**Laboratory Report**

Contents

[Cluster and MPI Programs Deployment: 2](#_Toc194297089)

[1) Creating Virtual Machines in Azure 2](#_Toc194297090)

[1.1) Creating the Headnode VM 2](#_Toc194297091)

[1.2) Creating compute-node1 & 2 VM 5](#_Toc194297092)

[2) Establish Connection 8](#_Toc194297093)

[3) Open Ports in Azure Network Security Group (NSG) 10](#_Toc194297094)

[4) Setting Up a Beowulf Cluster on 3 Azure VMs Using Ubuntu 22.04 11](#_Toc194297095)

[4.1) Prepare the Head Node 12](#_Toc194297096)

[4.2) Install and Test MPI 16](#_Toc194297097)

[4.3) Create MPI Programs 17](#_Toc194297098)

[5) Run MPI Programs and collect results 25](#_Toc194297099)

[Run on other clusters in different VMs: 28](#_Toc194297100)

[1) Add new user 28](#_Toc194297101)

[2) Set up MPI Programs 31](#_Toc194297102)

[3) Run MPI Programs and collect results 34](#_Toc194297103)

[Data Analysis and Visualisation: 38](#_Toc194297104)

[1. Communication Performance (Pingpong Tests) 38](#_Toc194297105)

[1.1) Latency Analysis 38](#_Toc194297106)

[1.2) Bandwidth Analysis 40](#_Toc194297107)

[2. Computational Performance (Matrix Multiplication) 41](#_Toc194297108)

[3. Key Insights and Recommendations 42](#_Toc194297109)

[Conclusion: 43](#_Toc194297110)

# Introduction

## 1.1 Purpose

The purpose of this project is to document the complete process of creating

a Virtual Computer Cluster on Azure. This includes the deployment of VMs

used to run parallel computing tasks, the configuration of cluster middleware

such as Slurm and OpenMPI, and a comprehensive analysis of MPI program

performance.

## 1.2 Overview of VM Types and Requirements

• Headnode: Manages the cluster and job scheduling.

• Worker nodes: Execute computational tasks.

Each VM is configured with appropriate CPU, memory, and storage to ensure

that the cluster performs optimally. The VMs are connected via a secure virtual

network to allow for efficient inter-node communication.

# Cluster and MPI Programs Deployment:

1. Creating Virtual Machines in Azure

The aim of this session is to create Virtual Machines in Azure for a Virtual Computer Cluster (VCC).

Two types of VMs will be created: **Headnode** VM and **Worker** node(s) VM.

The VCC should have one Headnode and two Compute-node nodes.

The Headnode VM will require the following resources:

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

The Compute-node nodes (Compute-node-one and Compute-node-two) VMs will require the following resources:

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.

* 1. Creating the Headnode VM

**Basics** set up:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Disk** set up:

A screenshot of a computer

Description automatically generated

**Networking** set up:

A screenshot of a computer

Description automatically generated

* 1. Creating compute-node1 & 2 VM

**Basics** set up:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Networking** set up:

Make sure that all VMs are on the same Virtual Network

A screenshot of a computer

Description automatically generated

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

**Basics** set up:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Networking** set up:

A screenshot of a computer

Description automatically generated

After successful completion of Tasks 1 and 2, VM Dashboard displays the following:

A screenshot of a computer

Description automatically generated

1. 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:

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

In the above screenshot, the ssh path of headnode would be *ssh* [*headnode@172.166.178.255*](mailto:headnode@172.166.178.255)*:*

A screenshot of a computer

Description automatically generated

1. Open Ports in Azure Network Security Group (NSG)

By default, Azure **blocks all inbound traffic** except for SSH (22). So, it need to manually open **6817 (Slurm Control)** and **6818 (Slurm Daemon)**.

**Modify NSG Rules in Azure Portal:**

1. **Go to the Azure Portal** → Navigate to **"Virtual Machines"**.
2. Click on the **head node VM** → Scroll down to **Networking**.
3. Under **"Inbound port rules"**, click **"Add inbound rule"**.
4. Configure the rule for Slurm ports **6817** and **6818**:
   * **Port:** 6817
   * **Protocol:** TCP
   * **Source:** Any
   * **Destination:** Any
   * **Action:** Allow
   * **Priority:** 101
   * **Name:** **Allow-Slurm-Control**
   * **Port:** 6818
   * **Protocol:** TCP
   * **Source:** Any
   * **Destination:** Any
   * **Action:** Allow
   * **Priority:** 102
   * **Name:** **Allow-Slurm-Daemon**
5. **Repeat steps 2-4 for each compute node**.

Headnode:  
A screenshot of a computer

Description automatically generated

Compute-node1:

A screenshot of a computer

Description automatically generated

Compute-node2:

A screenshot of a computer

Description automatically generated

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

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

1. **Head Node (Master)** → Runs the Slurm Controller (slurmctld).
2. **Compute Node 1**
3. **Compute Node 2**

**🔹 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.

### 4.1) Prepare the Head Node

**Connect to the Head Node**

*ssh headnode@172.166.178.255*

**Disable the Firewall and SELinux**

*sudo systemctl stop ufw*

*sudo systemctl disable ufw*

**Edit the Hosts File on All Nodes**

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

Run:

*sudo nano /etc/hosts*

Add:

*10.0.0.4 headnode*

*10.0.0.6 compute-node1*

*10.0.0.5 compute-node2*

*A screenshot of a computer

AI-generated content may be incorrect.*

Save (CTRL+X, Y, ENTER).

**Open the SSH configuration file**:

*nano ~/.ssh/config*

**Add the following configuration**:

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

A screenshot of a computer

AI-generated content may be incorrect.

**Test Connectivity**

Ensure all nodes can **ping each other**:

*ping compute-node1*

*ping compute-node2*

*A computer screen shot of a program

AI-generated content may be incorrect.*

**Enable SSH Passwordless Login**

On the head node, generate an SSH key:

*ssh-keygen -t rsa -b 4096*

*A screen shot of a computer

AI-generated content may be incorrect.*

Copy the key to compute nodes:

*ssh-copy-id compute-node1*

*ssh-copy-id compute-node2*

*A screenshot of a computer program

AI-generated content may be incorrect.*

### 4.2) Install and Test MPI

**Install OpenMPI on All the 3 Nodes**

*sudo apt install -y openmpi-bin openmpi-common openmpi-doc libopenmpi-dev*

*A screen shot of a computer

AI-generated content may be incorrect.*

**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):**

*sudo useradd -m -s /bin/bash mpiuser*

*sudo passwd mpiuser*

**Give sudo privileges if needed:**

*sudo usermod -aG sudo mpiuser*

**Allow passwordless SSH for mpiuser (only Headnode):**

*sudo su - mpiuser*

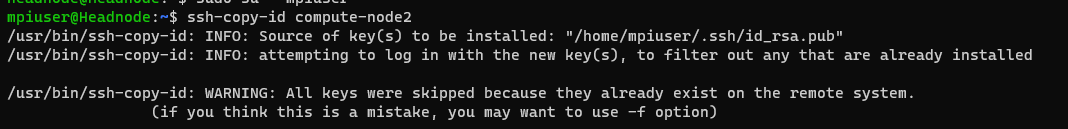
*ssh-keygen -t rsa -b 4096*

*ssh-copy-id compute-node1*

*A screen shot of a computer

AI-generated content may be incorrect.*

*ssh-copy-id compute-node2*

**

### 4.3) Create 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:

*nano mpi\_hosts*

Add:

*compute-node1 slots=1*

*compute-node2 slots=1*

*A screenshot of a computer

AI-generated content may be incorrect.*

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.

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

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

*wget* [*https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/pingpong.c*](https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/pingpong.c)

A black screen with white text

AI-generated content may be incorrect.

**Compile the MPI Program**

*mpicc pingpong.c -o pingpong*

*chmod +x pingpong*

**Make sure the compiled file have the (`x*`*) excecute permission**

*ls -l*

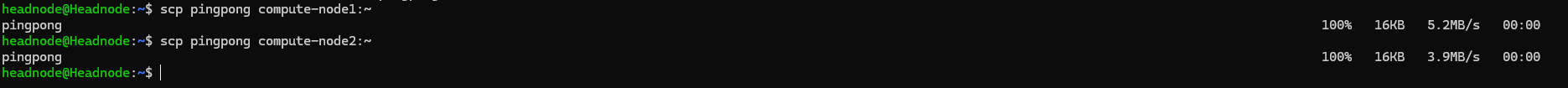
*A computer screen with white text and numbers

AI-generated content may be incorrect.*

**Copy pingpong to Compute Nodes**

*scp pingpong compute-node1:~*

*scp pingpong compute-node2:~*

**

+ 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.

*The* ***mat\_mat.c*** *source file can also be found here:* [*https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat\_mat.c*](https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat_mat.c)

+ Set up **mat\_mat.c**:

On **headnode**, use *`wget`* to download the file **mat\_mat.c** from the above github repo source:

*wget* [*https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat\_mat.c*](https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat_mat.c)

A screenshot of a computer

AI-generated content may be incorrect.

**Compile the MPI Program**

*mpicc mat\_mat.c -o mat\_mat*

*chmod +x mat\_mat*

**Make sure the compiled file have the (`x*`*) excecute permission**

*ls -l*

*A screen shot of a computer

AI-generated content may be incorrect.*

**Copy mat\_mat to Compute Nodes**

*scp mat\_mat compute-node1:~*

*scp mat\_mat compute-node2:~*

**

## 5) Run MPI Programs and collect results

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

**Run `pingpong` MPI Program**

- With **2 processes** and **1 slots per node**:

`mpi\_hosts` configuration:



Running:

*mpirun -np 2 --hostfile mpi\_hosts ./pingpong*

**Expected Output:**

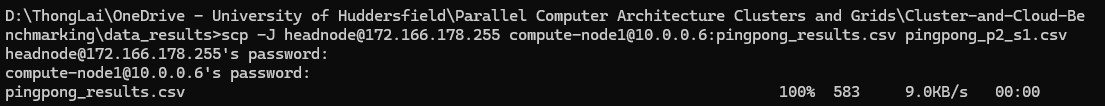
**A screenshot of a computer

AI-generated content may be incorrect.**

**Save the result:**

In another terminal, perform this to download the ***`pingpong\_p2\_s1.csv`*** result file for this benchmark:

*scp -J headnode@172.166.178.255 compute-node1@10.0.0.6:pingpong\_results.csv* ***pingpong\_p2\_s1.csv***



**Run `mat\_mat` MPI Program**

- With **2 processes** and **1 slots per node**:

`mpi\_hosts` configuration:



Running:

*mpirun -np 2 --hostfile mpi\_hosts ./mat\_mat*

**Expected Output:**

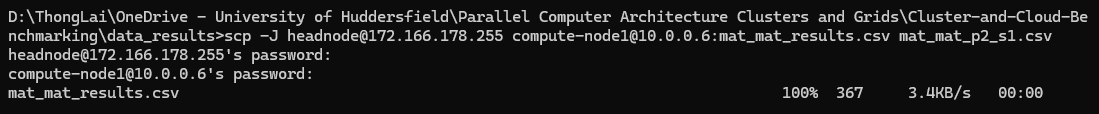
**A screen shot of a computer

AI-generated content may be incorrect.**

**Save the result:**

In another terminal, perform this to download the ***`mat\_mat\_p2\_s1.csv`*** result file for this benchmark:

*scp -J headnode@172.166.178.255 compute-node1@10.0.0.6:mat\_mat\_results.csv* ***mat\_mat\_p2\_s1.csv***



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** and **4 processes**, **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 |

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

# Deploying and Executing on Different VM Clusters:

This section will try to run on my colleague **Hatim Moaiyadi** **(U2187171)**’s Azure VMs (with similar settings including *headnode*, *compute-node1* and *compute-node2*), then run and get results from those 2 above MPI Programs `**pingpong.c**` and `**mat\_mat.c**` for later benchmarks comparisons.

## 1) Adding new user

First, will need **Hatim**’s administrative permission to add new user (**`tom`**)

**Connect to the Hatim’s Head Node**

*ssh headnode@****20.68.2.132***

**Create an new user “tom” (all nodes):**

*sudo useradd -m -s /bin/bash* ***tom***

*sudo passwd* ***tom***

**Check all users in the system:**

*cat /etc/passwd*

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

**Give sudo privileges if needed:**

*sudo usermod -aG sudo* ***tom***

**Allow passwordless SSH for mpiuser (only Headnode):**

*sudo su -* ***tom***

*ssh-keygen -t rsa -b 4096*

A screenshot of a computer program

AI-generated content may be incorrect.

*ssh-copy-id compute-node1*

*ssh-copy-id compute-node2*

A screenshot of a computer program

AI-generated content may be incorrect.

## 2) Set up MPI Programs

This section will try to run and collect results from the 2 prepared above MPI Programs (**pingpong.c** and **mat\_mat.c**) on **Hatim**’s clusters:

+ Create *`mpi\_hosts`* File

On **headnode**, run:

*nano mpi\_hosts*

Add:

*compute-node1 slots=1*

*compute-node2 slots=1*

*A screenshot of a computer

AI-generated content may be incorrect.*

Save and exit.

+ Set up **pingpong.c**:

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

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

*wget* [*https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/pingpong.c*](https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/pingpong.c)

**Check to see whether the file have downloaded successfully**

*ls*

A screenshot of a computer

AI-generated content may be incorrect.

**Compile the MPI Program**

*mpicc pingpong.c -o pingpong*

*chmod +x pingpong*

**Copy pingpong to Compute Nodes**

*scp pingpong compute-node1:~*

*scp pingpong compute-node2:~*

*A black screen with a black background

AI-generated content may be incorrect.*

+ Set 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*](https://github.com/ThongLai/Cluster-and-Cloud-Benchmarking/blob/main/mat_mat.c)

On **headnode**, use *`wget`* to download the file **mat\_mat.c** from the above github repo source:

*wget* [*https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat\_mat.c*](https://raw.githubusercontent.com/ThongLai/Cluster-and-Cloud-Benchmarking/refs/heads/main/mat_mat.c)

**Check to see whether the file have downloaded successfully**

*ls*

A screenshot of a computer

AI-generated content may be incorrect.

**Compile the MPI Program**

*mpicc mat\_mat.c -o mat\_mat*

*chmod +x mat\_mat*

**Copy mat\_mat to Compute Nodes**

*scp mat\_mat compute-node1:~*

*scp mat\_mat compute-node2:~*

*A black screen with a black background

AI-generated content may be incorrect.*

## 3) Run MPI Programs and collect results

- This section will run the prepared MPI programs above with different cluster settings scenarios using **Hatim’s** VMs (as ***`tom`*** user):

**Run `pingpong` MPI Program**

- With **2 processes** and **1 slots per node**:

`mpi\_hosts` configuration:



Running:

*mpirun -np 2 --hostfile mpi\_hosts ./pingpong*

**Expected Output:**

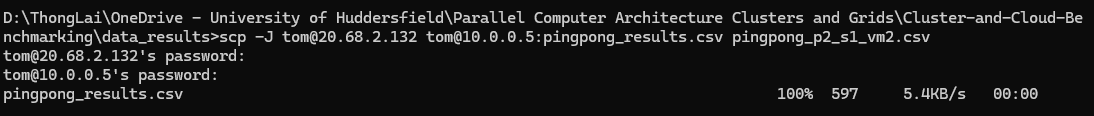
**A screenshot of a computer

AI-generated content may be incorrect.**

**Save the result:**

In another terminal, perform this to download the ***`pingpong\_p2\_s1\_*****vm2*.csv`*** result file for this benchmark:

*scp -J tom@20.68.2.132 tom@10.0.0.5:pingpong\_results.csv* ***pingpong\_p2\_s1\_vm2.csv***



**Run `mat\_mat` MPI Program**

- With **2 processes** and **1 slots per node**:

`mpi\_hosts` configuration:



Running:

*mpirun -np 2 --hostfile mpi\_hosts ./mat\_mat*

**Expected Output:**

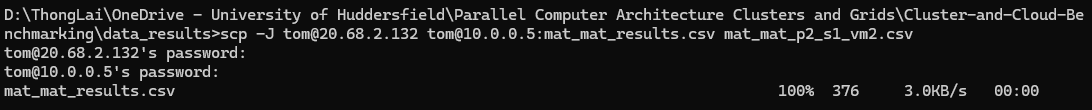
**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Save the result:**

In another terminal, perform this to download the ***`mat\_mat\_p2\_s1\_vm2.csv`*** result file for this benchmark:

*scp -J tom@20.68.2.132 tom@10.0.0.5:mat\_mat\_results.csv* ***mat\_mat\_p2\_s1\_vm2.csv***



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 |

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

# Data Analysis and Visualisation:

I gathered data for three configurations from both VMs:

* **p2\_s1 (2 processes, 1 slot)**
* **p2\_s2 (2 processes, 2 slots)**
* **p4\_s2 (4 processes, 2 slots)**

For each **pingpong** CSV files record Message Size (bytes), Latency (µs), and Bandwidth (MiB/s).

For each **mat\_mat** CSV reports Matrix Size (nxn), Data Size (bytes), Latency (µs), and Bandwidth (MiB/s).

## 1. Communication Performance (Pingpong Tests)

* 1. Latency Analysis

A graph with blue and orange bars

AI-generated content may be incorrect.

**-** My VM consistently delivers messages faster than Hatim's VM across all tested configurations.

**-** In both VM, the p2\_s2 configuration maintains consistent message delivery time (around 2000μs) 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 for messages larger than 65KB show a significant increase in delivery time for the p2\_s1 and p4\_s2 configurations:

A graph of a line

AI-generated content may be incorrect.

A graph of a line graph

AI-generated content may be incorrect.

* 1. Bandwidth Analysis

A graph with blue and orange bars

AI-generated content may be incorrect.

My VM achieves approximately 15-20% higher bandwidth than Hatim's VM.

For large messages, the p2\_s2 is the best configuration achieves 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:

A graph of a graph

AI-generated content may be incorrect.

A graph with a line

AI-generated content may be incorrect.

## 2. Computational Performance (Matrix Multiplication)

A graph with a blue bar

AI-generated content may be incorrect.

Processing time increases substantially as matrix size grows across all configurations.

For small matrices, the performance differences between configurations are minimal.

A graph of multiple colored bars

AI-generated content may be incorrect.

Configuration Performance (for 1024×1024 matrices):

+ p2\_s1 performs best (about 5.3M μs on my VM)

+ p2\_s2 performs worst (about 10.8M μs on my VM)

+ p4\_s2 falls in between (about 5.5M μs on my VM)

## 3. Key Insights and Recommendations

For Communication-Focused (Pingpong) Programs:

- Ultize all number of slots for each process (like p2\_s2) when high data transfer rates are important.

- Use 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.

# Conclusion:

Optimal Settings:

• Use lower number of slots (like p2\_s1) configuration when low-latency communication is critical, especially for matrix multiplication tasks or quick responses for small messages.

• When achieving higher bandwidth is needed (e.g., for large message transmission in the pingpong test) try to ultize all number of slots for each process would help with the data transfer rates.

Performance Recommendations: Avoid configurations involving many processes but lower slots (p4\_s2) as they introduce significant overhead.

Finished Online Learning Courses:  
Microsoft Azure Fundamentals (AZ-900) Cert Prep: 1 Cloud Concepts: <https://www.linkedin.com/learning/certificates/3d132cf79903b41856aa3e4bd5e0849e5753700f792826e2762a7d44c936a2ce?u=42583876>