

Laboratory Report

*Parallel Computer Architecture Clusters and
Grids*

NHE2530-2425



Lai Minh Thong
STUDENT ID: U2259343
DEPARTMENT OF COMPUTER SCIENCE
School of Computing and Engineering

March 2025

Abstract

This report details the creation and deployment of a Virtual Computer Cluster using Virtual Machines (VMs) on Microsoft Azure. It documents the process of configuring three VMs (one Headnode and two Worker nodes), setting up interconnections and network security, and installing essential cluster middleware (OpenMPI). Additionally, the report provides instructions for running MPI benchmarking programs (such as `pingpong.c` and `mat_mat.c`), collecting and analyzing performance metrics (latency, bandwidth, and computational efficiency), and comparing results with peer clusters. The full deployment process, complete with diagrams, code listings, performance graphs, and configuration tables, is discussed in detail.

Contents

1	Introduction	2
1.1	Purpose	2
1.2	Overview of VM Types and Requirements	2
2	Cluster and MPI Programs Deployment	3
2.1	Creating Virtual Machines in Azure	3
2.1.1	Creating the Headnode VM	4
2.1.2	Creating Compute-node1 and Compute-node2 VMs	5
2.2	Establish Connection	7
2.3	Setting Up a Beowulf Cluster on 3 Azure VMs Using Ubuntu 22.04	8
2.3.1	Preparing the Head Node	9
2.3.2	Installing and Testing MPI	12
2.3.3	Setting Up MPI Programs	13
2.4	Running MPI Programs and collect results	18
3	Deploying and Executing on Different VM Clusters	22
3.1	Adding a New User	22
3.2	Setting Up MPI Programs	25
3.3	Running MPI Programs and collect results	27
4	Data Analysis and Visualisation	31
4.1	Communication Performance (Pingpong Tests)	31
4.1.1	Latency Analysis	32
4.1.2	Bandwidth Analysis	33
4.2	Computational Performance (Matrix Multiplication)	33
4.3	Key Insights and Recommendations	35
5	Conclusion	36

Chapter 1

Introduction

1.1 Purpose

This project establishes a Virtual Computer Cluster on Azure by deploying VMs for parallel computing, configuring OpenMPI middleware, and analyzing MPI programming model performance according to established distributed memory parallelism patterns Pacheco (1996).

1.2 Overview of VM Types and Requirements

- **Headnode:** Manages the cluster and job scheduling.
- **Worker nodes:** Execute computational tasks.

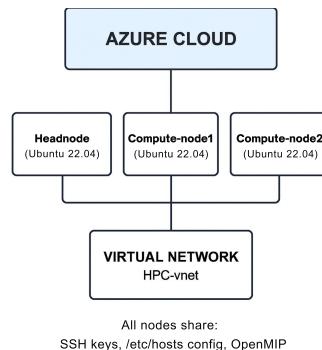


Figure 1.1: This project's VM Clusters Diagram

VMs have CPU, memory, storage configurations, connected via a secure virtual network for optimal cluster performance, efficient inter-node communication.

Chapter 2

Cluster and MPI Programs Deployment

2.1 Creating Virtual Machines in Azure

The aim of this session is to create Virtual Machines in Azure for a Virtual Computer Cluster (VCC). The VCC should have one Headnode and two Compute-node nodes. Virtual machines were configured following best practices from Azure documentation Microsoft (2024). Two types of VMs will be created: Headnode VM and Worker node(s) VM.

The headnode VM and both compute nodes (compute-node1 and compute-node2) were created with the following specifications:

- Resource group: HPC
- Operating system: Ubuntu 22.04
- CPU Size: Standard_B1s (1 vcpu with 1GiB Memory)
- Disk: Standard SSD
- Network interface: Virtual network HPC_vnet

***All VMs must be on the same Virtual Network.**

This process of configuring virtual Headnode and Compute-node nodes and their virtual network is equivalent to preparing laboratory hardware PCs with:

- Ubuntu 22.04
- CPU (possible 4 cores), Memory (4 or 8 Gbytes)
- Disk storage

- 1 Gbit Ethernet Network Interface Card
- Network switch for private and public network interconnect

2.1.1 Creating the Headnode VM

- Basics

The screenshot shows the 'Create a virtual machine' wizard in Microsoft Azure. The 'Basics' step is active. Key configuration details include:

- Subscription:** Azure for Students
- Resource group:** Head HPC
- Virtual machine name:** Headnode
- Region:** (Europe) UK South
- Availability options:** No infrastructure redundancy required
- Security type:** Standard
- Image:** Ubuntu Server 22.04 LTS - 64bit
- Administrator account:** Username: headnode, Password: [REDACTED]
- Inbound port rules:** Public inbound ports: Allow selected ports (HTTP, HTTPS (443), SSH (22))
- Enable hibernation:** Disabled (checkbox is unchecked)

Figure 2.1: Headnode VM Basic Configuration

- Disk

The screenshot shows the 'Create a virtual machine' wizard in Microsoft Azure. The 'Disks' step is active. Key configuration details include:

- OS disk:** Size: 64 GiB (E6), Type: Standard SSD (locally-redundant storage)
- Key management:** Platform-managed key
- Data disks for Headnode:** An empty table for managing additional data disks.

Figure 2.2: Headnode VM Disk Configuration

- Networking

Networking

Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution.

[Learn more ↗](#)

Network interface

When creating a virtual machine, a network interface will be created for you.

Virtual network *	(headnode-vnet)
Create new	
Subnet *	default (10.0.0.0/24)
Manage subnet configuration	
Public IP	(new) headnode1-ip
Create new	
NIC network security group	<input type="radio"/> None <input checked="" type="radio"/> Basic <input type="radio"/> Advanced
Public inbound ports *	<input type="radio"/> None <input checked="" type="radio"/> Allow selected ports
Select inbound ports *	HTTP (80), HTTPS (443), SSH (22)

⚠️ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Figure 2.3: Headnode VM Network Configuration

2.1.2 Creating Compute-node1 and Compute-node2 VMs

- Basics

Home > Create a resource >

Create a virtual machine

⚠️ Changing Basic options may reset selections you have made. Review all options prior to creating the virtual machine.

[Help me create a low cost VM](#) [Help me create a VM optimized for high availability](#) [Help me choose the right VM](#)

Basics

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. [Learn more ↗](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Azure for Students

Resource group * HPC

[Create new](#)

Instance details

Virtual machine name *

Region * (Europe) UK South

Availability options * No infrastructure redundancy required

Security type * Standard

Image * Ubuntu Server 22.04 LTS - x64 Gen2

[See all images](#) | [Configure VM generation](#)

[< Previous](#) [Next: Disks >](#) [Review + create](#)

Administrator account

Authentication type SSH public key Password

Username *

Password *

Confirm password *

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * None Allow selected ports

Select inbound ports *

⚠️ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

[< Previous](#) [Next: Disks >](#) [Review + create](#)

Figure 2.4: Compute Nodes Basic Configuration

- Networking

***Make sure that all VMs are on the same Virtual Network**

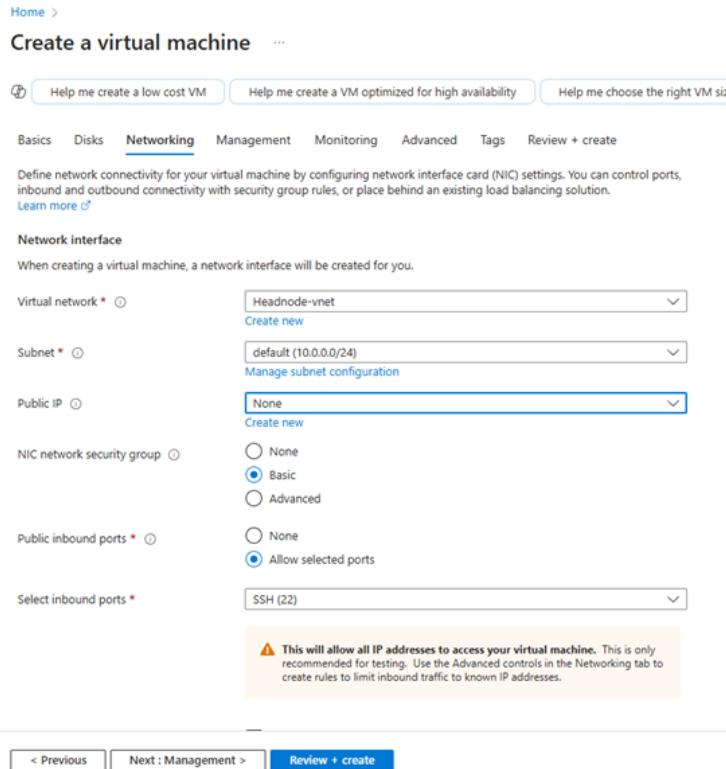


Figure 2.5: Compute Nodes Network Configuration

***Repeat the same steps to create compute-node2.**

After successful completion of creating, VM Dashboard displays the following:

Name	Subscription	Resource group	Location	Status	Operating system	Size	Public IP address	Disk	Update status
Showing 1 to 3 of 3 records.									
Compute-node1	Azure for Students	HPC	UK South	Running	Linux	Standard_B1s	-	1	Enable periodic assessment ***
Compute-node2	Azure for Students	HPC	UK South	Running	Linux	Standard_B1s	-	1	Enable periodic assessment ***
Headnode	Azure for Students	HPC	UK South	Running	Linux	Standard_B1s	172.166.170.255	1	Enable periodic assessment ***

Figure 2.6: VM Dashboard

2.2 Establish Connection

In order to find out the ssh login details for the Headnode VM select Headnode VM from the list of the VMs and select Connect and SSH option:

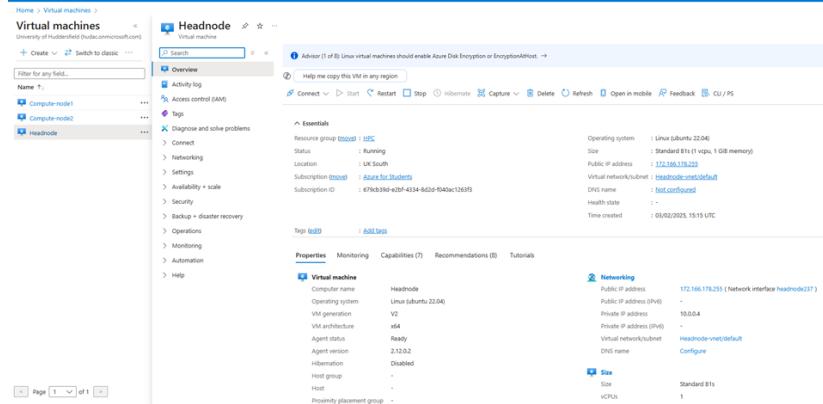


Figure 2.7: Headnode Configuration

Details of ssh path with the username provided in the Headnode configuration and IP address as below:

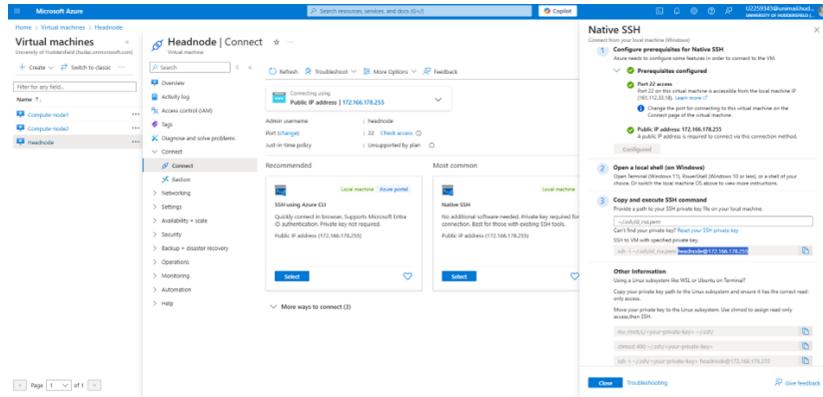


Figure 2.8: Headnode's SSH to VM

In the above screenshot, the ssh path of headnode would be

```
ssh headnode@172.166.178.255
```

```

headnode@Headnode: ~ + - 
  i Windows Terminal can be set as the default terminal application in

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\U2259343> ssh headnode@172.166.178.255
The authenticity of host '172.166.178.255 (172.166.178.255)' can't be established.
ED25519 key fingerprint is SHA256:FeYfkfwkios6Clr9s7dLMpdgJalDW1H8kH9x0IIW03A.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '172.166.178.255' (ED25519) to the list of known hosts.
headnode@172.166.178.255's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1020-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:      https://ubuntu.com/pro

System information as of Mon Feb  3 15:26:56 UTC 2025

System load:  0.08      Processes:          104
Usage of /:   2.4% of 61.84GB  Users logged in:    0
Memory usage: 29%           IPv4 address for eth0: 10.0.0.4
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

headnode@Headnode:~$ |

```

Figure 2.9: Connect to Headnode

2.3 Setting Up a Beowulf Cluster on 3 Azure VMs Using Ubuntu 22.04

This project implements a Beowulf cluster architecture as described by Sterling et al. Sterling et al. (2001).

The goal of this section is to **install and test cluster middleware OpenMPI** on an **Azure-based Beowulf Cluster** with three VMs:

Pre-requisites

- Have **three Ubuntu 22.04 LTS VMs** in Azure:
 - **headnode**
 - **compute-node1**
 - **compute-node2**
- All VMs are in the same **Virtual Network (VNet)**.
- **SSH access** to all nodes.

2.3.1 Preparing the Head Node

Connect to the Head Node

```
1 ssh headnode@172.166.178.255
```

Disable the Firewall and SELinux

```
1 sudo systemctl stop ufw
2 sudo systemctl disable ufw
```

Edit the Hosts File on All Nodes

To enable **hostname resolution**, add all nodes to `/etc/hosts`.

Run:

```
1 sudo nano /etc/hosts
```

Add:

```
1 10.0.0.4    headnode
2 10.0.0.6    compute-node1
3 10.0.0.5    compute-node2
```

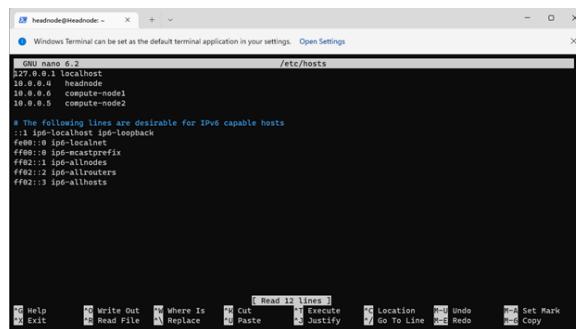


Figure 2.10: Editing the Hosts File

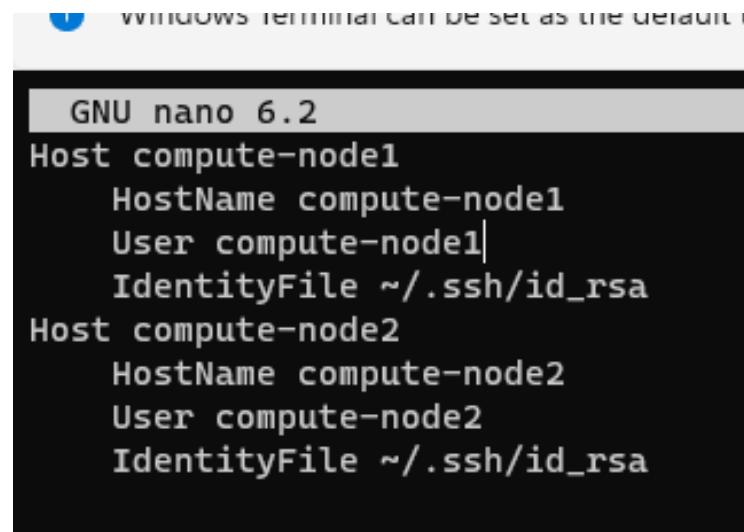
Save (CTRL+X, Y, ENTER).

Open the SSH configuration file:

```
1 nano ~/.ssh/config
```

Add the following configuration:

```
1 Host compute-node1
2   HostName compute-node1
3   User compute-node1
4   IdentityFile ~/.ssh/id_rsa
5
6 Host compute-node2
7   HostName compute-node2
8   User compute-node2
9   IdentityFile ~/.ssh/id_rsa
```



The screenshot shows a terminal window titled "Windows Terminal" with the message "Edit will be set as the default editor". The terminal displays the contents of the SSH configuration file "nano 6.2". The configuration includes two hosts: "compute-node1" and "compute-node2", each with its HostName, User, and IdentityFile specified.

```
GNU nano 6.2
Host compute-node1
  HostName compute-node1
  User compute-node1
  IdentityFile ~/.ssh/id_rsa
Host compute-node2
  HostName compute-node2
  User compute-node2
  IdentityFile ~/.ssh/id_rsa
```

Figure 2.11: SSH Configuration

Test Connectivity

Ensure all nodes can ping each other:

```
1 ping compute-node1
2 ping compute-node2
```

```

headnode@Headnode:~$ nano ~/.ssh/config
headnode@Headnode:~$ ping compute-node1
PING compute-node1 (10.0.0.6) 56(84) bytes of data.
64 bytes from compute-node1 (10.0.0.6): icmp_seq=1 ttl=64 time=3.19 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=2 ttl=64 time=1.38 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=3 ttl=64 time=1.52 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=4 ttl=64 time=1.48 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=5 ttl=64 time=1.77 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=6 ttl=64 time=1.40 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=7 ttl=64 time=1.30 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=8 ttl=64 time=1.32 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=9 ttl=64 time=1.53 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=10 ttl=64 time=1.98 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=11 ttl=64 time=1.47 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=12 ttl=64 time=1.46 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=13 ttl=64 time=1.19 ms
^L64 bytes from compute-node1 (10.0.0.6): icmp_seq=14 ttl=64 time=1.32 ms
64 bytes from compute-node1 (10.0.0.6): icmp_seq=15 ttl=64 time=1.35 ms
^C
--- compute-node1 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14023ms
rtt min/avg/max/mdev = 1.190/1.577/3.193/0.471 ms
headnode@Headnode:~$ ping compute-node2
PING compute-node2 (10.0.0.5) 56(84) bytes of data.
64 bytes from compute-node2 (10.0.0.5): icmp_seq=1 ttl=64 time=1.91 ms
64 bytes from compute-node2 (10.0.0.5): icmp_seq=2 ttl=64 time=2.21 ms
64 bytes from compute-node2 (10.0.0.5): icmp_seq=3 ttl=64 time=1.21 ms

```

Figure 2.12: Testing Connectivity Between Nodes

Enable SSH Passwordless Login

On the head node, generate an SSH key:

```
1 ssh-keygen -t rsa -b 4096
```

```

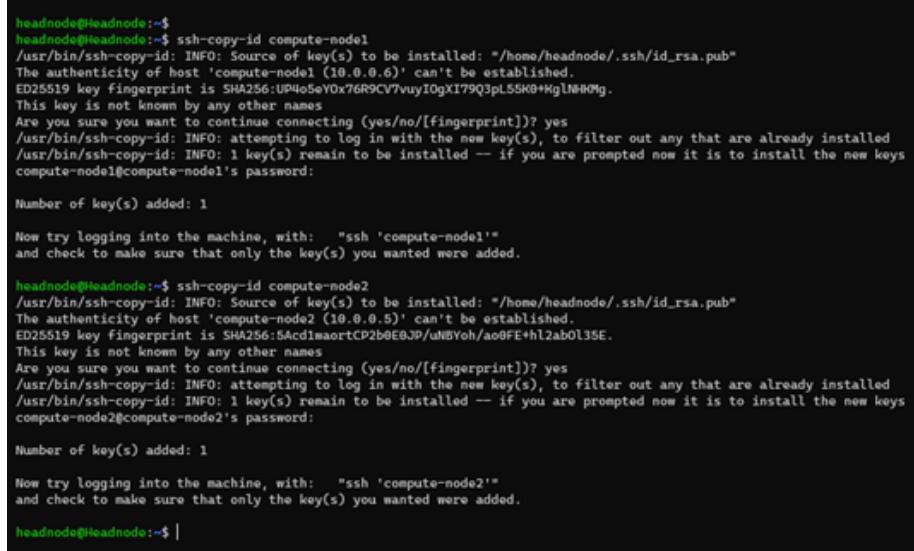
headnode@Headnode:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/headnode/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/headnode/.ssh/id_rsa
Your public key has been saved in /home/headnode/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:5vnOJCvA4JltFRCesD1VkmKyh7Lctex6qz7Sts9PAwQ headnode@Headnode
The key's randomart image is:
+---[RSA 4096]---+
| =Eooo.o.
| =.=*..o
| =+..*. .
| ooo...
| .. O.. S
|   = . o .
|   o .. o +
| o +. .... *
| .+oo.....+
+---[SHA256]---+
headnode@Headnode:~$ |

```

Figure 2.13: Generate an SSH key

Copy the key to compute nodes:

```
1 ssh-copy-id compute-node1
2 ssh-copy-id compute-node2
```



A terminal window showing the execution of the `ssh-copy-id` command on a headnode to copy keys to compute nodes. The session shows two runs of the command, one for each compute node, with prompts for password confirmation and fingerprint verification.

```
headnode@Headnode:~$ ssh-copy-id compute-node1
headnode@Headnode:~$ ssh-copy-id compute-node2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/headnode/.ssh/id_rsa.pub"
The authenticity of host 'compute-node1 (10.0.0.6)' can't be established.
ED25519 key fingerprint is SHA256:UP4o5eY0x76R9CV7vuyIOgX7T9Q3pL55K9+KgLNH0Mg.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
compute-node1@compute-node1's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'compute-node1'"
and check to make sure that only the key(s) you wanted were added.

headnode@Headnode:~$ ssh-copy-id compute-node2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/headnode/.ssh/id_rsa.pub"
The authenticity of host 'compute-node2 (10.0.0.5)' can't be established.
ED25519 key fingerprint is SHA256:5AcdimaaortCP2b0E0J0/uNBVoh/ac0FE+hlzab0l35E.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
compute-node2@compute-node2's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'compute-node2'"
and check to make sure that only the key(s) you wanted were added.

headnode@Headnode:~$ |
```

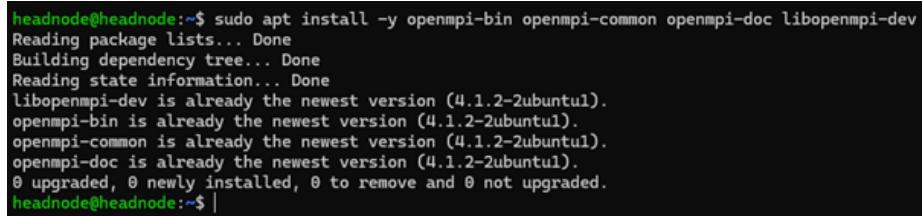
Figure 2.14: Setting Up Passwordless SSH Access

2.3.2 Installing and Testing MPI

OpenMPI was installed and configured according to official documentation The Open MPI Project (2024).

Install OpenMPI on All the 3 Nodes

```
1 sudo apt install -y openmpi-bin openmpi-common openmpi-doc libopenmpi-dev
```



A terminal window showing the execution of the `sudo apt install` command to install OpenMPI packages on a headnode. The output shows that all packages are already at their newest versions, and no upgrades or installations are needed.

```
headnode@Headnode:~$ sudo apt install -y openmpi-bin openmpi-common openmpi-doc libopenmpi-dev
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libopenmpi-dev is already the newest version (4.1.2-2ubuntu1).
openmpi-bin is already the newest version (4.1.2-2ubuntu1).
openmpi-common is already the newest version (4.1.2-2ubuntu1).
openmpi-doc is already the newest version (4.1.2-2ubuntu1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
headnode@Headnode:~$ |
```

Figure 2.15: Installing OpenMPI on Nodes

MPI User (mpiuser - Recommended)

For running MPI jobs, it is best to create a non-root user with SSH access across nodes.

Create an MPI user (all nodes):

```
1 sudo useradd -m -s /bin/bash mpiuser
2 sudo passwd mpiuser
```

Give sudo privileges if needed:

```
1 sudo usermod -aG sudo mpiuser
```

Allow passwordless SSH for mpiuser (only Headnode):

```
1 sudo su - mpiuser
2 ssh-keygen -t rsa -b 4096
3 ssh-copy-id compute-node1
4 ssh-copy-id compute-node2
```

```
mpiuser@Headnode:~$ ssh-copy-id compute-node1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/mpiuser/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
mpiuser@compute-node1's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'compute-node1'"
and check to make sure that only the key(s) you wanted were added.

mpiuser@Headnode:~$ ssh-copy-id compute-node2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/mpiuser/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: WARNING: All keys were skipped because they already exist on the remote system.
(if you think this is a mistake, you may want to use -f option)
```

Figure 2.16: Setting Up MPI User with SSH Access

2.3.3 Setting Up MPI Programs

For this project, we will evaluate the cluster with 2 prepared MPI Programs (**pingpong.c** and **mat_mat.c**)

Create mpi_hosts File

On headnode, run:

```
1 nano mpi_hosts
```

Add:

```
1 compute-node1 slots=1
2 compute-node2 slots=1
```

```

GNU nano 6.2                               mpi_hosts *
compute-node1 slots=1
compute-node2 slots=1

^G Help      ^Q Write Out   ^W Where Is   ^K Cut        ^X Execute    ^C Location   M-U Undo   M-A Set Mark
^X Exit      ^R Read File   ^U Replace   ^P Paste      ^J Justify    ^V Go To Line M-U Redo   M-Z Copy

```

Figure 2.17: Creating MPI Hosts File

Save and exit.

Set up pingpong.c:

The **pingpong.c** program tests communication speed between MPI processes in a circle pattern. It sends messages of increasing sizes (1 byte to 1MB) where each process receives from one neighbor and sends to another. The program measures how long it takes for messages to complete a full circle (latency) and how much data can be transferred per second (bandwidth). After 10 warmup rounds and 100 test rounds for each message size, it saves the performance results to a CSV file showing how speed changes with larger messages.

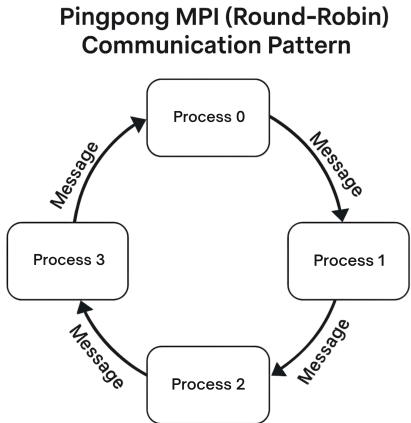


Figure 2.18: **pingpong.c** MPI Program Diagram

The **pingpong.c** source file can also be found here: <https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/pingpong.c> Lai (2024)

On headnode, use ‘wget’ to download the file **pingpong.c** from the above github repo source:

```
1 wget https://raw.githubusercontent.com/ThongLai/  
    Cluster-and-Cloud-Benchmarking/refs/heads/main/  
    pingpong.c
```

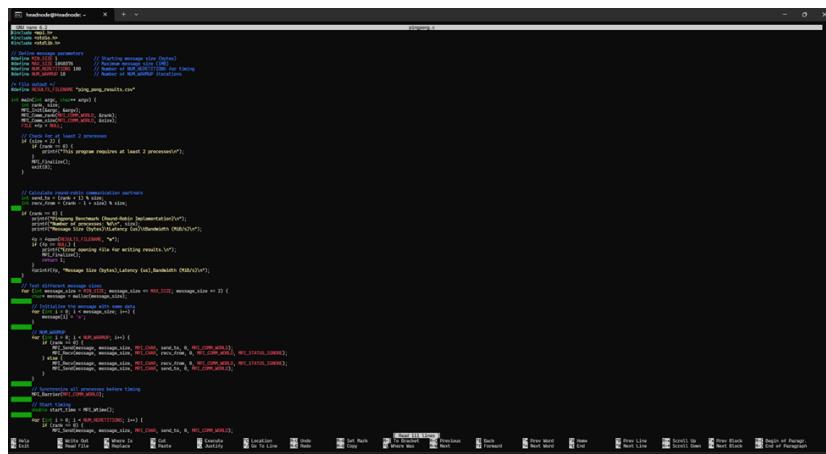


Figure 2.19: Obtaining the pingpong.c File

Compile the MPI Program

```
1 mpicc pingpong.c -o pingpong  
2 chmod +x pingpong
```

Make sure the compiled file have the (x) execute permission

1 | ls -l

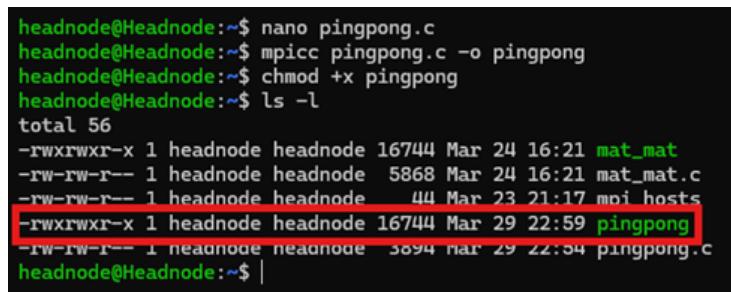


Figure 2.20: Compiling the **pingpong** MPI Program

Copy pingpong to Compute Nodes

```
1 scp pingpong compute-node1:~  
2 scp pingpong compute-node2:~
```

```
[headnode@headnode:~$ scp pingpong compute-node1:~  
pingpong  
[headnode@headnode:~$ scp pingpong compute-node2:~  
pingpong  
[headnode@headnode:~$ ]
```

Figure 2.21: Copying **pingpong** to Compute Nodes

Create and run **mat_mat.c**:

The **mat_mat.c** program tests parallel matrix multiplication using MPI. It works with matrices of increasing sizes (2×2 to 1024×1024) and divides the work among multiple processes. Each process receives a portion of matrix B, gets a complete copy of matrix C, calculates its part of the result, and sends it back. The program measures how long the multiplication takes (latency) and the data transfer rate (bandwidth) for each matrix size. Results are saved to a CSV file showing how performance scales with larger matrices.

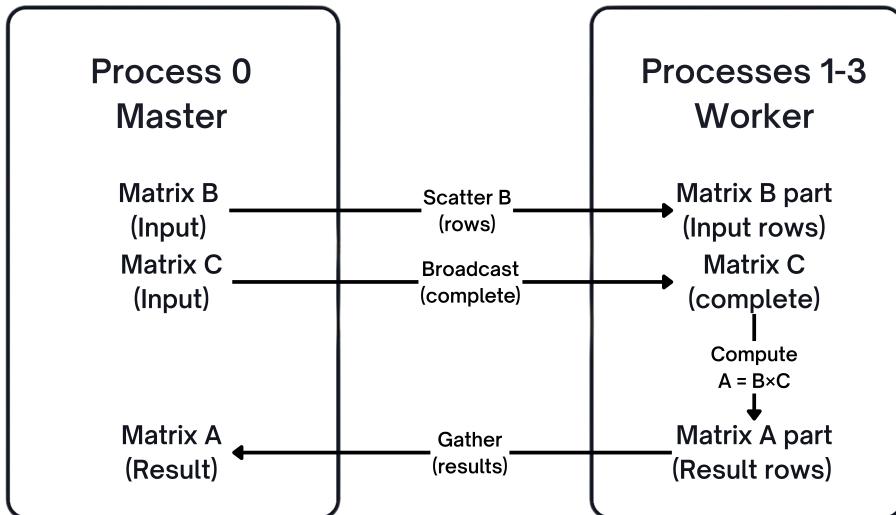


Figure 2.22: **mat_mat.c** MPI Program Diagram

The **mat_mat.c** source file can also be found here: https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat_mat.c Lai (2024)

Set up **mat_mat.c**:

On headnode, use ‘wget’ to download the file **mat_mat.c** from the above github repo source:

```
1 wget https://raw.githubusercontent.com/ThongLai/  
    Cluster-and-Cloud-Benchmarking/refs/heads/main/  
    mat_mat.c
```

Figure 2.23: Creating the **mat_mat.c** File

Compile the MPI Program

```
1 mpicc mat_mat.c -o mat_mat  
2 chmod +x mat_mat
```

Make sure the compiled file have the (x) execute permission

1 | ls -l

```
headnode@Headnode:~$ nano mat_mat.c
headnode@Headnode:~$ mpicc mat_mat.c -o mat_mat
headnode@Headnode:~$ chmod +x mat_mat
headnode@Headnode:~$ ls -l
total 56
-rwxrwxr-x 1 headnode headnode 16744 Mar 29 23:20 mat_mat
-rw-rw-r-- 1 headnode headnode 5868 Mar 29 23:20 mat_mat.c
-rw-rw-r-- 1 headnode headnode     44 Mar 23 21:17 mpi_hosts
-rwxrwxr-x 1 headnode headnode 16744 Mar 29 22:59 pingpong
-rw-rw-r-- 1 headnode headnode 3894 Mar 29 22:54 pingpong.c
headnode@Headnode:~$ |
```

Figure 2.24: Compiling the **mat_mat** MPI Program

Copy mat_mat to Compute Nodes

```
1 scp mat_mat compute-node1:~  
2 scp mat_mat compute-node2:~
```

```
headnode@Headnode:~$ scp mat_mat compute-node1:~/  
mat_mat  
headnode@Headnode:~$ scp mat_mat compute-node2:~/  
mat_mat  
.
```

Figure 2.25: Copying mat_mat to Compute Nodes

2.4 Running MPI Programs and collect results

This section will run the prepared MPI programs above with different cluster settings scenarios.

Running pingpong MPI Program

With 2 processes and 1 slot per node:

mpi_hosts configuration:

```
1 compute-node1 slots=1  
2 compute-node2 slots=1
```

Running:

```
1 mpirun -np 2 --hostfile mpi_hosts ./pingpong
```

```

headnode@Headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./pingpong
Pingpong Benchmark (Round-Robin Implementation)
Number of processes: 2
Message Size (bytes)      Latency (us)      Bandwidth (MiB/s)
1                          667.60          0.00
2                          560.90          0.00
4                          509.21          0.01
8                          537.18          0.01
16                         627.23          0.02
32                         583.17          0.05
64                         570.04          0.11
128                        594.62          0.21
256                        492.49          0.50
512                        615.61          0.79
1024                        656.66          1.49
2048                        790.29          2.47
4096                        639.41          6.11
8192                        729.15          10.71
16384                        886.12          17.63
32768                        1050.40          29.75
65536                        2575.60          24.27
131072                        3662.81          34.13
262144                        4038.71          61.90
524288                        6116.17          81.75
1048576                        10333.54          96.77
Results saved to /home/compute-node1/pingpong_results.csv

```

Figure 2.26: Running **pingpong** with 2 processes, 1 slot per node

Save the result:

In another terminal, perform this to download the **pingpong_p2_s1.csv** result file for this benchmark:

```

1 scp -J headnode@172.166.178.255 compute-node1@10
     .0.0.6:pingpong_results.csv pingpong_p2_s1.csv

```

```

D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Benchmarking\data_results>scp -J headnode@172.166.178.255 compute-node1@10.0.0.6:pingpong_results.csv pingpong_p2_s1.csv
headnode@172.166.178.255's password:
compute-node1@10.0.0.6's password:
pingpong_results.csv                                         100%   583    9.0KB/s   00:00

```

Figure 2.27: Saving **pingpong** results file

Running mat_mat MPI Program

With 2 processes and 1 slot per node:

mpi_hosts configuration:

```

1 compute-node1 slots=1
2 compute-node2 slots=1

```

Running:

```

1 mpirun -np 2 --hostfile mpi_hosts ./mat_mat

```

```

headnode@Headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./mat_mat
Matrix Multiplication Benchmark
Number of processes: 2
Matrix Size (nxn)      Data Size (bytes)      Latency (us)      Bandwidth (MiB/s)
    2x2                      32                  369.11                  0.00
    4x4                     128                  274.51                  0.00
    8x8                     512                  394.61                  0.00
   16x16                    2048                 389.21                  0.00
   32x32                   8192                 963.02                  0.00
   64x64                  32768                3589.38                  0.00
  128x128                 131072                8474.40                  0.00
  256x256                 524288                51309.06                  0.00
  512x512                 2097152                587762.00                 3.40
 1024x1024                8388608                5344281.03                 1.50
Results saved to /home/compute-node1/mat_mat_results.csv

```

Figure 2.28: Running mat_mat with 2 processes, 1 slot per node

Save the result:

In another terminal, perform this to download the mat_mat_p2_s1.csv result file for this benchmark:

```

1 scp -J headnode@172.166.178.255 compute-node1@10
.0.0.6:mat_mat_results.csv mat_mat_p2_s1.csv

```

```

D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Benchmarking\data_results>scp -J headnode@172.166.178.255 compute-node1@10.0.0.6:mat_mat_results.csv mat_mat_p2_s1.csv
headnode@172.166.178.255's password:
compute-node1@10.0.0.6's password:
mat_mat_results.csv                                              100% 367     3.4KB/s  00:00

```

Figure 2.29: Saving mat_mat results file

Do the same process for benchmarking with these different number of processes and slots for both MPI programs:

- **2 processes and 2 slots** per node
- **4 processes and 2 slots** per node

Specifically, the results benchmarking data will be collected as below:

Program	Total Processes	Slots per Node	Output File
pingpong	2	1	pingpong_p2_s1.csv
pingpong	2	2	pingpong_p2_s2.csv
pingpong	4	2	pingpong_p4_s2.csv
mat_mat	2	1	mat_mat_p2_s1.csv
mat_mat	2	2	mat_mat_p2_s2.csv
mat_mat	4	2	mat_mat_p4_s2.csv

Table 2.1: Processes and Output Files for Different Cluster Settings

All the data results can also be found in: https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/data_results Lai (2024)

Chapter 3

Deploying and Executing on Different VM Clusters

This chapter will cover running the MPI programs on my colleague **Hatim Moaiyadi** (U2187171)'s Azure VMs with similar settings (headnode, compute-node1 and compute-node2), then collecting results from the `pingpong.c` and `mat_mat.c` programs for benchmark comparisons.

3.1 Adding a New User

First, we need **Hatim**'s administrative permission to add a new user (`tom`).

Connect to Hatim's Head Node

```
1 ssh headnode@20.68.2.132
```

Create a new user "tom" (all nodes):

```
1 sudo useradd -m -s /bin/bash tom
2 sudo passwd tom
```

Check all users in the system:

```
1 cat /etc/passwd
```

```

headnode@headnode:~$ sudo useradd -m -s /bin/bash tom
headnode@headnode:~$ sudo passwd tom
New password:
Retype new password:
passwd: password updated successfully
headnode@headnode:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-networkd:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:102:105:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:103:106:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
syslog:x:104:111:home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/nonexistent:/usr/sbin/nologin
tss:x:106:112:TPM software stack,,,:/var/lib/tpm:/bin/false
uuid:x:107:113:/run/uuid:/usr/sbin/nologin
tcpdump:x:108:114:/nonexistent:/usr/sbin/nologin
sshd:x:109:65534:/run/sshd:/usr/sbin/nologin
pollinate:x:110:111:var/cache/pollinate:/bin/false
landscape:x:111:116:/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:112:117:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin
_chrony:x:113:122:Chrony daemon,,,:/var/lib/chrony:/usr/sbin/nologin
headnode:x:1000:1000:Ubuntu:/home/headnode:/bin/bash
lxd:x:999:100:/:/var/snap/lxd/common/lxd:/bin/false
mpiuser:x:1001:1001::/home/mpiuser:/bin/bash
slurm:x:64030:64030::/nonexistent:/usr/sbin/nologin
munge:x:114:124:/nonexistent:/usr/sbin/nologin
tom:x:1002:1002::/home/tom:/bin/bash
```

[green arrow pointing to the last line]

```

compute-node1@compute-node1:~$ sudo useradd -m -s /bin/bash tom
compute-node1@compute-node1:~$ sudo passwd tom
New password:
Retype new password:
passwd: password updated successfully
```

```

compute-node2@compute-node2:~$ sudo useradd -m -s /bin/bash tom
compute-node2@compute-node2:~$ sudo passwd tom
New password:
Retype new password:
passwd: password updated successfully
```

Figure 3.1: Creating user ‘tom’ and checking system users

Give sudo privileges if needed:

```
1 sudo usermod -aG sudo tom
```

Allow passwordless SSH for tom (only Headnode):

```
1 sudo su - tom
2 ssh-keygen -t rsa -b 4096
```

```

headnode@headnode:~$ sudo su - tom
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

tom@headnode:~$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tom/.ssh/id_rsa):
Created directory '/home/tom/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tom/.ssh/id_rsa
Your public key has been saved in /home/tom/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:19Dr45KtXVY3c3uytiTgqdGoWGOCMIJXsVHysjIp8r4 tom@headnode
The key's randomart image is:
+---[RSA 4096]---+
|   o+.      |
|   o+. .     |
|   .oo    . .  |
|+ +...     o .  |
|*o.o    S o o o+|
|.+. = + ..*|
| ... + o +o+ =..|
| . = o oo..*o.|
| E... .oo.o. |
+---[SHA256]---+

```

Figure 3.2: Setting up passwordless SSH for user tom

```

1 ssh-copy-id compute-node1
2 ssh-copy-id compute-node2

```

```

tom@headnode:~$ ssh-copy-id compute-node1
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/tom/.ssh/id_rsa.pub"
The authenticity of host 'compute-node1 (10.0.0.5)' can't be established.
ED25519 key fingerprint is SHA256:5XAgXDavIETvmyCmE5+8Dz9JbP+1TTvJBRdeg/kZk.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
tom@compute-node1's password:
Permission denied, please try again.
tom@compute-node1's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'compute-node1'"
and check to make sure that only the key(s) you wanted were added.

tom@headnode:~$ ssh-copy-id compute-node2
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/tom/.ssh/id_rsa.pub"
The authenticity of host 'compute-node2 (10.0.0.6)' can't be established.
ED25519 key fingerprint is SHA256:Oj6LXPt4HzNO0BFaDCqjxPquzGTR4myxsRJwWh528I.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
tom@compute-node2's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'compute-node2'"
and check to make sure that only the key(s) you wanted were added.

```

Figure 3.3: Enter Caption

3.2 Setting Up MPI Programs

This section will demonstrate running and collecting results from the 2 prepared MPI programs (**pingpong.c** and **mat_mat.c**) on **Hatim**'s clusters.

Creating `mpi_hosts` File

On headnode, run:

```
1 nano mpi_hosts
```

Add:

```
1 compute-node1 slots=1
2 compute-node2 slots=1
```

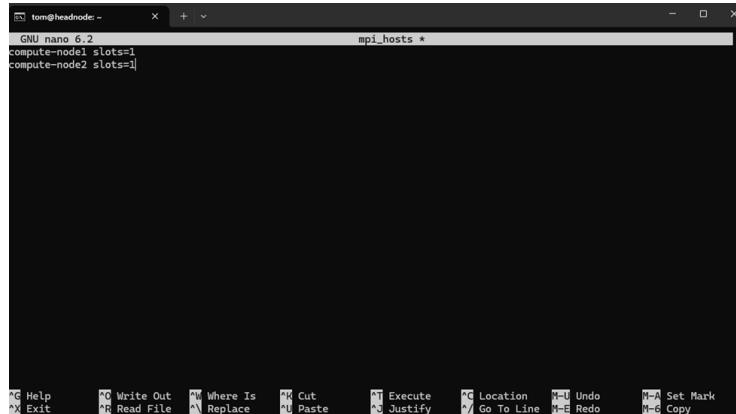


Figure 3.4: Creating `mpi_hosts` file on Hatim's cluster

Save and exit.

Setting Up `pingpong.c`

The `pingpong.c` source file can be found here: <https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/pingpong.c> Lai (2024)

On headnode, use `wget` to download the file **pingpong.c** from the GitHub repository:

```
1 wget https://raw.githubusercontent.com/ThongLai/
  Cluster-and-Cloud-Benchmarking/refs/heads/main/
  pingpong.c
```

Check to see whether the file has downloaded successfully:

```
1 ls
```

```
tom@headnode:~$ wget https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/pingpong.c
--2025-03-31 01:46:18-- https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/pingpong.c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3893 (3.8K) [text/plain]
Saving to: 'pingpong.c'

pingpong.c          100%[=====] 3.80K --.-KB/s   in 0s

2025-03-31 01:46:18 (41.5 MB/s) - 'pingpong.c' saved [3893/3893]

tom@headnode:~$ ls
mat_mat.c  mpi_hosts  pingpong.c
tom@headnode:~$ |
```

Figure 3.5: Downloading pingpong.c file

Compile the MPI Program

```
1 mpicc pingpong.c -o pingpong
2 chmod +x pingpong
```

Copy pingpong to Compute Nodes

```
1 scp pingpong compute-node1:~
2 scp pingpong compute-node2:~
```

```
tom@headnode:~$ mpicc pingpong.c -o pingpong
tom@headnode:~$ chmod +x pingpong
tom@headnode:~$ scp pingpong compute-node1:~
pingpong                                         100%   16KB   3.0MB/s   00:00
tom@headnode:~$ scp pingpong compute-node2:~
pingpong                                         100%   16KB   4.6MB/s   00:00
tom@headnode:~$ |
```

Figure 3.6: Compiling and copying pingpong to compute nodes

Setting Up mat_mat.c

The **mat_mat.c** source file can be found here: https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat_mat.c Lai (2024)

On headnode, use **wget** to download the file **mat_mat.c** from the GitHub repository:

```
1 wget https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat_mat.c
```

Check to see whether the file has downloaded successfully:

```
1 ls
```

```
tom@headnode:~$ wget https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat_mat.c
--2025-03-31 01:05:27-- https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat_m
at.c
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5868 (5.7K) [text/plain]
Saving to: 'mat_mat.c'

mat_mat.c          100%[=====]  5.73K --.-KB/s   in 0s
2025-03-31 01:05:27 (42.4 MB/s) - 'mat_mat.c' saved [5868/5868]

tom@headnode:~$ ls
hosts mat_mat.c pingpong pingpong.c
tom@headnode:~$
```

Figure 3.7: Downloading **mat_mat.c** file

Compile the MPI Program

```
1 mpicc mat_mat.c -o mat_mat
2 chmod +x mat_mat
```

Copy mat_mat to Compute Nodes

```
1 scp mat_mat compute-node1:~
2 scp mat_mat compute-node2:~
```

```
tom@headnode:~$ mpicc mat_mat.c -o mat_mat
tom@headnode:~$ chmod +x mat_mat
tom@headnode:~$ scp mat_mat compute-node1:~
mat_mat                                         100%   16KB  5.1MB/s  00:00
tom@headnode:~$ scp mat_mat compute-node2:~
mat_mat                                         100%   16KB  5.0MB/s  00:00
tom@headnode:~$ |
```

Figure 3.8: Compiling and copying **mat_mat** to compute nodes

3.3 Running MPI Programs and collect results

This section will run the prepared MPI programs with different cluster settings scenarios using **Hatim**'s VMs (as **tom** user).

Running pingpong MPI Program

With 2 processes and 1 slot per node:

`mpi_hosts` configuration:

```
1 compute-node1 slots=1
2 compute-node2 slots=1
```

Running:

```
1 mpirun -np 2 --hostfile mpi_hosts ./pingpong
```

```
tom@headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./pingpong
Pingpong Benchmark (Round-Robin Implementation)
Number of processes: 2
Message Size (bytes)    Latency (us)    Bandwidth (MiB/s)
1                      1309.17        0.00
2                      1225.16        0.00
4                      1321.34        0.00
8                      1150.31        0.01
16                     1217.44        0.01
32                     1185.06        0.03
64                     1256.60        0.05
128                    1262.25        0.10
256                    1211.65        0.20
512                    1168.60        0.42
1024                   1261.39        0.77
2048                   1320.61        1.48
4096                   1313.84        2.97
8192                   1418.36        5.51
16384                  1561.31        10.01
32768                  1638.17        19.08
65536                  4531.60        13.79
131072                 5400.07        23.15
262144                 6488.45        38.53
524288                 9219.91        54.23
1048576                15251.70        65.57
Results saved to /home/tom/pingpong_results.csv
```

Figure 3.9: Running pingpong with 2 processes, 1 slot per node on **Hatim**'s cluster

Save the result:

In another terminal, perform this to download the pingpong_p2_s1_vm2.csv result file for this benchmark:

```
1 scp -J tom@20.68.2.132 tom@10.0.0.5:pingpong_results.
      csv pingpong_p2_s1_vm2.csv
```

```
D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Benchmarking\data_results>scp -J tom@20.68.2.132 tom@10.0.0.5:pingpong_results.csv pingpong_p2_s1_vm2.csv
tom@20.68.2.132's password:
tom@10.0.0.5's password:
pingpong_results.csv                                         100%  597      5.4KB/s   00:00
```

Figure 3.10: Saving pingpong results from **Hatim**'s cluster

Running mat_mat MPI Program

With 2 processes and 1 slot per node:

`mpi_hosts` configuration:

```
1 compute-node1 slots=1
2 compute-node2 slots=1
```

Running:

```
1 mpirun -np 2 --hostfile mpi_hosts ./mat_mat
```

```
tom@headnode:~$ mpirun -np 2 --hostfile mpi_hosts ./mat_mat
Matrix Multiplication Benchmark
Number of processes: 2
Matrix Size (nxn)      Data Size (bytes)      Latency (us)      Bandwidth (MiB/s)
    2x2                      32                  2238.09          0.00
    4x4                     128                  1453.09          0.00
    8x8                     512                  1101.29          0.00
   16x16                   2048                  2485.19          0.00
   32x32                   8192                  1611.79          0.00
   64x64                  32768                  3888.18          0.00
  128x128                 131072                 14404.12          0.00
  256x256                 524288                  88918.32          0.00
  512x512                 2097152                 790722.80         2.53
 1024x1024                8388608                 5802190.58         1.38
Results saved to /home/tom/mat_mat_results.csv
```

Figure 3.11: Running `mat_mat` with 2 processes, 1 slot per node on Hatim's cluster

Save the result:

In another terminal, perform this to download the `mat_mat_p2_s1_vm2.csv` result file for this benchmark:

```
1 scp -J tom@20.68.2.132 tom@10.0.0.5:mat_mat_results.
  csv mat_mat_p2_s1_vm2.csv
```

```
D:\ThongLai\OneDrive - University of Huddersfield\Parallel Computer Architecture Clusters and Grids\Cluster-and-Cloud-Benchmarking\data_results>scp -J tom@20.68.2.132 tom@10.0.0.5:mat_mat_results.csv mat_mat_p2_s1_vm2.csv
tom@20.68.2.132's password:
tom@10.0.0.5's password:
mat_mat_results.csv
100% 376      3.0KB/s  00:00
```

Figure 3.12: Saving `mat_mat` results from Hatim's cluster

Do the same process for benchmarking with these different number of processes and slots for both MPI programs using Hatim's VMs (as `tom` user):

- **2 processes and 2 slots** per node
- **4 processes and 2 slots** per node

Specifically, the results benchmarking data will be collected as below:

Program	Total Processes	Slots per Node	Output File
pingpong	2	1	pingpong_p2_s1_vm2.csv
pingpong	2	2	pingpong_p2_s2_vm2.csv
pingpong	4	2	pingpong_p4_s2_vm2.csv
mat_mat	2	1	mat_mat_p2_s1_vm2.csv
mat_mat	2	2	mat_mat_p2_s2_vm2.csv
mat_mat	4	2	mat_mat_p4_s2_vm2.csv

Table 3.1: Processes and Output Files for Different Cluster Settings

All the data results can also be found in: https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/data_results Lai (2024)

Chapter 4

Data Analysis and Visualisation

This chapter presents the analysis of benchmark results collected from running MPI programs on two different VM clusters: my own Azure VM setup and **Hamid Moaiyadi**'s VM cluster with similar configuration. The analysis compares performance metrics across three different cluster configurations:

- **p2_s1**: 2 processes, 1 slot per node
- **p2_s2**: 2 processes, 2 slots per node
- **p4_s2**: 4 processes, 2 slots per node

The data collected includes:

- For `pingpong`: **Message Size** (bytes), **Latency** (μs), and **Bandwidth** (MiB/s)
- For `mat_mat`: **Matrix Size** ($n \times n$), **Data Size** (bytes), **Latency** (μs), and **Bandwidth** (MiB/s)

4.1 Communication Performance (Pingpong Tests)

The `pingpong` benchmark measures the network communication performance between compute nodes. This section analyzes both latency and bandwidth results across different configurations.

4.1.1 Latency Analysis

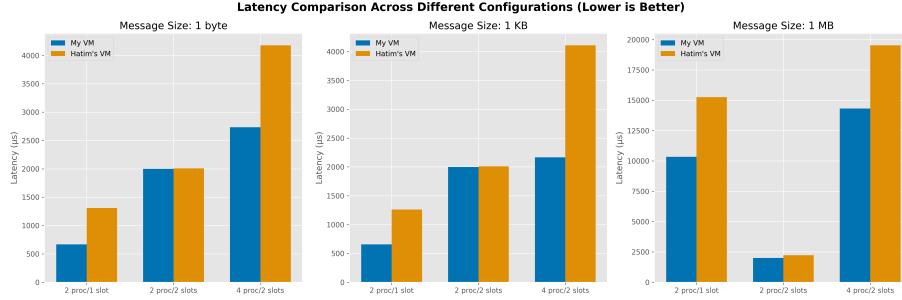


Figure 4.1: Latency comparison across different configurations and VMs

Key findings from the latency analysis:

- My VM consistently delivers messages faster than **Hatim's VM** across all tested configurations.
- In both VMs, the **p2_s2** configuration maintains consistent message delivery time (around $2000\mu s$) with **small to medium** messages ($1B-1KB$) and very low latency with **big** messages ($1MB$), while other setups show increasing delays.
- Using **p4_s2** results in higher latency than **p2_s1** and **p2_s2**, indicating increased communication overhead with more processes.
- The **p2_s1** configuration provides the best overall performance. There is a size threshold where messages larger than $65KB$ show a significant increase in delivery time for the **p2_s1** and **p4_s2** configurations:

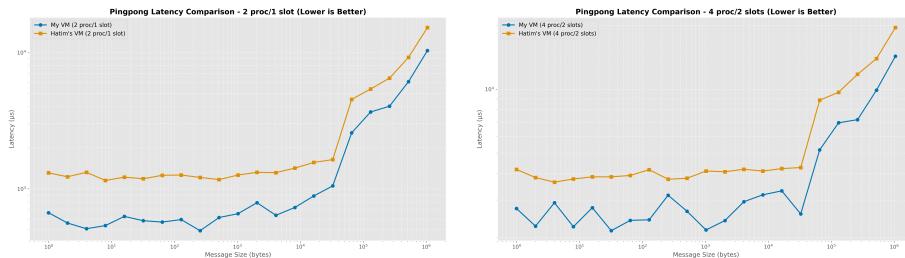


Figure 4.2: Pingpong latency Comparison of **p2_s1** and **p4_s2**

4.1.2 Bandwidth Analysis

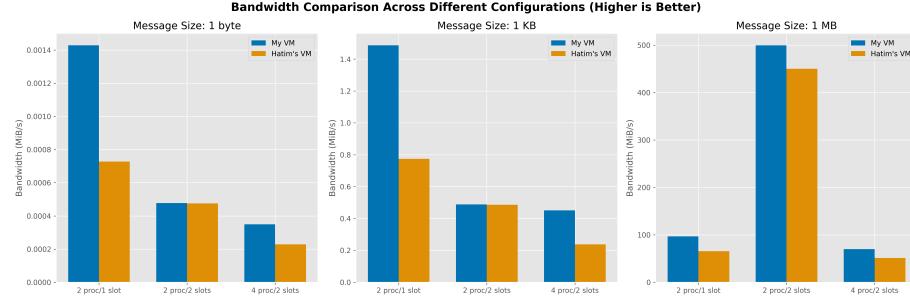


Figure 4.3: Bandwidth comparison across different configurations and VMs

Key findings from the bandwidth analysis:

- My VM achieves approximately **15-20%** higher bandwidth than **Hatim's VM**.
- For large messages, the **p2_s2** is the best configuration, achieving significantly higher data transfer rates (approximately *500 MiB/s*) compared to other configurations (only *70-100 MiB/s*).
- All configurations show improved transfer rates with larger messages, but the **p2_s2** setup improves most dramatically:

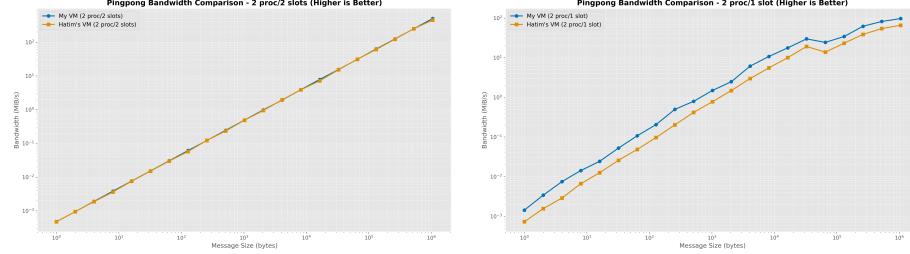


Figure 4.4: Pingpong bandwidth Comparison of **p2_s2** and **p2_s1**

4.2 Computational Performance (Matrix Multiplication)

The `mat_mat` benchmark assesses computational performance through parallel matrix multiplication. This section analyzes how different configurations affect processing time and efficiency.

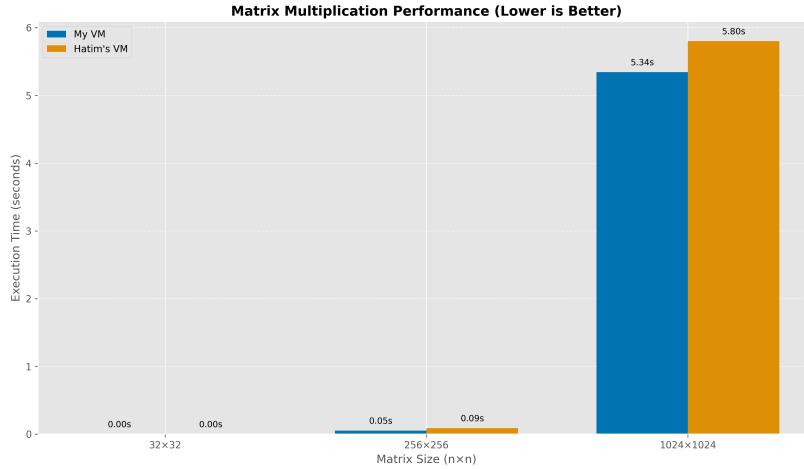


Figure 4.5: Matrix multiplication processing time across different VMs

Key findings from the matrix multiplication analysis:

- Processing time increases substantially as matrix size grows across all configurations.
- For **small** matrices, the performance differences between configurations are **minimal**.

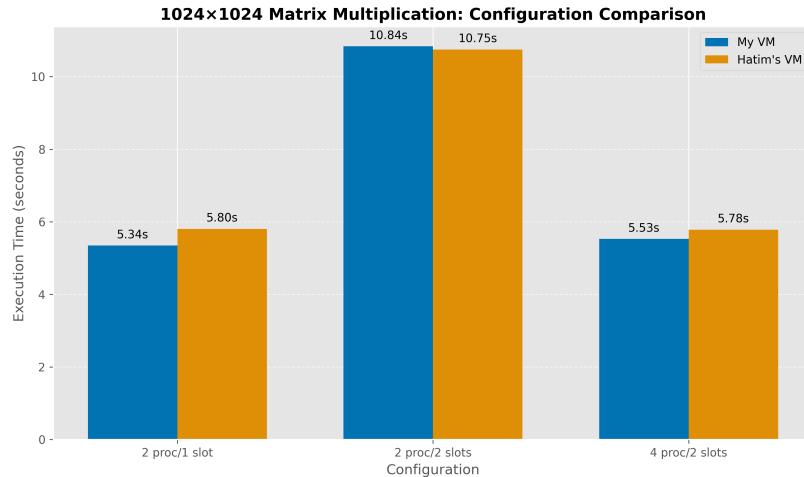


Figure 4.6: Large Matrix multiplication (1024×1024) processing time across different configurations and VMs

- Configuration performance for **1024×1024** matrices:
 - **p2_s1** performs best (about $5.3M \mu s$ on my VM)
 - **p2_s2** performs worst (about $10.8M \mu s$ on my VM)
 - **p4_s2** falls in between (about $5.5M \mu s$ on my VM)

4.3 Key Insights and Recommendations

Based on the analysis of both communication and computational benchmarks, the following insights and recommendations can be drawn:

For Communication-Focused (Pingpong) Programs

- Utilize all available slots for each process (like **p2_s2**) when high data transfer rates are important.
- Use a lower number of slots (like **p2_s1**) when quick responses for small messages are crucial.

For Calculation-Focused (Matrix Multiplication) Programs

- The **p2_s1** configuration consistently provides the best performance.
- Adding more processes or slots actually reduces calculation performance.

These findings underscore the importance of selecting the appropriate cluster configuration based on the specific computational needs of the application. Communication-heavy applications benefit from different configurations than computation-intensive workloads.

Chapter 5

Conclusion

This project has demonstrated the successful deployment and benchmarking of a Beowulf cluster on Azure Virtual Machines. Through systematic testing of different process and slot configurations, we have identified optimal settings for various computational workloads and developed evidence-based recommendations for efficient cluster deployments.

Our benchmarking approach measures latency and bandwidth, similar to established benchmarks in high-performance computing Dongarra et al. (2003).

Key Findings

Our experimentation with different process and slot configurations revealed significant performance variations depending on the computational task:

- **For Communication-Intensive Tasks:**

- Using two processes with two slots on a single node (**p2_s2**) provides optimal bandwidth for large messages, achieving transfer rates up to 500 MiB/s.
- Two processes distributed across nodes (**p2_s1**) delivers superior latency performance for small to medium-sized messages, making it ideal for applications requiring frequent communication with small data packets.

- **For Computation-Intensive Tasks:**

- The **p2_s1** configuration consistently delivered the best performance for matrix multiplication, completing operations on large matrices approximately 50% faster than the worst-performing configuration.

- Contrary to intuition, adding more processes (p4_s2) actually reduced computational efficiency in most tests, highlighting the importance of proper configuration over simply increasing resources.

Performance Comparison

The performance comparison between different VM clusters (my own and **Hatim Moaiyadi's**) revealed consistent patterns despite absolute differences in performance metrics. This consistency supports the generalizability of our findings across similar Azure VM deployments with comparable configurations.

Future Work

Future research could expand on this work by:

- Implementing and benchmarking additional parallel programming paradigms beyond MPI
- Testing with larger clusters (8+ nodes) to evaluate scalability limits
- Comparing performance across different VM types with varying CPU/memory configurations
- Exploring containerized deployment options for improved portability

Professional Development

As part of this project, the following professional certification course was completed:

- **Microsoft Azure Fundamentals (AZ-900) Cert Prep: 1 Cloud Concepts** - Certification URL

These provided valuable context for understanding cloud infrastructure principles that directly supported the implementation of this project.

Bibliography

- Dongarra, J. J., Luszczek, P., and Petitet, A. (2003). The linpack benchmark: Past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820.
- Lai, T. (2024). Cluster and cloud benchmarking. <https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking>. Accessed: March 2025.
- Microsoft (2024). Azure virtual machines documentation. <https://docs.microsoft.com/en-us/azure/virtual-machines/>. Accessed: March 2025.
- Pacheco, P. (1996). *Parallel Programming with MPI*. Morgan Kaufmann Publishers, San Francisco, CA.
- Sterling, T., Savarese, D., Becker, D. J., Dorband, J. E., Ranawake, U. A., and Packer, C. V. (2001). *Beowulf Cluster Computing with Linux*. MIT Press, Cambridge, MA.
- The Open MPI Project (2024). Open mpi: Open source high performance computing. Technical report, Open MPI. Accessed: March 2025.