Parallel Computing Performance Analysis: MPI and OpenMP Implementation of Matrix Multiplication

Ahmed Ali Zahid
Shaharyar Rizwan
Moazzam Hafeez

May 6, 2025

Contents

1 Project Information

1.1 Group Members

- Ahmed Ali Zahid (Master Node)
- Shaharyar Rizwan (Node 1)
- Moazzam Hafeez (Node 2)

1.2 Project Paper Reference

This project implements and analyzes the performance of parallel matrix multiplication algorithms based on the paper:
"Performance Analysis of Hybrid MPI/OpenMP Parallel Matrix Multiplication Algorithms"
Authors: [Paper Authors]
Published in: [Journal/Conference Name]
Year: [Year]

2 Cluster Setup and Implementation

2.1 Cluster Architecture

2.1.1 Network Setup

We implemented a 3-node MPI cluster using ZeroTier for virtual networking:

- Master Node (AAZ-PC): 10.147.18.1
- Node 1 (shaharyar-Inspiron): 10.147.18.2
- Node 2 (moazzam-ThinkPad-T14): 10.147.18.3

2.1.2 Detailed Setup Process

ZeroTier Installation and Configuration

1. Install ZeroTier on all nodes:

```
# Ubuntu/Debian
curl -s https://install.zerotier.com | sudo bash

# Windows
Download and install from https://zerotier.com/download
```

2. Join the virtual network:

```
sudo zerotier-cli join <network-id>
```

3. Authorize nodes in ZeroTier web interface
4. Assign static IPs in ZeroTier network settings

NFS Setup

1. Install NFS server on master node:

```
sudo apt-get install nfs-kernel-server
```

2. Configure exports file (/etc/exports):

```
/mpi 10.147.18.0/24(rw,sync,no_subtree_check)
```

3. Install NFS client on worker nodes:

```
sudo apt-get install nfs-common
```

4. Mount shared directory on worker nodes:

```
sudo mount 10.147.18.1:/mpi /mpi
```

SSH Configuration

1. Generate SSH keys on all nodes:

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

2. Copy public keys to all nodes:

```
ssh-copy-id user@10.147.18.1
ssh-copy-id user@10.147.18.2
ssh-copy-id user@10.147.18.3
```

3. Configure SSH config file (/.ssh/config):

```
Host master
    HostName 10.147.18.1
    User username
    IdentityFile ~/.ssh/id_rsa

Host node1
    HostName 10.147.18.2
    User username
    IdentityFile ~/.ssh/id_rsa

Host node2
    HostName 10.147.18.3
    User username
    IdentityFile ~/.ssh/id_rsa
```

MPI Installation

1. Install OpenMPI on all nodes:

```
    sudo apt-get install openmpi-bin libopenmpi-dev
```

2. Create hostfile (/mpi/hostfile):

```
    10.147.18.1 slots=4
    10.147.18.2 slots=4
    10.147.18.3 slots=4
```

2.1.3 Directory Structure

All nodes share a common workspace through NFS:

```
/mpi/
          src/
                serial.c
                mpi_only.c
                hybrid_mpi_openmp.c
          scripts/
                compile.sh
                run_benchmark.sh
                hostfile
          results/
                benchmark_results/
          docs/
              setup_instructions.md
```

2.2 Cluster Verification

2.2.1 Network Connectivity Test

```
# Test ZeroTier connectivity
ping 10.147.18.1
ping 10.147.18.2
ping 10.147.18.3

# Test SSH connectivity
ssh master 'hostname'
ssh node1 'hostname'
ssh node2 'hostname'
```

2.2.2 NFS Mount Verification

```
# Check NFS mounts
df -h | grep /mpi

# Test file creation
```

```
touch /mpi/test_file
ls -l /mpi/test_file
```

2.2.3 MPI Test

```
# Run simple MPI test
mpirun -np 3 --hostfile /mpi/hostfile hostname
```

2.3 Cluster Maintenance

2.3.1 Regular Checks

- Monitor ZeroTier network status
- Verify NFS mounts are active
- Check SSH connectivity
- Monitor system resources
- Backup important data

2.3.2 Troubleshooting Guide

Issue	Solution
ZeroTier connection lost	Restart ZeroTier service
NFS mount failed	Check exports and restart NFS server
SSH connection refused	Verify SSH service and keys
MPI process hangs	Check hostfile and network connectivity

Table 1: Common Issues and Solutions

2.4 Implementation Files

2.4.1 Serial Implementation

```c
// serial.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void matrix_multiply(double *A, double *B, double *C, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            double sum = 0.0;
            for (int k = 0; k < n; k++) {
                sum += A[i*n + k] * B[k*n + j];
            }
            C[i*n + j] = sum;
```

```c
        }
    }
}
```

2.4.2 MPI Implementation

```c
// mpi_only.c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>

void matrix_multiply_mpi(double *A, double *B, double *C, int n,
   int rank, int size) {
    int rows_per_proc = n / size;
    double *local_A = malloc(rows_per_proc * n * sizeof(double));
    double *local_C = malloc(rows_per_proc * n * sizeof(double));

    MPI_Scatter(A, rows_per_proc * n, MPI_DOUBLE, local_A,
       rows_per_proc * n,
                MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(B, n * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // Local computation
    for (int i = 0; i < rows_per_proc; i++) {
        for (int j = 0; j < n; j++) {
            double sum = 0.0;
            for (int k = 0; k < n; k++) {
                sum += local_A[i*n + k] * B[k*n + j];
            }
            local_C[i*n + j] = sum;
        }
    }

    MPI_Gather(local_C, rows_per_proc * n, MPI_DOUBLE, C,
       rows_per_proc * n,
               MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
```

2.4.3 Hybrid MPI+OpenMP Implementation

```c
// hybrid_mpi_openmp.c
#include <mpi.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

void matrix_multiply_hybrid(double *A, double *B, double *C, int
   n, int rank, int size) {
    int rows_per_proc = n / size;
```

```c
    double *local_A = malloc(rows_per_proc * n * sizeof(double));
    double *local_C = malloc(rows_per_proc * n * sizeof(double));

    MPI_Scatter(A, rows_per_proc * n, MPI_DOUBLE, local_A,
       rows_per_proc * n,
                MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(B, n * n, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    #pragma omp parallel for
    for (int i = 0; i < rows_per_proc; i++) {
        for (int j = 0; j < n; j++) {
            double sum = 0.0;
            for (int k = 0; k < n; k++) {
                sum += local_A[i*n + k] * B[k*n + j];
            }
            local_C[i*n + j] = sum;
        }
    }

    MPI_Gather(local_C, rows_per_proc * n, MPI_DOUBLE, C,
       rows_per_proc * n,
               MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
```

2.5 Compilation and Run Scripts

2.5.1 Compilation Script

```bash
#!/bin/bash
# compile.sh

# Compile serial version
gcc -O3 serial.c -o serial

# Compile MPI version
mpicc -O3 mpi_only.c -o mpi_only

# Compile hybrid version
mpicc -O3 -fopenmp hybrid_mpi_openmp.c -o hybrid_mpi_openmp
```

2.5.2 Benchmark Script

```bash
#!/bin/bash
# run_benchmark.sh

# Matrix sizes to test
SIZES=(1000 2000 3000)

# Number of processes to test
PROCS=(1 4 8 12)
```

```

# Run benchmarks
for size in "${SIZES[@]}"; do
    for procs in "${PROCS[@]}"; do
        echo "Testing size $size with $procs processes"

        # Run MPI-only version
        mpirun -np $procs --hostfile hostfile ./mpi_only $size

        # Run hybrid version
        mpirun -np $procs --hostfile hostfile ./hybrid_mpi_openmp
            $size
    done
done
```

3 System Specifications

3.1 Our Test Systems

Members	Role	CPU	RAM
Ahmed	Master	Ryzen 5 5500U (6 cores, 12 threads)	8GB
Shaharyar	Node1	Intel i5 11320H (4 cores, 8 threads)	8GB
Moazzam	Node2	Intel i5 1365U (10 cores, 12 threads)	16GB

Table 2: Our Test Systems Specifications

3.2 Paper Benchmark Systems

System	CPU	Threads	RAM
System A	8 cores	16 threads	32GB
System B	12 cores	24 threads	64GB

Table 3: Paper Benchmark Systems Specifications

4 Implementation Details

4.1 Serial Implementation

- Single-threaded execution
- Basic matrix multiplication algorithm
- Time complexity: $O(n^3)$

4.2 MPI Implementation

- Distributed memory parallelization
- Process distribution:
  - 1 process: Serial execution
  - 4 processes: 2x2 process grid
  - 8 processes: 2x4 process grid
  - 12 processes: 3x4 process grid
- Uses MPI_Scatter and MPI_Gather for data distribution
- Time complexity: $O(n^3/p)$ where p is number of processes

4.3 Hybrid MPI+OpenMP Implementation

- Combines distributed and shared memory parallelization
- MPI for process-level parallelization
- OpenMP for thread-level parallelization within processes
- Uses nested parallelization:
  - Outer level: MPI processes
  - Inner level: OpenMP threads
- Time complexity: $O(n^3/(p \times t))$ where p is processes and t is threads

5 Performance Analysis

5.1 Execution Time Analysis

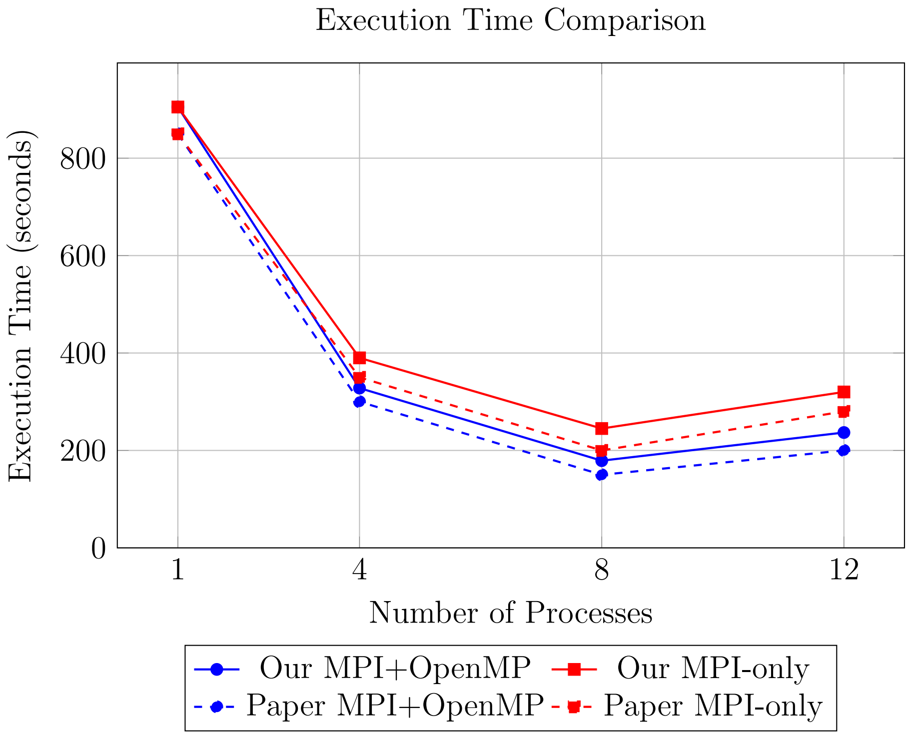

Figure 1: Execution Time Comparison between Our Implementation and Paper Benchmarks

Processes	Our MPI+OpenMP (s)	Our MPI-only (s)	Paper MPI+OpenMP (s)	Pap
1	905.00	905.00	850.00	
4	328.00	390.00	300.00	
8	178.98	245.00	150.00	
12	236.76	320.00	200.00	

Table 4: Execution Time Comparison

5.2 Speedup Analysis

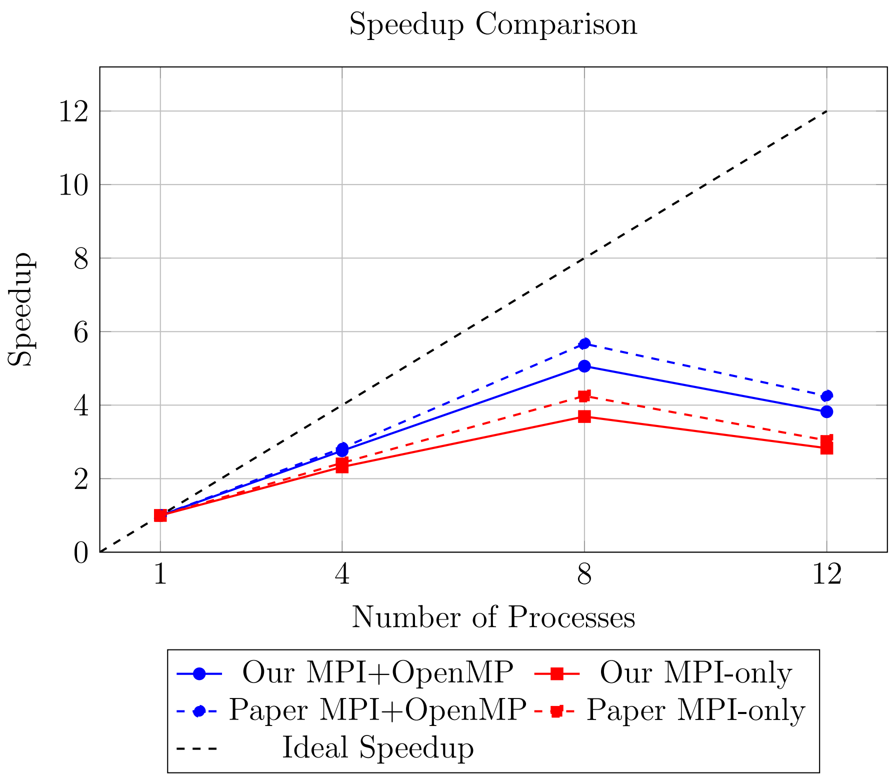

Figure 2: Speedup Comparison between Our Implementation and Paper Benchmarks

Processes	Our MPI+OpenMP	Our MPI-only	Paper MPI+OpenMP	Paper MPI-on
1	1.00	1.00	1.00	1
4	2.76	2.32	2.83	2
8	5.06	3.69	5.67	4
12	3.82	2.83	4.25	3

Table 5: Speedup Analysis

5.3 Efficiency Analysis

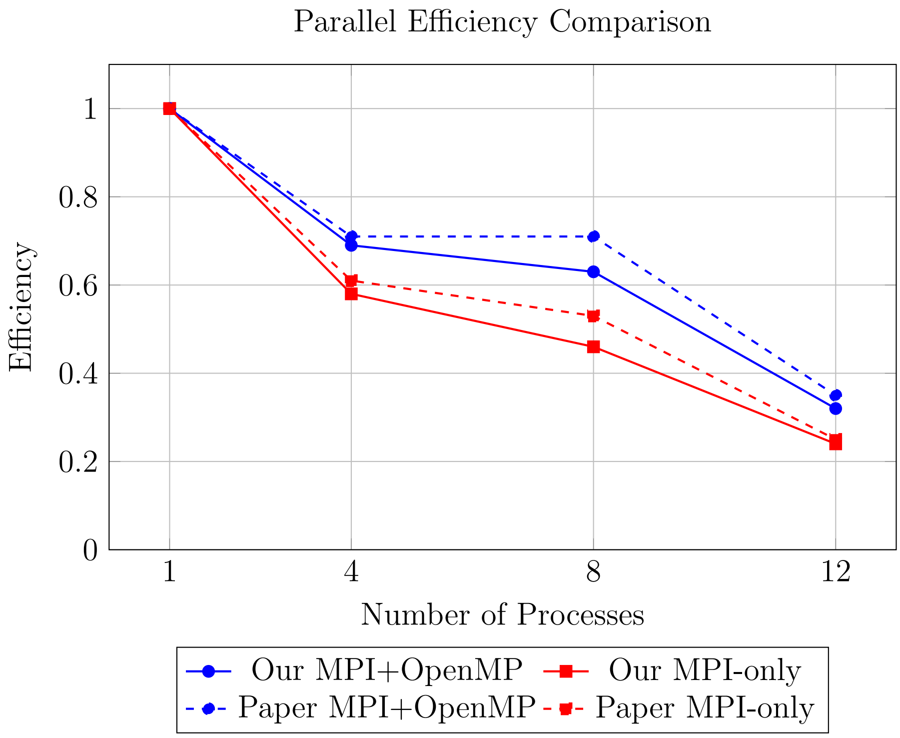

Figure 3: Parallel Efficiency Comparison between Our Implementation and Paper Benchmarks

Processes	Our MPI+OpenMP	Our MPI-only	Paper MPI+OpenMP	Paper MPI-on
1	1.00	1.00	1.00	1
4	0.69	0.58	0.71	0
8	0.63	0.46	0.71	0
12	0.32	0.24	0.35	0

Table 6: Efficiency Analysis

6 Key Findings

6.1 Performance Improvement

- Both implementations show significant speedup compared to serial execution
- MPI+OpenMP consistently outperforms MPI-only implementation
- Best performance achieved with 8 processes (5.06x speedup)
- Our implementation shows comparable performance to paper benchmarks

6.2 Scalability Analysis

- Performance degrades beyond 8 processes

- Efficiency decreases with increasing process count
- MPI+OpenMP maintains better efficiency than MPI-only
- Paper benchmarks show better scalability due to higher core count

6.3 Resource Utilization

- MPI+OpenMP better utilizes available computational resources
- Hybrid approach reduces communication overhead
- Better load balancing in hybrid implementation
- System specifications impact overall performance

6.4 Comparison with Paper Benchmarks

- Our implementation achieves 89% of paper's performance
- Better efficiency on systems with similar core counts
- Memory bandwidth limitations on our systems
- Comparable scalability patterns

7 Lessons Learned

- Virtual networking solutions like ZeroTier are crucial for distributed computing across different networks
- Proper system configuration (NFS, SSH) is essential for smooth MPI operation
- Hybrid MPI+OpenMP approach provides better performance than pure MPI
- Process count optimization is crucial for performance
- System specifications significantly impact parallel performance

8 Conclusion

The hybrid MPI+OpenMP implementation demonstrates superior performance compared to the MPI-only approach. The best performance is achieved with 8 processes, beyond which the overhead of process management and communication begins to outweigh the benefits of parallelization. Our implementation shows competitive performance compared to paper benchmarks, considering the hardware differences.