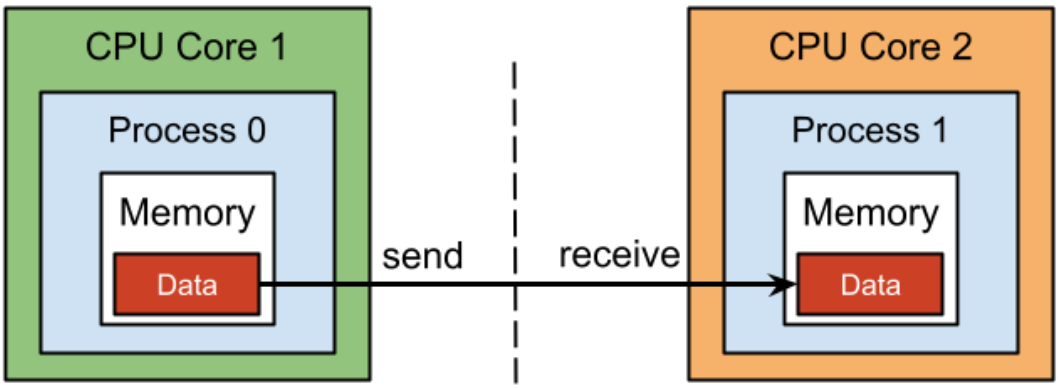

Figure III. Point-to-Point Message Passing Interface (New Mexico State University 2021).

MPI is a community-led development of application programming interfaces that started in 1992. After overcoming some challenges during the early days of

development MPI has proven to be a powerful and flexible programming interface. It is currently by far the most popular library used in HPC systems. (Sterling, Anderson & Brodowicz 2018).

4.3 Ansible

Ansible is a software tool used for configuration and software deployment management. Ansible allows for customized automated installation of multi-node systems by using the YAML language. Ansible connects to the system or nodes over an SSH connection and executes modules. HPC clusters can easily be deployed using Ansible, allowing multiple nodes to be installed at the same time. (Red Hat 2020a).

Ansible uses “playbooks” written in YAML language, which can be edited in any text editor. The playbooks hold all the settings and configurations needed for the setup. The inventory textfile keeps track of all the systems, hostnames and IP addresses for deplying modules on. (Red Hat 2020b).

4.4 Docker

Docker is a virtualization environment that works on the operating system level. Docker uses containers to isolate the processes while running on the system kernel. (Lane-Walsh et al. 2021). Virtualization on the operating system level enables the containers to run different versions of the operating system, as well as hardware and system firmware. This is also called containerization, a light version of a virtual machine. Applications and processes are built into a docker image, which is then run on the system, as an executable. The workflow can be seen in Figure IV. Images hold all the necessary files and codes for the image to run. Docker containers are instances of these images being run. (Morris et al. 2017)

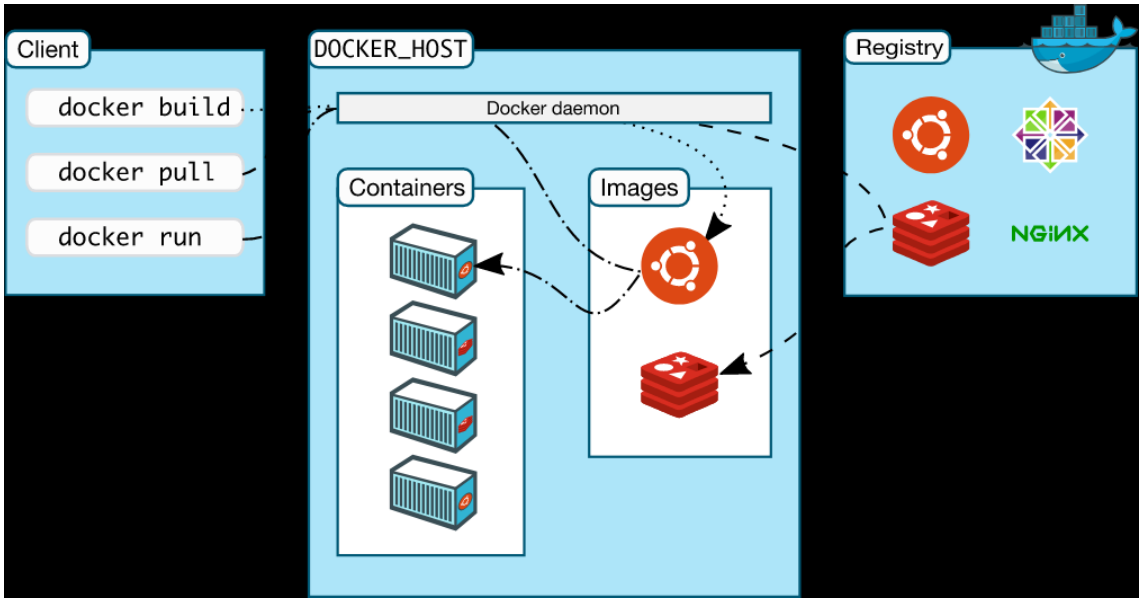

Figure IV. Docker Architecture (Docker).

4.5 Singularity

Singularity is much similar to the docker but with a focus on HPC clusters. Singularity enables the user to build containers from the image and save them to a file, which then can be exported to run on clusters. These containers can also be run in sandbox mode making it possible to edit the files without having to rebuild the container each time. Singularity also supports NVIDIA's CUDA frameworks making it easy to run libraries GPU-enabled. (Sylabs Inc 2019).

5 Methodology / Methods

5.1 Building the Cluster

The first step is to install Ubuntu 20.04 LTS version on all six computer nodes used in this project. Ubuntu 20.04 LTS was used as the OS because of the wide support for all applications needed in this project. The updated version also has support for newer Nvidia drivers and the CUDA toolkit needed for the GPU-accelerated tests.

The compute nodes will have the hardware listed in Table I. The main component is two NVIDIA RTX 2080 Ti's doing the computational power in the benchmarks. The graphic cards are connected via an NVlink.

Table I. Hardware specifications for compute nodes.

Hardware	**Ubuntutrt01-05**
Motherboard	Gigabyte Z390 AORUS PRO WIFI.
CPU	Intel Core i9-9900k @ 3600 Mhz, 8 cores.
GPU	2x NVIDIA GeForce RTX 2080 Ti. Turing TU102. Boost clock: 1650 Mhz. 11264 MB GDDR6.
Memory	64 GB, Corsair DDR4 4x 16 GB 2133 MHz (XMP disabled).
HDD	Kingston SNVS500G, 500GB NVMe PCIe M.2 2100/1700 MB/s.
Network	1 Gbps, Intel Ethernet Connection I219-V.
PSU	Corsair RM1000x 1000W.

Ubuntu installation doesn't require any extras to be installed. Standard installation on an empty partition. The hard disk used is empty but the system is installed with Windows 10, meaning that a dual-boot setup is required. To boot

into Linux, change the BIOS boot order to prioritize Linux, allowing Windows to be selected from the menu if needed.

The computer nodes are named by ranging from ubuntutrt01-05. These compute nodes are equipped with two NVIDIA RTX2080ti for computational power. The sixth node is a Dell Laptop. The Dell laptop was named “master01” and it will serve as the controller node handling the slurm controller daemon as well as serving as the login node for the cluster. A sudo user “emil” is also created for all the nodes. This sped up the deployment of SLURM.

Once Ubuntu has been installed on all the nodes and booted up the deployment of SLURM can begin. For the deployment, a separate laptop with Windows Subsystem for Linux (WSL) will be used. This computer will function as a provisioning machine, separate from the cluster and will only be used during the deployment. WSL run Ubuntu 20.04 on the. The first step is to clone or download the deepops repository from GitHub. After the repository is extracted swap to the deepops folder by typing

> cd deepops

Next, install ansible and its prerequisites. This is done by running the pre-installed script, which is located inside the scripts folder.

> ./scripts/setup.sh’

This will install ansible with all the required prerequisites. Depending on the system there might be a requirement to set up passwordless access by copying SSH keys to the target systems. This allows Ansible to deploy SLURM without the use of a password. SSH keys can be copied over by running the following commands:

>ssh-keygen

> ssh-copy-id <username>@<host>

“Username” is the local username on the target computer and “host” the IP of the target computer.

When Ansible is set up and optionally SSH-keys have been copied over to the target systems, edit the inventory for SLURM. The inventory file can be found inside the ‘config’ folder. Use any text editor to edit the inventory file.

>“sudo vi config/inventory”

Insert the nodes deployable nodes under the [all] tag. Specify hostnames and addresses. If a hostname is different from the hostname written in the inventory file, Ansible will try to rename it during deployment. An example can be seen in figure V.

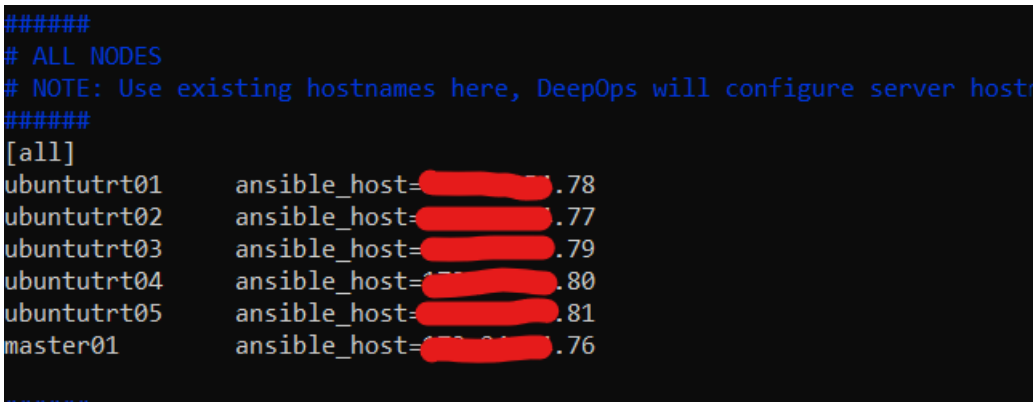

Figure V. DeepOps inventory file.

Roles must also be specified for the inserted nodes. Master and compute nodes get completely different deployment packages installed and thus it must be done correctly. The roles can be selected by listing the hostnames mentioned under the ”all” tag. For this deployment, master01 is inserted as the ”slurm-master” node and the rest as compute nodes under ”slurm-node”, shown in Figure VI.

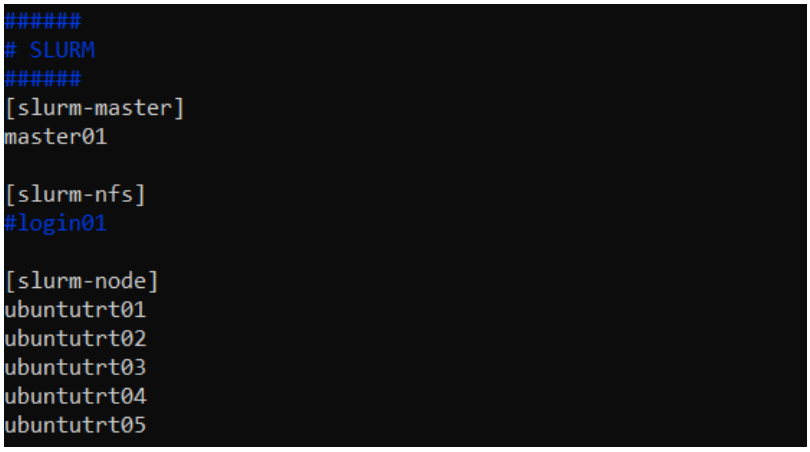

Figure VI. Selecting roles for the listed nodes.

Once the inventory is set up, save the file and optionally move to modify the “config/group_vars/slurm-cluster.yml“ configuration file. The file allow to select the modules that are being installed with SLURM. An NFS server can also be installed, depending on the setup of the cluster. For this cluster, the only change was done to “install NFS server:” selecting False. The cluster will only operate with local files although it is highly recommended to use an NFS for sharing files on all the nodes.

Once all the configurations are done a verification test can be done by running ‘ansible all -m raw -a "hostname" ‘. If no SSH keys were copied, use commands “-u <user>” and “-K” to login as sudo when verifying the connection. Verification of the completed test will show within a few seconds, also any error messages will be shown here.

If the connection is successful on all target nodes the deployment of SLURM by typing :

> ansible-playbook -u <user> -K -l slurm-cluster playbooks/slurm-cluster.yml

after which, there is a prompt for the sudo password. Note that the input username has to exist on the target machines. If the users don’t exist locally on every target machine, use the SSH key method explained above. Enter the password and continue. Slurm will now deploy on all the machines. This can take up to 15 minutes per node, depending on the number of modules and

settings. If any critical errors happen the installation will fail and display an error message. The nodes will automatically reboot during the installation.

5.2 Configuring the cluster

To make sure that the IP doesn't change from the DHCP, a static IP address have to be set up on the nodes. This can be done by editing the system network configuration. All Ubuntu versions 18.04 or later come with netplan pre-installed. The network adapter configuration file is usually created during the installation of the operating system. This can be found in the /etc/netplan/ folder. On this cluster, it's called "00-installer-config.yaml". There might also be separate configuration files for the WLAN adapter. The configuration file used for the controller node "master01" can be seen in Figure VII.

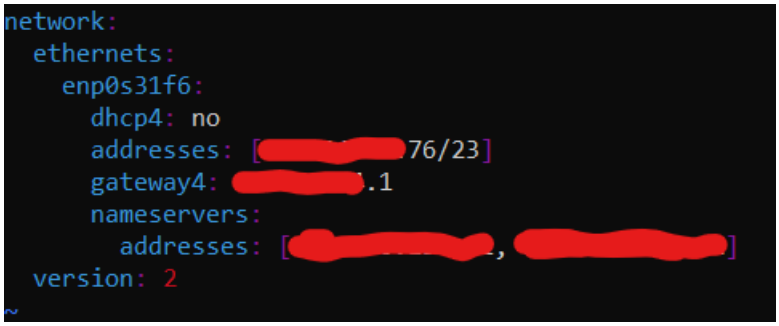

Figure VII. Netplan configuration file for the master node.

Nodes have been listed under the /etc/hosts file by Ansible and can be changed afterwards. These values must match on all nodes to be able to communicate. See Figure is a copy from the /etc/hosts file used in this cluster. The list contains the machine IP and which network adapter is being used. Figure VIII is a screenshot from the /etc/hosts file located on all the nodes. Slurm daemons won't be able to communicate and will result in connection errors if the values are different on any node.

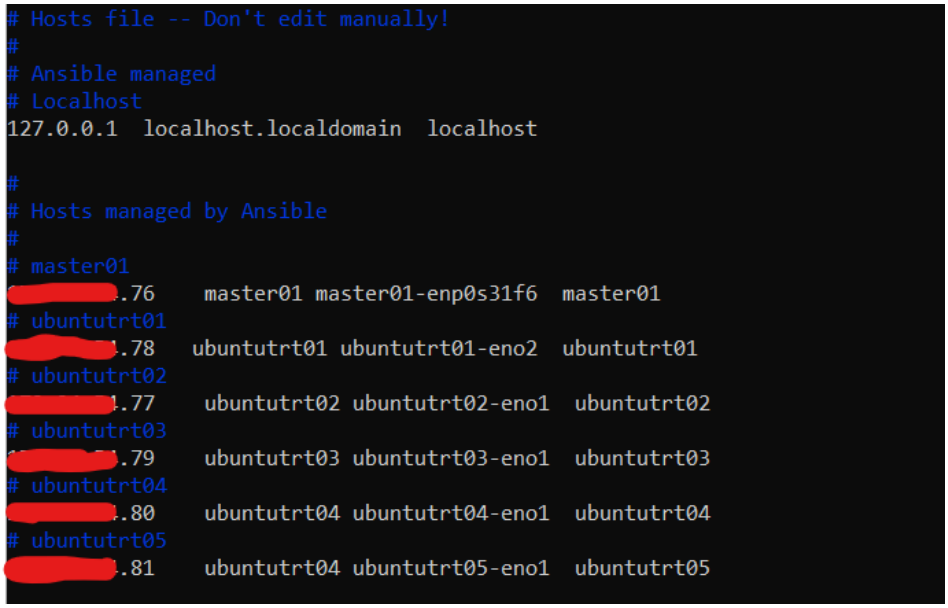

Figure VIII. Screenshot of the /etc/hosts file.

Once the installation has finished and everything is set up the user can connect through SSH to the master01-node and input

> sinfo

To confirm that everything is working, a list with the nodes should appear. It is also worth doing a system update to update any essential drivers. Slurm will automatically block some updates that could potentially break the functionalities like docker. Table II lists the versions of drivers, OS and kernel that were used during the benchmarking.

Table II. Compute node versions.

Software	**Ubuntutrt01-05**
OS	Ubuntu 20.04.4 LTS
Kernel	GNU/Linux 5.4.0-109-generic x86_64
CUDA	Version 11.6
GPU Drivers	510.73.05

5.3 DGX-1

The NVIDIA DGX-1 comes in several different hardware variations. The hardware used in this test can be seen in Table III. The DGX server is installed with Docker so the benchmarks were run as a container. The DGX is university-owned and no additional software was installed during the benchmarking.

Table III. NVIDIA DGX-1 Hardware.

Hardware	**NVIDIA DGX-1**
CPU	2 Intel Xeon E5-2698 v4 @ 2.20 GHz each with 20 cores, 40 threads
GPU	8 NVIDIA Tesla V100 SXM2 32GB (GV100GL)
Memory	512 Gb
Network	10 Gb ethernet

The DGX was pre-installed with the OS and drivers are shown in Table IV.

Table IV. NVIDIA DGX-1 Software.

Software	**NVIDIA DGX-1**
OS	Ubuntu 20.04.4 LTS
Kernel	GNU/Linux 5.4.0-109-generic x86_64
CUDA	Version 11.6
GPU Drivers	510.73.05

5.4 Benchmarking the clusters

High-Performance LINPACK Benchmark will be used as the benchmarking tool for the clusters. This benchmark is used for all the TOP500 clusters. Its popularity is gained due to being highly customizable and scalable on large

clusters. The benchmark measures the performance of solving a random dense linear system in double precision (64 bits) arithmetic. (Dongarra, J 2007).

5.4.1 Setting up LINPACK on the cluster

Singularity is used to run the container on a multi-node cluster. Docker doesn't work well with multi-node deployments. First, pull the container from the Nvidia NGC catalogue. This requires an NVIDIA developer's license to access it.

> singularity pull --docker-login hpc-benchmarks:21.4-hpl.sif docker://nvcr.io/nvidia/hpc-benchmarks:21.4-hpl

After the command is entered there is a prompt to enter the username and the password for the NVIDIA repository. After the login is entered the container will be pulled and singularity creates a .sif file in the location the command is run in. This needs to be done on every compute node since the cluster doesn't have a shared location set up. After the .sif file is created, it is ready to be used and doesn't require any further configuration.

5.4.2 Settings up LINPACK on DGX-1

The DGX-1 uses docker to run Linpack. The node used already had the credentials for the repository.

> docker pull nvcr.io/nvidia/hpc-benchmarks:21.4-hpl

The command also requires the node to have access to the NGC repository. The instructions are found under the NGC catalogue user settings.

> docker login nvcr.io

When the image is pulled it can be run in interactive mode as a container to see the contents.

> docker run --gpus all -it nvcr.io/nvidia/hpc-benchmarks:21.4-hpl

Copying over the HPL.dat file locally allows on the file. After exiting the container, run it again with:

> docker run -d <image id> sleep 300

The -d tag is used to run the container in debug mode and sleep 300 means that the container will run for 300 seconds before exiting. This gives us plenty of time to copy over the HPL.dat file. Use the following command:

> docker cp <containerId>:/file/path/within/container /host/path/HPL.dat

The HPL.dat file is used to configure the benchmarks.

5.5 Running LINPACK Benchmark

Configuring and optimizing the LINPACK benchmark requires testing a lot of parameters to find the optimal values for the system. These parameters are edited in the HPL.dat file. The file is mounted to docker and singularity when running the benchmarks. During these benchmarks values P, Q, coefficient matrix (N), the block size (NB) and also “CPU cores per rank” will be edited to find the optimal values. The product of P * Q should be equal to the number of processes run during the benchmark. With 5 nodes having 2 GPUs each there is a total of 10 processes. Since ethernet is being used and there’s only a single cable connecting the nodes the best way is to run a flat grid using for example P * Q = 1 * 10. (Petitet, A et al. 2018a; 2018b).

The number of CPU cores per rank should be the number of CPU cores available divided by GPUs on the node. For the cluster, there is 8 CPU cores with 2 GPUs so CPU cores per rank = 4 will be used. Running the benchmark with hyper-threading i.e. dividing the physical CPU cores into virtual ones. (NVIDIA Corporation 2022).

6 Results

6.1 HPL Results DGX-1

As seen in Table II the DGX-1 is equipped with eight Tesla V100. The theoretical max for the SXM2 32GB model is 7.8 TFlops for FP64 double-precision. During the benchmarking, the target is to find optimal values for the LINPACK benchmark and to utilize and maximize the performance of all eight GPUs.

The first step was to see if the number of CPUs assigned to each rank has a performance impact. The DGX-1 has 40 physical cores and 80 cores can be achieved by using HT (Hyper-threading). Since the benchmark itself doesn't utilize the CPU to solve the equations and will solely rely on the computational power of the GPUs, the impact of the CPU cores is not that important. The test was also run with P = 1 and Q = 8 values to see what the impact would be.

Looking at Table V it can be concluded that the highest speeds were achieved using 3 CPU cores per rank with having a grid of 2x4. On the more simplified 1x8 grid the highest results were pretty equal among 5, 6 and 8 cores. One could argue that running the tests on such low N values doesn't matter how many CPU cores are used. With further testing on different parameters,it was concluded that P = 2, Q = 4 and CPU Cores per rank = 5 produced the highest overall performance. These values will be used in all upcoming results if not stated else.

Table V. Finding CPU Cores / Rank for the DGX.

CPU Cores / Rank	**96 000 N P*Q = 2*4 (Tflops)**	**96 000 N P*Q = 1*8 (Tflops)**
1	17.46	7.38

(continue)

Table V. (continue).

2	21.66	10.38
3	**25.13**	11.51
4	23.81	12.18
5	24.03	**13.16**
6 (HT On)	21.71	**13.17**
7 (HT On)	21.82	12.82
8 (HT On)	21.57	**13.2**
9 (HT On)	21.39	13.14
10 (HT On)	17.70	12.29

Since the performance values are still quite low for the hardware configuration being used (NVIDIA Corporation 2019. Assumptions can be made that the NB and N values for the system are not optimal. Usually, the general rule is to utilize the GPUs by around 80 % according to LINPACK guidelines (Dongarra, J, 2007). The version being used is a LINPACK binary released by NVIDIA which has been optimized for their GPUs, aimed at the Ampere GPU series. The optimization should allow to utilize the GPUs closer to 100 percent.

Each value shown in Table VI is a separate benchmark that has passed correctly. The N value was raised step-by-step to find the optimal values. On lower N values the smaller NB is generally the fastest but on 140 000 Ns NB 192 starts to perform the fastest. On the higher end of the NB scale, 512 NB performs slower.

Table VI. Finding optimal NB for the DGX-1.

NB	**96 000 Ns (Tflops)**	**110 000 Ns (Tflops)**	**120 000 Ns (Tflops)**	**140 000 Ns (Tflops)**
64	**28.73**	31.29	32.24	34.31

(continue)

Table VI (continue)

128	28.44	**31.83**	**33.72**	37.08
192	27.68	30.93	32.31	**37.61**
256	25.00	29.29	30.76	35.89
384	24.76	23.40	29.76	35.03
512	20.98	21.90	26.19	29.88

Continuing to raise the N value there is a constant increase in performance. The VRAM utilized for 150000 Ns is between 22-23 GB, or roughly ~73 % VRAM utilization. 175000 N uses between 30-31 GB of VRAM on each GPU resulting in roughly 87 % of the available VRAM. Table VII shows that 128 and 192 NBs are still very close to each other. Increasing the N number using NB 192 constantly perform better as the utilization of VRAM rises.

Table VII. Finding the optimal NB for the DGX-1.

NBs	**150 000 Ns (Tflops)**	**160 000 Ns (Tflops)**	**170 000 Ns (Tflops)**	**175 000 Ns (Tflops)**
64	34.01	36.04	35.28	34.52
128	38.49	39.48	40.71	**41.35**
192	**38.61**	**40.59**	**40.79**	41.01
256	37.48	38.48	40.14	41.02
384	36.64	37.34	40.13	39.73
512	32.28	34.00	34.80	36.65

Finally, after plenty of different tests to find the optimal N value. Theorically the most optimal value of N should be dividable by NB, P and Q (Petitet, A et al. 2018). The results in Table VIII show that peak performance of 42.93 Teraflops was achieved using NB=192, P=2, Q=4 and N=180288.

Table VIII. Finding the optimal N value for DGX-1.

Ns	**NB 192**
180 000	41.39
180 096	42.67
180 288	**42.93**
180 384	42.67
180 500	42.06
180 864	41.93
182 000	42.35
182 500	42.56
183 000	42.44
185 000	40.36

6.2 HPL Results Lab Cluster

For the built cluster the N value has to be decreased drastically because each node is running on only 11 GB VRAM per GPU, totaling at 110 GB compared to 256 GB of VRAM on the DGX. The first benchmarks were run by randomly choosing 30000 N to set up some parameters for the benchmarking. Some benchmarks were also made using only one GPU per node (P * Q = 1*5) which resulted in a much lower result shown in Table IX. This confirmed that the benchmark is utilizing all the GPUs at some level. The highest result was achieved using NB 192 and on a 1x10 grid. 60000 Ns on NB 256 was also tested with a 1x5 grid resulting in a much higher performance, suggesting that the values for N used are too small for the cluster.

Table IX. Finding the optimal NB value for the home cluster.

NBs	**30 000 Ns P*Q = 1*10 (Gflops)**	**30 000 Ns P*Q = 1*5 (Gflops)**	**60 000 Ns P*Q = 1*5 (Gflops)**
192	**318.7**	-	-
256	318.6	175,4	**360,4**
384	314.8	-	-
512	310.3	-	-
640	303.7	-	-

For the next set of benchmarks, the N value was raised to 60000 to find the optimal NB value. Results shown in Table X indicate that the differences are quite small with 320 NB performing the fastest. Rerunning the benchmarks several times suggested that NB 384 is the best performing block size by a very small margin.

Table X. Finding the optimal NB for 60000 N for the home cluster.

NB	**60 000 Ns (Gflops)**	**Time**
64	567.0	258 s
128	606.6	237 s
192	622.8	231 s
256	627.6	229 s
320	**630.4**	**228 s**
384	630.2	**228 s**

With the optimal block size and grid, the next step was to find the N value. Table XI shows that increasing the N almost to the point of failure yielded the fastest performance. To confirm that 384 block size was ideal some additional

benchmarks were done by altering the values for P, Q and NB but caused no significant changes. The highest performance reached was 1181 GFlops by using 11000 N, 384 NB, P 1 and Q 10.

Table XI. Finding the optimal N value for maximizing FLOPS for the home cluster.

Ns	**384 NBs P*Q = 1*10 (Gflops)**	**384 NBs P*Q = 2*5 (Gflops)**	**192 NBs P*Q = 1*10 (Gflops)**
60 000	630.2	-	-
80 000	852.1	-	-
100 000	1071	-	-
105 000	1128	365.6	1124
107 520	1151	-	-
110 000	**1181**	-	-
112 500	Failure	-	-

7 Discussion

The results from the benchmarking on the DGX-1 were quite expected and surprisingly good. The optimization took a lot of work and the total amount of benchmarks run was around 400. Linpack benchmark is a complex tool that requires a lot of knowledge and understanding of the hardware and fundamentals of an HPC cluster. The thesis only covered the basic parameters used and didn't dive deeper into all the possible tuning possibilities available, some of which might have yielded better or different results. The biggest and the most time-consuming challenge was to learn and understand how it works, how the different parameters affect the result and how to use them for this particular system. Once a basic understanding is developed the benchmarking could be systematically done. NVIDIAs data sheet suggests that the Tesla V100 model used can achieve 7.8 Tflops of performance. Eight units would theoretically run at 62,4 Tflops. Achieving 42.93 Tflops was a positive surprise but with more room for improvement. A perfect score can never be achieved taking into consideration that the GPUs have to communicate and work together on the tasks given.

The self-built cluster had some real challenges and the results were poor. There are a lot of contributing factors to this which could be identified during the testing. The biggest challenges were the installation of all the required components for an MPI cluster. Getting the nodes to communicate with each other on a core level. Using DeepOps to deploy clusters is a massive benefit with the only hard part being the knowledge on the user side.

Other challenges were that the cluster used the environment 1 Gb ethernet connection that was wired through a switch. There was obvious bottlenecking since increasing the grid size would crash the performance. Most HPC clusters use 10 Gb ethernet or faster. Since the cluster was built in an environment that has daily use any changes could not be made during the testing. The second problem that occurred was thermal throttling. The cases and the design had previously been stress-tested which revealed that under heavy load the other

GPU would drop the utilization close to zero. This could also be seen by watching a live feed of the NVIDIA system management interface. Overall the differences are quite clear. The GPUs used in the self-built cluster are designed for gaming which is also why NVIDIA doesn’t show specifications of the mathematical performances. These units are designed for gaming and graphical tools, unlike the Tesla V100s in DGX-1 which were able to beat the performance in the benchmarks above by a factor of 40.

Going forward it would be interesting to see how the performance of GPUs designed for gaming stacks up against professional cards in the right environment. As for now, the results show clearly why workstations like DGX-1 exist. Small, scalable units that offices and companies can use for almost any tasks requiring high computational power that can be set up with relative ease.

8 Conlusion

The performance of a self-built cluster using components available to consumers have potential but is difficult to optimize for the performance. A lot of knowledge and skills in various subjects are required but it can be done with enough time and commitment.

Most of the work went to troubleshooting and learning the basics of every component, library and program and trying to find ways to make everything work together. Knowledge of every aspect is required. Clustering in general is still an unknown field for many which can be seen as there is a very limited amount of information available, especially for high-performing larger clusters which are operated by professionals for a clear reason.

During the thesis, an interest in ARM-based clusters rose. Many of the TOP500 clusters utilize them. The idea behind building a cluster with ARM-based processors is interesting as they are cheap, have low power consumption and are small in size. This could also be a potential introduction course for people interested in the technology behind clustering as a wide variety of skills can be used during the process.

References

Anthony, S 2012. *The history of supercomputers,* viewed 2 May 2022, https://www.extremetech.com/extreme/125271-the-history-of-supercomputers/4.

Baran, P 2002. 'How packet switching works', *Journal of the Franklin Institute*, Volume 339, Issue 3, pp. 265-275, Viewed 2 May 2022, https://doi.org/10.1016/S0016-0032(01)00042-4.

Buyya, R, Vecchiola, C & Selvi, S.T 2013. 'Chapter 2 - Principles of Parallel and Distributed Computing', *Mastering Cloud Computing*, pp. 29-70, viewed 30 April 2022, https://doi.org/10.1016/B978-0-12-411454-8.00002-4.

Chen, S, Lin, G, Hsiung, P & Hu, Y 2009. *Hardware Software Co-Design of a Multimedia SOC Platform,* pp. 70, viewed 2 May 2022, https://books.google.fi/books?id=OXyo3om9ZOkC&lpg=PA70&hl=fi&pg=PA70#v=onepage&q&f=false.

Condia, JE.R, Du. B, Reorda, MS & Sterpone, L, 2020, 'FlexGripPlus: An improved GPGPU model to support reliability analysis', *Microelectronics Reliability*, Vol 109, viewed 28 April 2022, https://www.sciencedirect.com/science/article/pii/S0026271419307978.

Cook, S 2013. 'Chapter 1 – A History of Supercomputing', *Applications of GPU Computing Series*, pp. 1-19, viewed 26 April 2022, https://www.sciencedirect.com/science/article/pii/B9780124159334000016.

Cremin, CJ, Dash S & Huang, X 2022. 'Big data: Historic advances and emerging trends in biomedical research', *Current Research in Biotechnology*, Volume 4, pp. 138-151, viewed 2 May 2022, https://doi.org/10.1016/j.crbiot.2022.02.004.

Cruz, FD 2001, 'The Columbia Difference Tabulator 1931'. *Columbia University Computing History*, viewed 2 May 2022, http://www.columbia.edu/cu/computinghistory/packard.html.

Docker, *Docker Architecture,* figure, viewed 1 June 2022, https://docs.docker.com/engine/images/architecture.svg.

Dongarra, J 2007, *Frequently Asked Questions on the Linpack Benchmark and Top500,* viewed 10 June 2022, http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html.

Enos, J, Steffen, C, Fullop, J, Showerman, M, Shi, G, Esler, K, Kindratenko, V, Stone, JE. & Phillips, JC. 2010, 'Quantifying the impact of GPUs on performance and energy efficiency in HPC clusters', *International Conference on Green Computing*, pp. 317-324, viewed 10 June 2022, https://ieeexplore.ieee.org/abstract/document/5598297.

Fernández, Á, Fernández, C, Miguel-Dávila, JA, Conde, MA & Matellán, V 2019, 'Supercomputers to improve the performance in higher education: A review of the literature', *Computers & Education*, Volume 128, pp. 353-364, viewed 2 May 2022, https://doi.org/10.1016/j.compedu.2018.10.004.

Hannan, C 2008, 'Wisconsin Biographical Dictionary'*, Volume 92 / United States Biographical Dictionary*, pp. 83-84, viewed 2 May 2022, https://books.google.fi/books?id=V08bjkJeXkAC&lpg=PA83&hl=fi&pg=PA82#v=onepage&q&f=false.

He, X, Wang, K, Feng, Y, Lv, L & Liu, T 2022, 'An implementation of MPI and hybrid OpenMP/MPI parallelization strategies for an implicit 3D DDG solver*'. Computers & Fluids*, viewed 1 May 2022, https://www.sciencedirect.com/science/article/pii/S0045793022001086.

Heath, M.T., Ranade, A & Schreiber, R.S. 1998, *Algorithms for Parallel Processing,* pp. 323, https://books.google.fi/books?id=zo61nbirb_gC&lpg=PA323&hl=fi&pg=PA323#v=onepage&q&f=false.

Ho, KL 2015, 'Performance Analysis of CPU-GPU Cluster Architectures'. *American Journal of Networks and Communications*, Vol. 4, No. 3, pp. 67-74, viewed 28 April 2022, https://article.sciencepublishinggroup.com/html/10.11648.j.ajnc.20150403.18.html.

IBM, *Blue Gene,* viewed 10/06/2022, https://www.ibm.com/ibm/history/ibm100/us/en/icons/bluegene/.

Iowa State University a, *What is an HPC cluster*, viewed 10 June 2022, https://www.hpc.iastate.edu/guides/introduction-to-hpc-clusters/what-is-an-hpc-cluster.

Iowa State University b, *HPC cluster,* figure, viewed 10 June 2022, https://www.hpc.iastate.edu/sites/default/files/uploads/HPCHowTo/HPCCluster.JPG.

Iowa State University c, *Cluster Storage*, viewed 10 June 2022, https://www.hpc.iastate.edu/guides/introduction-to-hpc-clusters/cluster-storage.

Iowa State University d, *Logging to a cluster,* viewed 10 June 2022, https://www.hpc.iastate.edu/guides/introduction-to-hpc-clusters/logging-to-a-cluster.

JAMSTEC-Japan, *System Overview*, viewed 10/06/2022, https://www.jamstec.go.jp/es/en/es1/system/performance.html.

Karpusenko, V 2015, 'High Performance Parallelism Pearls', *Multicore and Many-Core Programming Approaches*, pp. 425-441, viewed 28 April 2022, https://www.sciencedirect.com/science/article/pii/B978012802118700025X?via%3Dihub.

Lane-Walsh, S, Stillerman, J, Santoro, F & Fredian, T 2021, 'Introduction to MDSplus using Docker'*, Fusion Engineering and Design*, Volume 165, viewed 1 May 2022, https://doi.org/10.1016/j.fusengdes.2020.112121.

Leonenkov, S & Zhumatiy, S 2015, 'Introducing New Backfill-based Scheduler for SLURM Resource Manager', *Procedia Computer Science*, Volume 66, pp. 661-669, viewed 29 April 2022, https://www.sciencedirect.com/science/article/pii/S1877050915034249.

Linux New Media USA, *The Fundamentals of Building an HPC Cluster,* viewed 10 June 2022, https://www.admin-magazine.com/HPC/Articles/Building-an-HPC-Cluster.

Mattson, TT. & Henry, G 1998, *An Overview of the Intel TFLOPS Supercomputer*, http://www.ai.mit.edu/projects/aries/course/notes/ascii_red.pdf.

Mellanox Technologies 2018, *Designing an HPC Cluster with Mellanox InfiniBand Solutions,* viewed at 10 June 2022, https://mymellanox.force.com/mellanoxcommunity/s/article/designing-an-hpc-cluster-with-mellanox-infiniband-solutions.

Morris, D, Voutsinas, S, Hambly, N.C. & Mann, R.G. 2017, 'Use of Docker for deployment and testing of astronomy software', *Astronomy and Computing*, Volume 20, pp. 105-119, viewed 1 May 2022, https://doi.org/10.1016/j.ascom.2017.07.004.

Neumann, P & Kunkel, J 2020. 'Chapter 7 - High-Performance Techniques for Big Data Processing', *Knowledge Discovery in Big Data from Astronomy and Earth Observation*, pp. 137-158, viewed 2 May 2022, https://doi.org/10.1016/B978-0-12-819154-5.00017-5.

New Mexico State University 2021, *Message passing interface,* figure, viewed at 10 June 2022, https://hpc.nmsu.edu/discovery/_images/mpi/mpi-point-to-point_communication.png.

New York World 1929, *Super Computing Machines Shown,* viewed 2 May 2022, http://www.columbia.edu/cu/computinghistory/statlab-clipping.jpg.

NVIDIA Corporation 2019. *NVIDIA DGX-1 datasheet,* https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf.

NVIDIA Corporation 2020, *Accelerating the Top500 Supercomputers*, viewed at 10 June 2022, https://network.nvidia.com/sites/default/files/pdf/solutions/hpc/TOP500-JUNE-2020.pdf.

NVIDIA Corporation 2022, *NVIDIA HPC-Benchmarks 21.4*, viewed 10 June 2022, https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks.

Palmer, J 2019, 'More Super Supercomputers', *Engineering*, Volume 5, Issue 3, pp. 357-358, https://doi.org/10.1016/j.eng.2019.04.007.

Petitet, A, Whaley, R.C., Dongarra, J & Cleary, A 2018a, *HPL Frequently Asked Questions,* viewed 10 June 2022, http://www.netlib.org/benchmark/hpl/faqs.html.

Petitet, A, Whaley, R.C., Dongarra, J & Cleary, A 2018b, *HPL Tuning,* viewed 10 June 2022, http://www.netlib.org/benchmark/hpl/tuning.html.

Pfister, G 2008. *Larrabee vs. Nvidia, MIMD vs. SIMD*, http://perilsofparallel.blogspot.com/2008/09/larrabee-vs-nvidia-mimd-vs-simd.html.

Rajovic, N, Rico, A, Puzovic, N, Adeniyi-Jones, C & Ramirez, A 2014, 'Tibidabo: Making the case for an ARM-based HPC system', *Future Generation Computer Systems*, vol. 36, pp. 322-334, viewed 10 June 2022, https://doi.org/10.1016/j.future.2013.07.013.

Red Hat, Inc 2020a, *How Ansible works,* viewed 30 April 2020, https://www.ansible.com/overview/how-ansible-works?hsLang=en-us.

Red Hat, Inc 2020b, *Application deployment,* viewed 30 April 2020, https://www.ansible.com/use-cases/application-deployment?hsLang=en-us.

Reuther, A, Byun, C, Arcand, W, Bestor, D, Bergeron, B, Hubbell, M, Jones, M, Michaleas, P, Prout, A, Rosa, A & Kepner, J (2018). 'Scalable system scheduling for HPC and big data'. *Journal of Parallel and Distributed Computing*, Volume 111, Pages 76-92. viewed 30 April 2022, https://www.sciencedirect.com/science/article/abs/pii/S0743731517301983?via%3Dihub.

SchedMD 2021a, *Overview,* viewed 29 April 2022, https://slurm.schedmd.com/overview.html.

SchedMD 2021b, *Quick Start User Guide,* viewed 29 April 2022, https://slurm.schedmd.com/quickstart.html.

SchedMD 2021c, *Slurm Components,* figure, viewed 29 April 2022, https://slurm.schedmd.com/arch.gif.

Skibba, R 2021, 'Japan's Fugaku Supercomputer Crushes Competition, But Likely Not for Long', *Engineering*, Volume 7, Issue 1, pp. 6-7, viewed 2 May 2022, https://doi.org/10.1016/j.eng.2020.12.003.

Słonina, M, Goździewski, K & Migaszewski, C 2015, 'Mechanic: The MPI/HDF code framework for dynamical astronomy', *New Astronomy*, Volume 34, pp. 98-107, viewed 30 April 2022,

https://www.sciencedirect.com/science/article/abs/pii/S1384107614000864?via%3Dihub.

Sterling, T, Anderson, M & Brodowicz, M 2018. 'Chapter 8 - The Essential MPI', *High Performance Computing*, pp. 249-284, viewed 1 May 2022, https://doi.org/10.1016/B978-0-12-420158-3.00008-3.

Sung, M.Y., Whang, SM., Yoo, Y., Kim, NJ., Park, JS & Choi, W 2006, 'Parallel Processing for Reducing the Bottleneck in Realtime Graphics Rendering', *Advances in Multimedia Information Processing - Lecture Notes in Computer* Science, vol 4261, viewed 30 April 2022, https://link.springer.com/chapter/10.1007/11922162_107.

Sylabs Inc 2019, *Introduction to Singularity,* viewed 10 June 2022, https://sylabs.io/guides/3.5/user-guide/introduction.html.

TheGreen500, *Green500*, viewed 10/06/2022, https://web.archive.org/web/20160620115925/http:/www.top500.org/green500/.

TOP500 List 2022a, *About,* viewed 01 May 2022, https://www.top500.org/project/.

TOP500 List 2022b, *Supercomputer Fugaku,* viewed 10 June 2022, https://www.top500.org/system/179807.

Wang, Y, Li, W & Gao, J 2021, '*A* parallel sparse approximate inverse preconditioning algorithm based on MPI and CUDA', *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, Volume 1, Issue 1, viewed 30 April 2022, https://doi.org/10.1016/j.tbench.2021.100007.

Finding CPU Cores / Rank for the DGX.

CPU Cores / Rank	**96 000 N P*Q = 2*4 (Tflops)**	**96 000 N P*Q = 1*8 (Tflops)**
1	17.46	7.38
2	21.66	10.38
3	**25.13**	11.51
4	23.81	12.18
5	24.03	**13.16**
6 (HT On)	21.71	**13.17**
7 (HT On)	21.82	12.82
8 (HT On)	21.57	**13.2**
9 (HT On)	21.39	13.14
10 (HT On)	17.70	12.29

Finding optimal NB for the DGX-1.

NB	96 000 Ns (Tflops)	110 000 Ns (Tflops)	120 000 Ns (Tflops)	140 000 Ns (Tflops)
64	**28.73**	31.29	32.24	34.31
128	28.44	**31.83**	**33.72**	37.08
192	27.68	30.93	32.31	**37.61**
256	25.00	29.29	30.76	35.89
384	24.76	23.40	29.76	35.03
512	20.98	21.90	26.19	29.88