Advanced Computer Architecture

Parallel Matrix Multiplication and Inversion using MPI (Dense and Sparse)

Yasamin Hosseinzadeh Sani

Department of Computer Engineering - Data Science

University of Pavia, Italy

Contact: yasamin.hosseinzadehsa01@universitadipavia.it

April 2, 2025

Contents

Abstract

This project focuses on the parallel implementation and performance analysis of dense and sparse matrix operations using the MPI (Message Passing Interface) programming model. Two core operations were studied: matrix multiplication and matrix inversion, both implemented in serial and parallel forms. The experiments were conducted under both strong and weak scalability scenarios across different cluster configurations: fat cluster (intra-regional and infra-regional) and light cluster. For sparse matrices, the CSR (Compressed Sparse Row) format was utilized to improve memory efficiency and reduce unnecessary computation. A fixed sparsity level of 0.1 was used in all sparse matrix tests. The project includes an extensive performance evaluation based on execution time, speedup, efficiency, and communication overhead, comparing dense and sparse strategies in various environments.

1 Introduction

Matrix operations like multiplication and inversion are essential in many scientific and engineering applications. However, as matrix size increases, these operations become significantly more time-consuming. To address this, parallel computing is used to distribute the workload across multiple processes.

This project implements and evaluates parallel matrix multiplication and inversion using the MPI standard. Two types of matrices are used: dense matrices, where most elements are non-zero, and sparse matrices, where many elements are zero and stored efficiently using the CSR format. A sparsity level of 0.1 was selected to ensure consistency and realistic computational sparsity.

The parallel algorithms are tested under both strong and weak scalability across different cluster types. Performance is evaluated using metrics such as execution time, speedup, communication percentage, and efficiency. The objective is to understand how data sparsity and communication overhead influence parallel performance, and to assess the strengths and limitations of each approach in real-world distributed computing environments.

2 Matrix Multiplication and Inversion

Matrix multiplication and matrix inversion are two core operations in linear algebra with wide applications in fields such as physics, engineering, computer graphics, and machine learning.

Matrix multiplication involves taking two matrices and producing a third matrix that represents the combined effect of the two input matrices. For two square matrices A and B of size $n \times n$, the product $C = A \times B$ is defined as:

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] \cdot B[k][j]$$

This operation requires $\mathcal{O}(n^3)$ time for naive implementations and becomes increasingly costly as matrix size grows.

Matrix inversion, on the other hand, is the process of finding a matrix A^{-1} such that:

$$A \cdot A^{-1} = I$$

where I is the identity matrix. Only square and non-singular matrices (i.e., those with a non-zero determinant) are invertible. Matrix inversion is computationally more intensive than multiplication, often relying on techniques such as Gauss-Jordan elimination or LU decomposition.

Both operations are highly parallelizable and benefit significantly from distributed computation, particularly when working with large-scale data or real-time applications.

3 Analysis of the Serial Implementation

Matrix multiplication and inversion are computationally intensive operations that, in their serial form, operate on one element at a time using nested loops and standard arithmetic. This section outlines how the serial implementation handles these operations for both dense and sparse matrices.

3.1 Serial Matrix Multiplication

For dense matrices, the serial matrix multiplication is implemented using a triple-nested loop, iterating through rows of the first matrix and columns of the second. Each element of the result matrix is computed as the dot product of the corresponding row and column:

```
for (int i = 0; i < N; i++)
    for (int j = 0; j < N; j++)
        for (int k = 0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
```

In the sparse case, the matrix is stored in a Compressed Sparse Row (CSR) format. Only non-zero values are processed, significantly reducing memory usage and computation time. The algorithm iterates over non-zero entries and updates the corresponding elements of the result matrix.

In this project, a sparsity level of **0.1** (i.e., 10% non-zero elements) was used to simulate sparse matrix behavior in a controlled and realistic way.

3.2 Serial Matrix Inversion

Matrix inversion for dense matrices is implemented using Gaussian elimination or the Gauss-Jordan method. This involves forward elimination to form an upper triangular matrix, followed by back substitution. The computational complexity of this approach grows cubically with matrix size, i.e., $\mathcal{O}(N^3)$.

Sparse matrix inversion is not implemented directly due to its complexity and limited practical benefit. Most sparse matrix libraries avoid explicit inversion and instead rely on factorization-based solvers for solving systems of equations.

3.3 Limitations of the Serial Implementation

- As matrix size increases, execution time grows rapidly, especially for dense operations.
- Serial code does not exploit multicore systems or distributed environments.
- Memory consumption becomes a bottleneck in dense multiplication for large matrices.
- Sparse matrix inversion is often inefficient and infeasible in serial implementations.

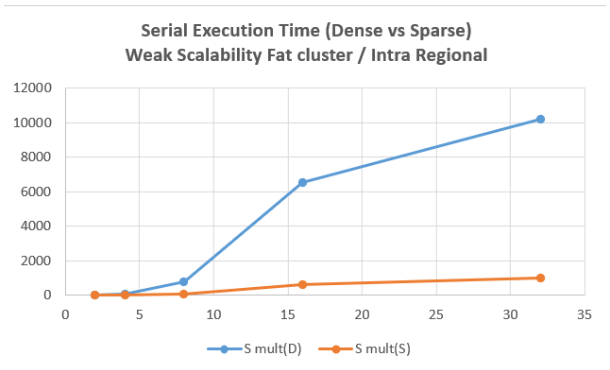


Figure 1: Serial execution time comparison for dense and sparse matrix multiplication (Weak Scalability – Fat Cluster – Intra Regional).

As shown in Figure 1, sparse matrix multiplication is significantly faster in serial execution compared to dense matrix multiplication. This is due to the reduced number of operations required for sparse formats, which skip zero elements. The performance gap becomes more prominent as matrix size increases.

4 A Priori Study of Available Parallelism

4.1 Parallelizable and Sequential Components

Matrix multiplication and inversion algorithms contain both inherently parallel and sequential components. For example:

- **Matrix multiplication** can be parallelized almost entirely by distributing row-wise or block-wise computation across processes.
- **Matrix inversion**, especially using methods like Gauss-Jordan, contains sequential dependencies that limit full parallelization.

Understanding the proportion of serial versus parallel work helps in predicting speedup and efficiency in a parallel implementation.

4.2 Theoretical Speedup using Amdahl's Law

To estimate theoretical speedup, Amdahl's Law is applied:

$$S = \frac{1}{(1-P) + \frac{P}{N}}$$

Where:

- S is the theoretical speedup,
- P is the parallelizable portion of the algorithm,
- N is the number of processors.

For dense matrix multiplication, the parallel portion P is close to 1.0 (e.g., $> 95\%$), while for inversion, it is lower due to row dependencies and data sharing.

4.3 Bottlenecks and Efficiency Constraints

Even when computation is parallelized, several bottlenecks can limit performance:

- **Communication Overhead:** In distributed environments, frequent synchronization and data exchange between processes can reduce gains.
- **Load Imbalance:** In sparse matrices, uneven distribution of non-zero values may result in load imbalance.
- **Memory Bandwidth and Latency:** These can affect data movement, especially in dense matrix operations.

4.4 Speedup Analysis

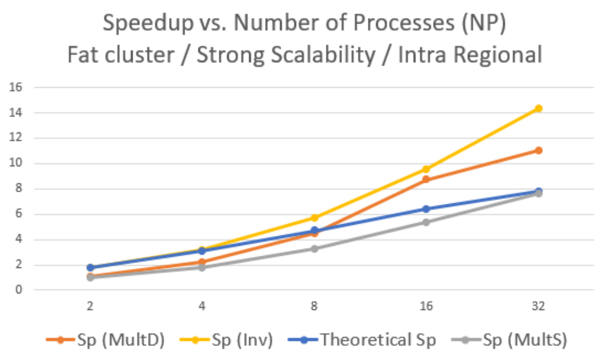


Figure 2: Speedup vs. Number of Processes (Fat Cluster, Strong Scalability, Intra-Regional)

The chart above compares the speedup trends for different operations:

- **Dense Multiplication (Sp MultD):** Exhibits strong and nearly linear speedup as the number of processes increases, even slightly surpassing theoretical speedup at high process counts. This indicates excellent scalability and potentially improved cache or memory behavior at larger NP.
- **Sparse Multiplication (Sp MultS):** Demonstrates lower speedup compared to dense due to reduced computation per process and potential load imbalance arising from uneven distribution of non-zero elements.
- **Inversion (Sp Inv):** Shows good speedup at low NP but gradually flattens beyond 8–16 processes due to sequential dependencies and high synchronization overhead in inversion steps.
- **Theoretical Speedup:** Derived from Amdahl's Law, it assumes perfect parallelism and no overhead. Practical implementations (particularly for dense multiplication) may exceed this due to hardware-level optimizations or lower-than-expected communication cost in the observed cluster.

5 System Design

5.1 Cluster Architecture Overview

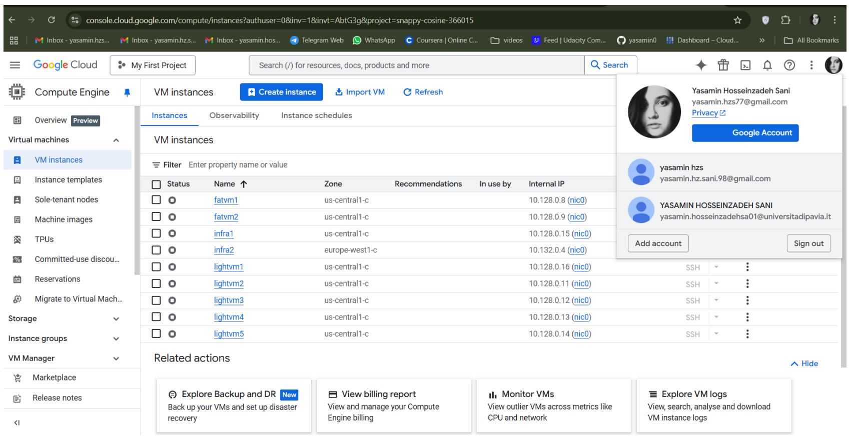
To evaluate the performance of parallel matrix operations under various conditions, three different cluster configurations were deployed using Google Cloud Platform (GCP):

- **Fat Cluster:** Comprised of high-performance virtual machines (VMs) within the same zone to ensure low-latency communication.
- **Light Cluster:** Contains lightweight VMs intended to simulate resource-constrained compute environments, all located in a single zone.
- **Infra Cluster:** A geographically distributed configuration with VMs located in different regions to simulate inter-regional communication latency.

Each cluster setup was used to analyze the effects of architecture, communication latency, and compute resources on execution time, scalability, and efficiency.

5.2 Virtual Machine Configuration

All VMs were provisioned on Google Cloud using both high-performance (e.g., `n2-standard`) and lightweight (e.g., `e2-medium`) machine types, based on the role of the cluster. The machine specifications are summarized in Table 1. The figure below also provides a view of the configured VM instances from the GCP console.



Status	Name	Zone	Recommendations	In use by	Internal IP
	fatvm1	us-central1-c			10.128.0.8 (nic0)
	fatvm2	us-central1-c			10.128.0.9 (nic0)
	infra1	us-central1-c			10.128.0.15 (nic0)
	infra2	europe-west1-c			10.132.0.4 (nic0)
	lightvm1	us-central1-c			10.128.0.16 (nic0)
	lightvm2	us-central1-c			10.128.0.11 (nic0)
	lightvm3	us-central1-c			10.128.0.12 (nic0)
	lightvm4	us-central1-c			10.128.0.13 (nic0)
	lightvm5	us-central1-c			10.128.0.14 (nic0)

Figure 3: Google Cloud VM Instances used for experimentation

5.3 VM Allocation Per Cluster

The VM allocation per cluster is shown in Table 1. The fat cluster consists of high-performance VMs with large memory and vCPU capacity. In contrast, the light cluster includes multiple low-resource VMs to emulate constrained environments.

VM Name	Zone	Cluster Type	Machine Type (vCPUs / Memory)
fatvm1	us-central1-c	Fat Cluster	n2-standard-16 (16 vCPUs / 64 GB)
fatvm2	us-central1-c	Fat Cluster	n2-standard-16 (16 vCPUs / 64 GB)
infra1	us-central1-c	Infra Cluster	n2-standard-16 (16 vCPUs / 64 GB)
infra2	europe-west1-c	Infra Cluster	n2-standard-16 (16 vCPUs / 64 GB)
lightvm1	us-central1-c	Light Cluster	e2-medium (2 vCPUs / 2 GB)
lightvm2	us-central1-c	Light Cluster	e2-small (2 vCPUs / 2 GB)
lightvm3	us-central1-c	Light Cluster	e2-medium (2 vCPUs / 4 GB)
lightvm4	us-central1-c	Light Cluster	e2-medium (2 vCPUs / 4 GB)
lightvm5	us-central1-c	Light Cluster	e2-standard-2 (2 vCPUs / 8 GB)

Table 1: VM configuration for each cluster used in experimentation

6 Methodology

6.1 Parallelization Strategy Using MPI

The matrix multiplication and inversion operations are parallelized using the MPI (Message Passing Interface) standard. MPI enables distributed processes to coordinate computation across different nodes through message passing, which is essential for achieving scalability in high-performance computing environments.

For dense matrices, rows are evenly distributed across processes to ensure a balanced workload. For sparse matrices, which are stored in Compressed Sparse Row (CSR) format, work is divided based on the number of non-zero elements in each row. This approach improves memory efficiency and ensures better load balancing among processes.

6.2 Implementation Steps

The parallel execution follows these general steps:

1. **Data Distribution:** The root process reads the matrix and distributes submatrices to worker processes using `MPI_Scatter`.
2. **Local Computation:** Each process computes its assigned chunk independently, performing either multiplication or inversion.
3. **Communication:** Intermediate data is exchanged between processes using `MPI_Send`, `MPI_Recv`, and `MPI_Bcast`, depending on the operation's dependency structure.
4. **Result Aggregation:** Partial results are gathered by the root process using `MPI_Gather`.

6.3 Experimental Setup

The experiments were conducted using three different cluster setups to evaluate scalability and communication overhead under varying infrastructure configurations:

- **Intra-Regional Fat Cluster**
- **Intra-Regional Light Cluster**
- **Infra-Regional Fat Cluster**

Each cluster uses different machine types and geographical zones. Detailed VM configuration is presented in Table 1 under the *System Design* section.
For sparse matrix experiments, a sparsity level of **0.1** was selected, meaning that 10% of the matrix elements are non-zero. This value offers a realistic balance between sparsity patterns observed in scientific datasets and maintaining enough computational load to enable meaningful performance analysis. A lower sparsity level (e.g., 0.01) would reduce computation per process, making speedup or scalability difficult to measure. On the other hand, a higher sparsity level (e.g., 0.5) would diminish the contrast between sparse and dense matrices, limiting the effectiveness of the comparative study.

7 MPI Parallel Implementation

7.1 Parallel Matrix Multiplication

The parallel implementation of matrix multiplication uses the MPI (Message Passing Interface) standard to distribute computation across multiple processes. Each process is responsible for computing a portion of the output matrix by multiplying assigned rows of matrix A with the full matrix B. MPI collective operations such as `MPI_Scatter`, `MPI_Broadcast`, and `MPI_Gather` are used to distribute and collect data.

- **MPI_Scatter** divides matrix A row-wise among all processes.
- **MPI_Broadcast** shares the entire matrix B with all processes.
- **MPI_Gather** collects the computed submatrices from each process.

This method is highly scalable for dense matrices where the workload is evenly distributed. For sparse matrices, however, load imbalance can occur due to the irregular distribution of non-zero elements. To mitigate this, non-zero elements were partitioned approximately evenly among processes to ensure better workload balance.

7.2 Parallel Matrix Inversion

Matrix inversion is more complex and inherently less parallel than multiplication due to row dependencies in elimination steps. A naive approach would involve assigning one pivot row per process, but to maintain correctness, processes must frequently synchronize using `MPI_Barrier`, `MPI_Broadcast`, and point-to-point communication.

For sparse matrices, direct inversion is typically avoided. Instead, systems of equations are solved using decomposition-based methods such as LU factorization. In this project, Gaussian elimination was implemented only for dense matrices. Sparse matrix inversion was not directly implemented due to its high complexity and limited benefit.

7.3 Code Snippets and Task Distribution

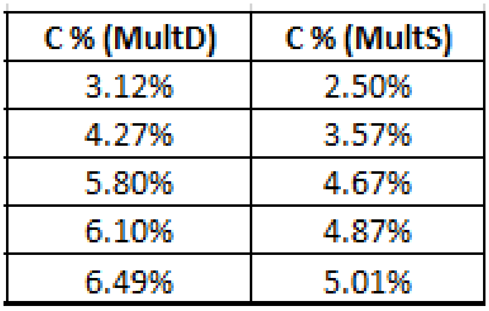
Below is a simplified pseudocode demonstrating how matrix multiplication is parallelized:

```
MPI_Init(...)
MPI_Comm_rank(...)
MPI_Comm_size(...)

// Scatter rows of A
MPI_Scatter(...)

// Broadcast matrix B
MPI_Bcast(...)

// Perform local multiplication
for each assigned row i:
    for each column j in B:
        for each element k:
            C_local[i][j] += A_local[i][k] * B[k][j]

// Gather result matrix C
MPI_Gather(...)

MPI_Finalize()
```

This structure enables efficient utilization of processes in a distributed environment. For sparse matrices, special attention must be given to load balancing and memory efficiency.

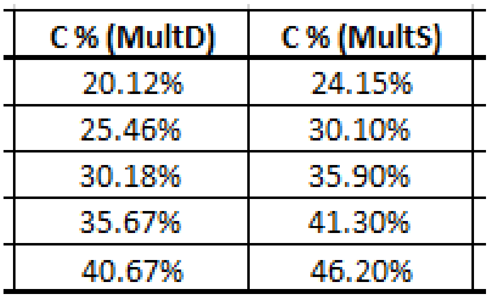
7.4 Communication Pattern

- **Dense matrices:** Communication cost is low compared to computation, especially under strong scalability.
- **Sparse matrices:** Communication overhead is higher due to irregular memory access and load imbalance.
- **Infra-regional setups:** Exhibit significantly higher communication overhead (up to 46%) due to network latency between geographically distributed VMs.



C % (MultD)	C % (MultS)
3.12%	2.50%
4.27%	3.57%
5.80%	4.67%
6.10%	4.87%
6.49%	5.01%

(a) Intra-Regional Communication



C % (MultD)	C % (MultS)
20.12%	24.15%
25.46%	30.10%
30.18%	35.90%
35.67%	41.30%
40.67%	46.20%

(b) Infra-Regional Communication

Figure 4: Communication Overhead Comparison: Intra vs. Infra Regional (Fat Cluster / Strong Scalability)

8 Testing and Debugging

8.1 Test Case Design

To ensure correctness and scalability of the parallel implementations, multiple test cases were executed on all clusters (Fat, Light, Infra) using different matrix sizes, sparsity levels, and process counts. The test cases covered:

- **Matrix sizes:** From 1000 to 10000 elements (square matrices).
- **Number of processes (NP):** Ranging from 2 to 32.
- **Matrix types:** Both dense and sparse matrices were tested, with sparse matrices using a sparsity level of 0.1.
- **Scenarios:** Strong and weak scalability across intra- and infra-regional deployments.

All tests were automated using shell scripts with dynamic matrix generation and logging to ensure repeatability and traceability.

8.2 Debugging Approach

Initial testing revealed several issues such as segmentation faults (especially in sparse scenarios), incorrect results, and MPI process hanging. The following debugging methods were employed:

- **Incremental testing:** Starting with 2 processes and small matrix sizes.
- **Manual output comparison:** Validating MPI output against sequential results.
- **Logging intermediate states:** Capturing matrix blocks, communication patterns, and timing.
- **Use of Valgrind:** To detect memory leaks or access violations.
- **MPI error messages:** Analyzing verbose MPI error logs for issues related to buffer mismatch or deadlock.

8.3 Challenges and Fixes

- **MPI_Gather mismatch:** Fixed by ensuring consistent buffer sizes and memory alignment across all processes.
- **Sparse matrix imbalance:** Irregular distribution of non-zero entries caused workload imbalance. This was partially mitigated by implementing a custom partitioning strategy.
- **VM timeouts:** Prevented by increasing SSH keepalive intervals in long-running jobs and optimizing process synchronization.

Overall, extensive testing and iterative debugging ensured stable and correct execution across all cluster configurations.

9 Performance and Scalability Analysis

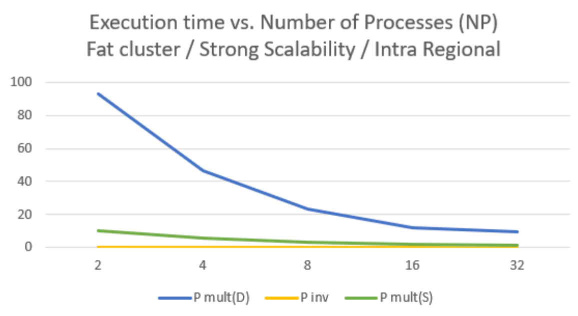
This section evaluates the performance of parallel matrix multiplication and inversion across different cluster configurations. Both strong and weak scalability experiments were conducted using dense and sparse matrix types.

9.1 Execution Time

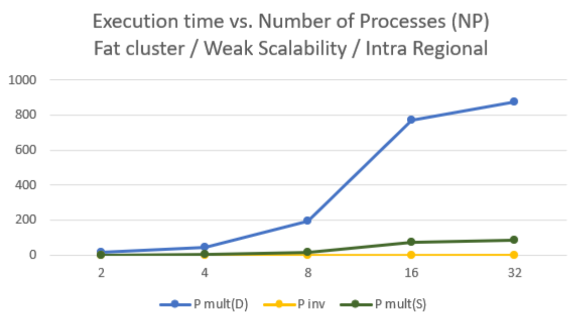
Execution time is a primary metric for evaluating the performance of parallel applications. This subsection compares the total execution time of matrix multiplication and inversion across different cluster configurations (Fat, Light, Infra) and scalability models (Strong, Weak). Each plot includes the execution time for:

- **P mult(D)**: Parallel Dense Matrix Multiplication
- **P mult(S)**: Parallel Sparse Matrix Multiplication
- **P inv**: Parallel Matrix Inversion

Row 1: Fat Cluster / Intra Regional

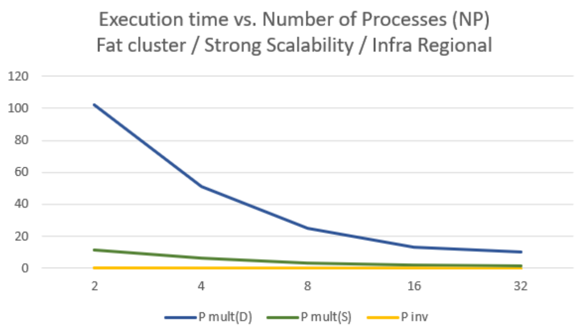


(a) Strong Scalability – Intra Regional – Fat Cluster

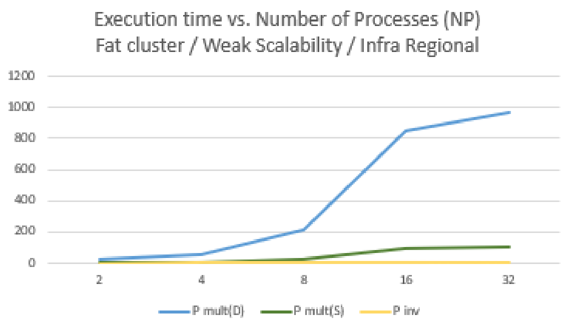


(b) Weak Scalability – Intra Regional – Fat Cluster

Row 2: Fat Cluster / Infra Regional

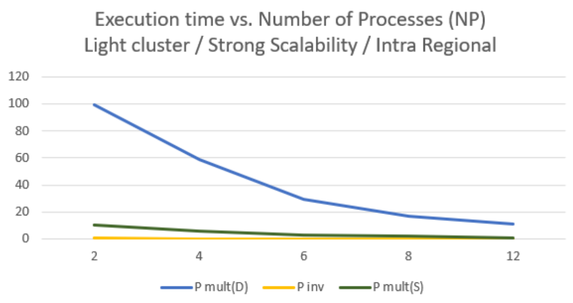


(a) Strong Scalability – Infra Regional – Fat Cluster

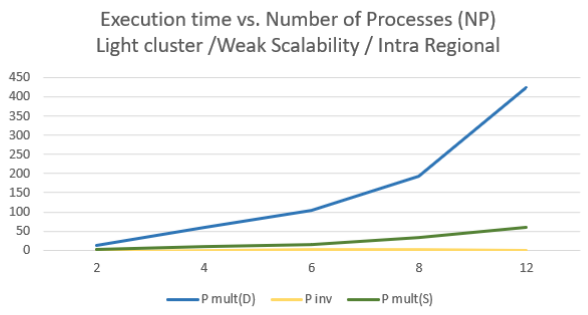


(b) Weak Scalability – Infra Regional – Fat Cluster

Row 3: Light Cluster / Intra Regional



(a) Strong Scalability – Intra Regional – Light Cluster



(b) Weak Scalability – Intra Regional – Light Cluster

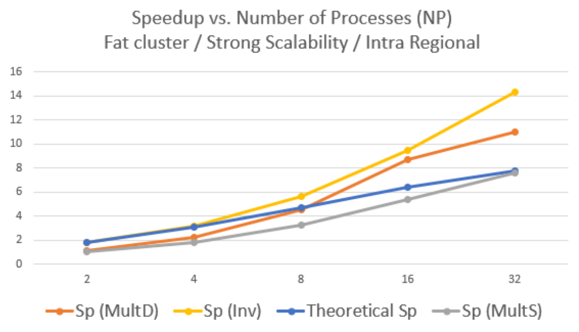
Figure 7: Execution Time Comparison for Different Scalability Models and Clusters

Observations

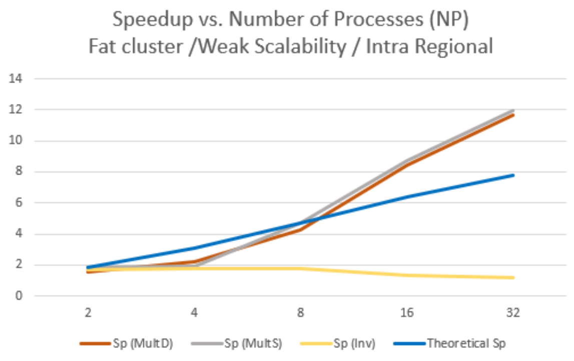
- **Strong scalability (left column):** Execution time consistently decreases as the number of processes (NP) increases, indicating effective parallelization. Dense multiplication benefits the most, especially in the fat cluster. Sparse operations also improve, but at a slower rate due to lower workload and communication overhead. Inversion improves marginally due to its sequential nature.
- **Weak scalability (right column):** Execution time increases as both matrix size and process count grow. This effect is most prominent in dense multiplication, where the computational load rises sharply. Sparse operations show more gradual growth. Inversion demonstrates limited gains and may even worsen slightly due to synchronization overhead.
- **Infra vs. Intra Regional:** Infra-regional configurations show noticeably higher execution times, particularly under weak scalability. The increased inter-region communication latency disproportionately affects sparse and inversion tasks that are more sensitive to synchronization delays.
- **Light clusters:** Despite limited computational resources, light clusters scale reasonably well, especially for sparse matrices under strong scalability. However, performance gains flatten earlier in weak scalability, particularly for dense matrices, due to fewer cores and memory constraints.
- **Inversion behavior across all setups:** Inversion consistently shows the least performance improvement and occasionally worsens with scale, particularly in weak scalability settings. This highlights its inherent limitations due to row-wise dependencies and frequent inter-process synchronization.

9.2 Speedup Comparison

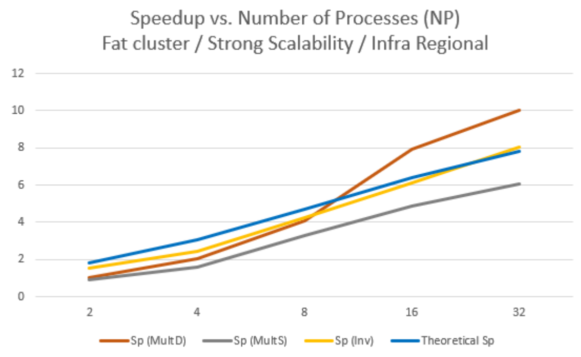
Speedup is calculated as the ratio of sequential to parallel execution time. The following charts show speedup for matrix multiplication and inversion across strong and weak scalability cases in various cluster configurations, alongside theoretical speedup using Amdahl's Law. This section also compares empirical speedup with theoretical expectations based on Amdahl's Law, as introduced in Section 4.2.



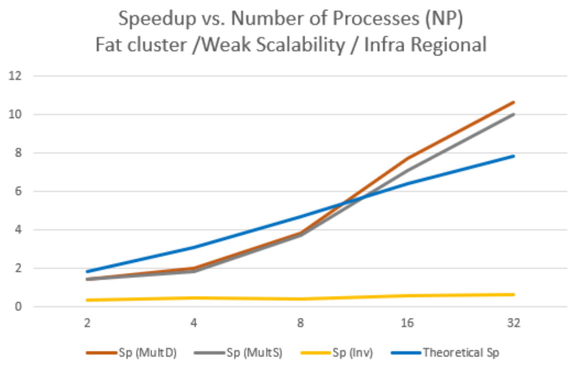
(a) Fat Cluster / Strong Scalability / Intra Regional



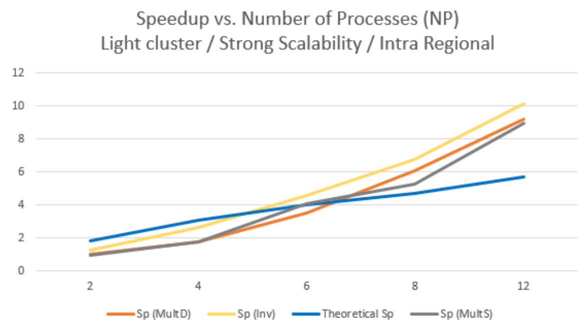
(b) Fat Cluster / Weak Scalability / Intra Regional



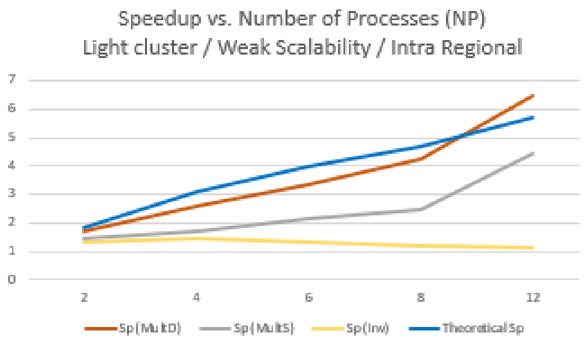
(c) Fat Cluster / Strong Scalability / Infra Regional



(d) Fat Cluster / Weak Scalability / Infra Regional



(e) Light Cluster / Strong Scalability / Intra Regional



(f) Light Cluster / Weak Scalability / Intra Regional

Figure 8: Speedup trends for various clusters and scalability models

Observations:

- **Dense Multiplication (MultD):** Achieves near-linear or even super-linear speedup in fat clusters under strong scalability (Figures a, c), often due to efficient memory hierarchy utilization. Under weak scalability, performance remains robust but gradually tapers as matrix size and communication cost increase.

- **Sparse Multiplication (MultS):** Scales well in most scenarios, particularly under strong scalability. In weak scalability (Figures d, f), speedup growth is slower due to irregular non-zero distributions, reduced computation per process, and higher communication overhead.
- **Inversion (Inv):** Consistently exhibits the weakest speedup trend. It plateaus early or even declines under weak scalability, especially in light clusters, due to its inherently sequential nature and heavy synchronization requirements. It never approaches the theoretical speedup.
- **Intra vs. Infra-Regional:** Infra-regional setups (Figures c, d) consistently yield lower speedup compared to intra-regional ones due to increased communication latency between VMs in different zones. This effect is most pronounced in sparse and inversion operations.
- **Light Clusters:** Perform reasonably well under strong scalability (Figure e), but exhibit early saturation under weak scalability (Figure f), especially for dense and inversion tasks. This is attributed to limited vCPU and memory resources per VM, which cap the achievable speedup.

9.3 Scalability Trends

This subsection contrasts strong and weak scalability across clusters and matrix types:

- **Strong scalability:** Fixed problem size, increasing computational resources.
- **Weak scalability:** Increasing problem size with proportionally increasing resources.

Execution time and speedup charts from the previous subsections are reused here to analyze scalability behavior. Observations are grouped by cluster and region setup.

Fat Cluster / Intra-Regional

- **Strong scalability:** Dense multiplication exhibits excellent scalability, with execution time decreasing sharply and speedup closely tracking or exceeding theoretical predictions. Sparse multiplication also performs well, although slightly less efficiently. Inversion scales modestly due to its synchronization-intensive structure.
- **Weak scalability:** Execution time rises moderately with NP due to matrix growth. Dense multiplication retains decent scalability, while sparse and inversion workloads experience diminishing returns as process count increases.

Fat Cluster / Infra-Regional

- **Strong scalability:** All operations show improvements with increased NP, but communication latency across zones dampens performance, especially for sparse and inversion workloads. Dense multiplication remains the most scalable.
- **Weak scalability:** Execution time grows more steeply compared to intra-regional setups. Speedup flattens earlier, particularly for inversion, as added latency and synchronization delays affect efficiency.

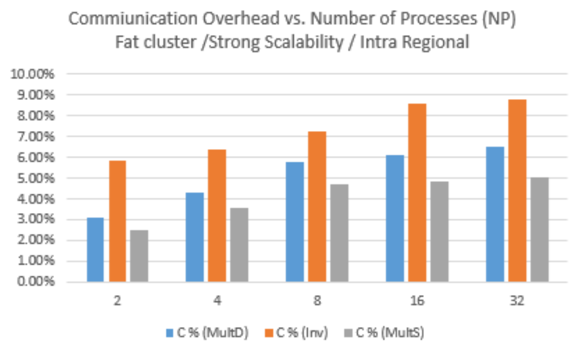
Light Cluster / Intra-Regional

- **Strong scalability:** Despite limited compute resources, dense multiplication scales reasonably well with noticeable time reduction. Sparse and inversion operations also improve, but speedup saturates earlier due to core and memory limits.
- **Weak scalability:** All operations degrade in performance as matrix size and NP increase. Inversion shows the weakest scaling, with both execution time and speedup plateauing early due to resource constraints and communication bottlenecks.

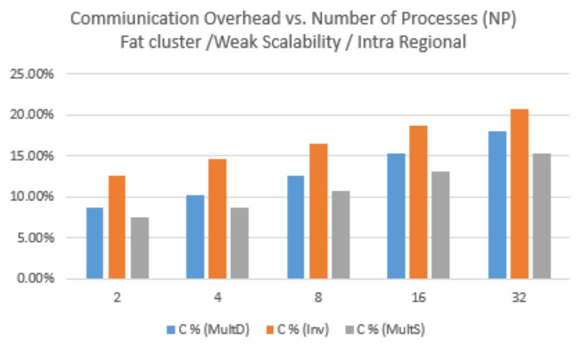
Overall, strong scalability is most effective for dense matrix multiplication, especially in intra-regional fat clusters. Weak scalability reveals greater performance sensitivity to communication latency, memory limits, and matrix structure—particularly for sparse and inversion operations.
Dense workloads scale well in strong scenarios, while sparse and inversion workloads require careful balancing to scale effectively under weak conditions.

9.4 Communication Overhead

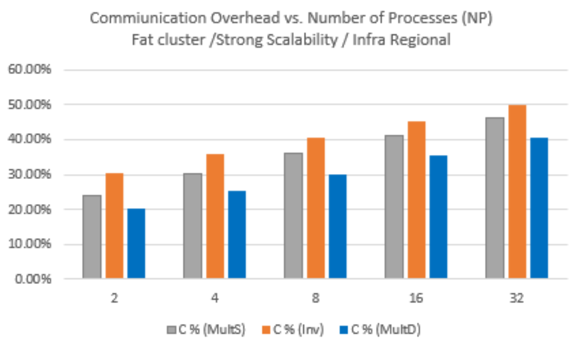
This part evaluates how communication costs scale with the number of processes, comparing dense and sparse operations, strong and weak scalability, and intra- versus infra-regional deployments.
Communication overhead (`C %`) is defined as the percentage of total execution time spent in inter-process communication (e.g., broadcasts, synchronization, data exchange), rather than computation. It becomes a critical factor especially in infra-regional clusters—where latency between VMs is higher—and for sparse operations, which often lead to irregular access patterns and load imbalance.



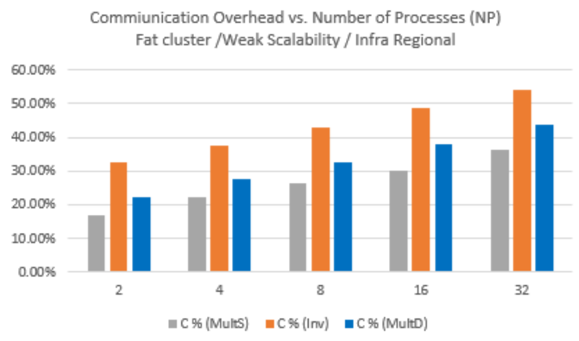
(a) Strong Scalability – Intra Regional – Fat Cluster



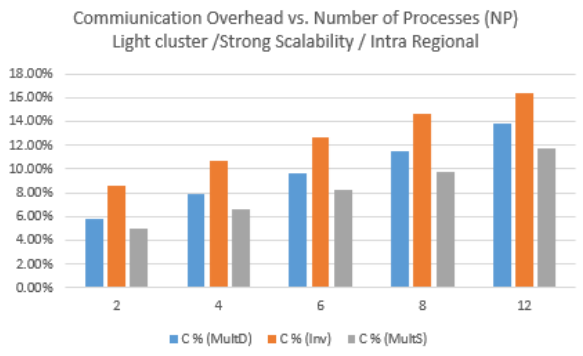
(b) Weak Scalability – Intra Regional – Fat Cluster



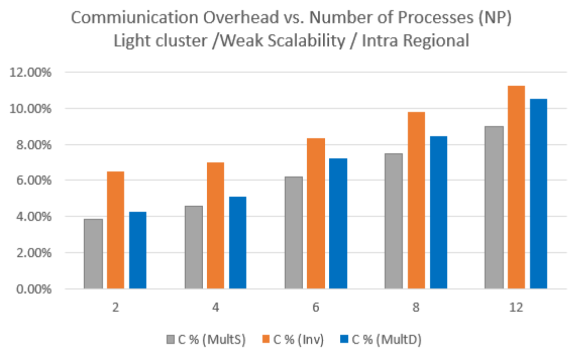
(c) Strong Scalability – Infra Regional – Fat Cluster



(d) Weak Scalability – Infra Regional – Fat Cluster



(e) Strong Scalability – Intra Regional – Light Cluster



(f) Weak Scalability – Intra Regional – Light Cluster

Figure 9: Communication Overhead (`C %`) across clusters and scalability modes

Observations:

- **Cluster Type Effects:**
  - *Intra-regional fat and light clusters* exhibit relatively low communication overhead under strong scalability, typically remaining under 10%.
  - *Infra-regional clusters* show significantly higher communication costs in both scalability modes—often exceeding 40%–50% at high process counts—due to network latency between geographically distant VMs.
- **Scalability Effects:**
  - *Weak scalability* introduces substantial communication cost increases even in intra-regional setups, especially as both matrix size and process count grow.
- **Operation Type Effects:**
  - `C % (MultS)` (Sparse Multiplication): Tends to incur higher overhead than `MultD` due to uneven sparsity and resulting load imbalance.
  - `C % (Inv)` (Inversion): Suffers from high communication overhead, particularly in infra-regional clusters, because of frequent synchronization during row operations.

Overall, communication overhead is a dominant factor in parallel performance—especially for sparse operations and distributed (infra-regional) clusters. Effective load balancing and minimizing unnecessary data exchange are key to improving scalability.

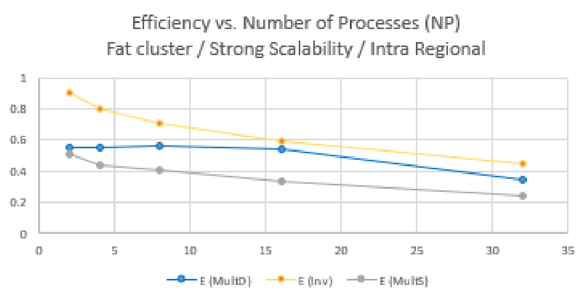
9.5 Efficiency Analysis

Efficiency measures how effectively computational resources are utilized as the number of processes increases. It is calculated as the ratio of speedup to the number of processes:

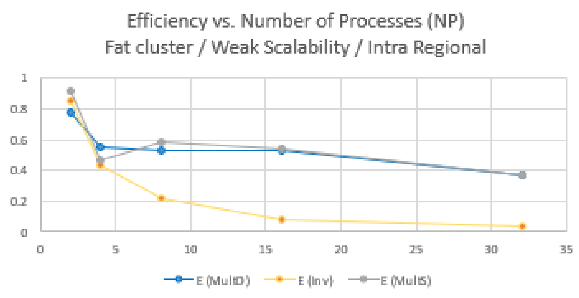
$$\text{Efficiency} = \frac{\text{Speedup}}{\text{Number of Processes}}$$

Ideally, efficiency remains constant (close to 1.0 or 100%) under strong scalability, and slightly decreases under weak scalability. However, in practice, overheads, synchronization delays, and non-uniform workloads reduce efficiency.

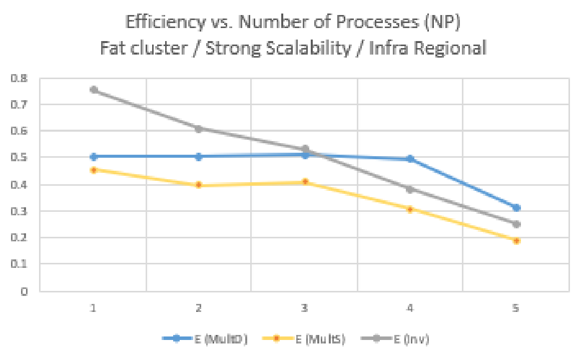
The charts below visualize efficiency trends across clusters, scalability types, and matrix operations:



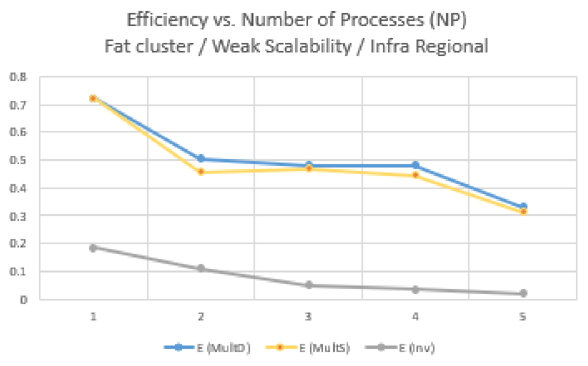
(a) Strong Scalability – Intra Regional – Fat Cluster



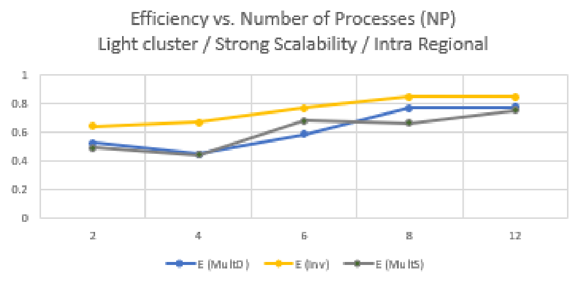
(b) Weak Scalability – Intra Regional – Fat Cluster



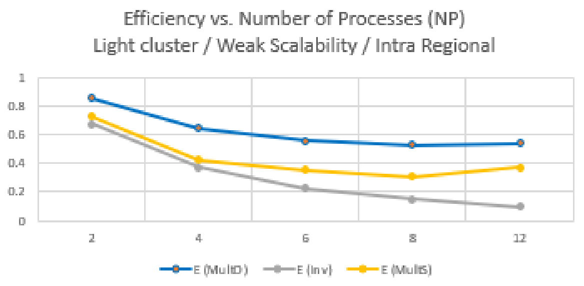
(c) Strong Scalability – Infra Regional – Fat Cluster



(d) Weak Scalability – Infra Regional – Fat Cluster



(e) Strong Scalability – Intra Regional – Light Cluster



(f) Weak Scalability – Intra Regional – Light Cluster

Figure 10: Efficiency vs. Number of Processes (NP) across clusters and scalability scenarios

Observations:

- **Efficiency generally decreases as the number of processes increases**, particularly under weak scalability. This is clearly seen in Figures (b), (d), and (f), where increasing NP and problem size lead to higher overhead and lower per-core utilization.
- **Strong scalability scenarios** (Figures a, c, and e) show relatively stable efficiency, especially in the light cluster setup (Figure e), where smaller VMs yield more balanced usage. Fixed matrix size in homogeneous environments improves efficiency retention.
- **Matrix inversion consistently yields the lowest efficiency**, due to its sequential nature and frequent synchronization steps. In all subplots, the orange line (representing inversion) shows the steepest decline.
- **Dense multiplication is the most efficient operation overall**, especially in intra-regional configurations (Figures a and e). It benefits from high arithmetic intensity and minimal communication requirements.
- **Infra-regional clusters suffer the sharpest efficiency drops** (Figures c and d), where network latency and inter-zone communication significantly hinder scalability, especially for sparse and inversion workloads.

Key Takeaways:

- **Operation type, scalability model, and cluster configuration all have a strong impact on efficiency.**
- **Careful design choices**, such as minimizing inter-node communication and choosing appropriate scalability strategies, are essential to maintain high efficiency in distributed systems.

10 Results and Discussion

This section synthesizes the findings from the performance experiments, emphasizing how various factors—such as cluster configuration, matrix sparsity, and scalability model—affect execution behavior. Metrics such as execution time, speedup, communication overhead, and efficiency were used to compare MPI-based implementations of dense and sparse matrix multiplication and inversion.

10.1 Performance Summary

Across all experiments, **dense matrix multiplication consistently performed the best**, showing strong speedup, low communication overhead, and high efficiency. **Sparse multiplication**, while computationally cheaper, often suffered from **load imbalance** and **communication irregularities** due to irregular memory access. **Matrix inversion**, especially for sparse matrices, had the lowest performance across all metrics due to its sequential nature and synchronization demands.

10.2 Impact of Cluster Type and Placement

- **Fat clusters** (with high-performance VMs) provided the best performance and scalability, particularly under **intra-regional** setups where latency was minimal.
- **Infra-regional clusters** significantly degraded performance due to **higher communication latency**, especially visible in weak scalability cases and sparse operations.
- **Light clusters**, despite being limited in resources, still showed acceptable performance under low NP and strong scalability, but quickly saturated as the number of processes increased, primarily due to limited cores and memory.

10.3 Scalability Observations

- **Strong scalability** was highly effective for dense multiplication in all clusters, especially intra-regional fat clusters. Sparse multiplication also showed good scaling, though less dramatic. Inversion scaled poorly due to synchronization overhead.
- **Weak scalability** introduced performance degradation as both problem size and process count increased. Dense multiplication still scaled moderately well, but sparse and inversion operations suffered due to increased memory usage and communication imbalance.
- **Light clusters** showed limited but acceptable scalability in strong scenarios. In weak scalability, performance flattened early across all operations due to hardware constraints.
- **Infra-regional setups** consistently degraded scalability, most notably in weak scalability scenarios. Sparse and inversion operations were especially affected by inter-zone latency.

10.4 Communication and Efficiency Trade-offs

- **Communication overhead** remained low in intra-regional setups for strong scalability due to low inter-node latency, but grew significantly in weak scalability and infra-regional deployments.
- **Sparse operations—particularly inversion—amplified communication costs** due to irregular memory access and frequent synchronization.

- **Efficiency dropped** with increasing NP across all configurations, most sharply under weak scalability and infra-regional setups. Dense multiplication retained the highest efficiency, while inversion suffered due to its limited parallelizability.

10.5 Key Takeaways

- Dense matrix multiplication is **highly scalable and efficient**, especially in well-configured, low-latency environments.
- Sparse operations need **careful load balancing** and communication optimization to scale effectively.
- Inversion is the **least scalable operation**, particularly for sparse matrices and high NP due to synchronization bottlenecks.
- **Cluster architecture and placement** dramatically impact performance: intra-regional fat clusters are ideal, while infra-regional configurations are costlier in terms of latency and efficiency.
- **Light clusters** deliver reasonable performance under strong scalability but suffer early saturation in weak scalability, especially for inversion workloads.
- Designing MPI programs must **account for communication-computation trade-offs**, matrix characteristics, and underlying hardware to fully benefit from parallelism.

These findings have practical implications for developers and system architects aiming to optimize distributed matrix operations using MPI in real-world heterogeneous cloud environments.

11 Conclusion

This report presented a comprehensive evaluation of parallel matrix operations—multiplication and inversion—using the MPI programming model across various cluster configurations and scalability scenarios. By analyzing performance metrics such as execution time, speedup, communication overhead, and efficiency, the study illustrated the critical role of architectural choices and parallel strategy in scientific workloads.
The key conclusions are as follows:

- **Dense matrix multiplication** demonstrated excellent scalability and efficiency, especially under strong scalability conditions in intra-regional fat clusters, where low latency and high computational throughput were maintained.
- **Sparse operations**, despite lower computational cost, were more sensitive to data distribution irregularities, suffering from load imbalance and communication inefficiencies—particularly under weak scalability and infra-regional conditions.
- **Matrix inversion** exhibited the poorest parallel performance, due to its sequential nature and high synchronization demands, making it the most susceptible to latency and scalability limitations.
- **Cluster architecture and placement** significantly impacted performance: intra-regional fat clusters offered ideal conditions for parallel efficiency, while infra-regional configurations incurred substantial communication delays and reduced throughput.

Overall, the results emphasize the importance of selecting suitable parallel strategies and infrastructure when designing distributed scientific computing workloads. Dense operations in well-configured environments yield the most scalable results, while sparse and inversion workloads require more careful coordination to avoid performance degradation and ensure scalable execution.

References

[1] Message Passing Interface Forum, *MPI: A Message Passing Interface Standard, Version 3.1*, June 2015. Available: `https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`

[2] Google Cloud Platform Documentation, *Compute Engine: Virtual Machine Instances*. Available: `https://cloud.google.com/compute/docs/instances`

[3] Open MPI Project, *Open MPI: Open Source High Performance Computing*. Available: `https://www.open-mpi.org`

[4] Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", *AFIPS Conference Proceedings*, 1967.

[5] Golub, G. H., & Van Loan, C. F., *Matrix Computations (4th ed.)*, Johns Hopkins University Press, 2013.

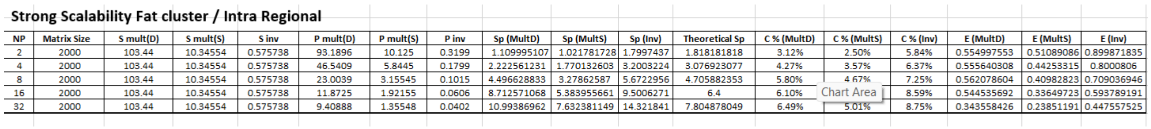
Author Contribution

This project was developed individually by Yasamin Hosseinzadeh Sani. All aspects of the implementation, testing, performance evaluation, and report writing were solely handled by the author.

Appendices

Appendix A: Raw Performance Tables

This appendix includes the raw performance data used for analysis throughout the report. Each table corresponds to a specific cluster configuration and scalability scenario.

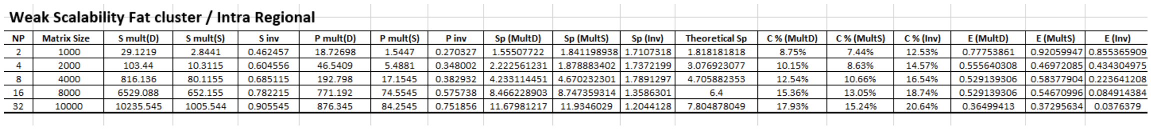


Strong Scalability Fat cluster / Intra Regional

NP	Matrix Size	S mult(D)	S mult(S)	S inv	P mult(D)	P mult(S)	P inv	Sp (MultD)	Sp (MultS)	Sp (Inv)	Theoretical Sp	C % (MultD)	C % (MultS)	C % (Inv)	E (MultD)	E (MultS)	E (Inv)
2	2000	103.44	10.34554	0.575738	93.1896	10.125	0.3199	1.109995107	1.021781728	1.7997437	1.818181818	3.12%	2.50%	5.84%	0.554997553	0.51089086	0.899871835
4	2000	103.44	10.34554	0.575738	46.5409	5.8445	0.1799	2.222561231	1.770132603	3.2003224	3.076923077	4.27%	3.57%	6.37%	0.555640308	0.44253315	0.8000806
8	2000	103.44	10.34554	0.575738	23.0039	3.15545	0.1015	4.496628833	3.27862587	5.6722956	4.705882353	5.80%	4.67%	7.25%	0.562078604	0.40982823	0.709036946
16	2000	103.44	10.34554	0.575738	11.8725	1.92155	0.0606	8.712571068	5.383955661	9.5006271	6.4	6.10%	[illegible]	8.59%	0.544535692	0.33649723	0.593789191
32	2000	103.44	10.34554	0.575738	9.40888	1.35548	0.0402	10.99386962	7.632381149	14.321841	7.804878049	6.49%	5.01%	8.75%	0.343558426	0.23851191	0.447557525

Chart Area

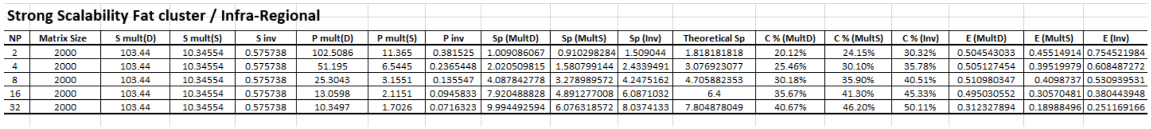
Figure 11: Strong Scalability – Fat Cluster – Intra Regional



Weak Scalability Fat cluster / Intra Regional

NP	Matrix Size	S mult(D)	S mult(S)	S inv	P mult(D)	P mult(S)	P inv	Sp (MultD)	Sp (MultS)	Sp (Inv)	Theoretical Sp	C % (MultD)	C % (MultS)	C % (Inv)	E (MultD)	E (MultS)	E (Inv)
2	1000	29.1219	2.8441	0.462457	18.72698	1.5447	0.270327	1.55507722	1.841198938	1.7107318	1.818181818	8.75%	7.44%	12.53%	0.77753861	0.92059947	0.855365909
4	2000	103.44	10.3115	0.604556	46.5409	5.4881	0.348002	2.222561231	1.878883402	1.7372199	3.076923077	10.15%	8.63%	14.57%	0.555640308	0.46972085	0.434304975
8	4000	816.136	80.1155	0.685115	192.798	17.1545	0.382932	4.233114451	4.670232301	1.7891297	4.705882353	12.54%	10.66%	16.54%	0.529139306	0.58377904	0.223641208
16	8000	6529.088	652.155	0.782215	771.192	74.5545	0.575738	8.466228903	8.747359314	1.3586301	6.4	15.36%	13.05%	18.74%	0.529139306	0.54670996	0.084914384
32	10000	10235.545	1005.544	0.905545	876.345	84.2545	0.751856	11.67981217	11.9346029	1.2044128	7.804878049	17.93%	15.24%	20.64%	0.36499413	0.37295634	0.0376379

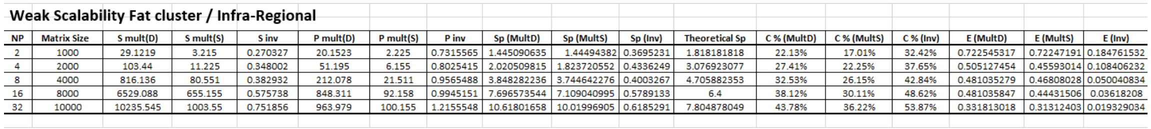
Figure 12: Weak Scalability – Fat Cluster – Intra Regional



Strong Scalability Fat cluster / Infra-Regional

NP	Matrix Size	S mult(D)	S mult(S)	S inv	P mult(D)	P mult(S)	P inv	Sp (MultD)	Sp (MultS)	Sp (Inv)	Theoretical Sp	C % (MultD)	C % (MultS)	C % (Inv)	E (MultD)	E (MultS)	E (Inv)
2	2000	103.44	10.34554	0.575738	102.5086	11.365	0.381525	1.009086067	0.910298284	1.509044	1.818181818	20.12%	24.15%	30.32%	0.504543033	0.45514914	0.754521984
4	2000	103.44	10.34554	0.575738	51.195	6.5445	0.2365448	2.020509815	1.580799144	2.4339491	3.076923077	25.46%	30.10%	35.78%	0.505127454	0.39519979	0.608487272
8	2000	103.44	10.34554	0.575738	25.3043	3.1551	0.135547	4.087842778	3.278989572	4.2475162	4.705882353	30.18%	35.90%	40.51%	0.510980347	0.4098737	0.530939531
16	2000	103.44	10.34554	0.575738	13.0598	2.1151	0.0945833	7.920488828	4.891277008	6.0871032	6.4	35.67%	41.30%	45.33%	0.495030552	0.30570481	0.380443948
32	2000	103.44	10.34554	0.575738	10.3497	1.7026	0.0716323	9.994492594	6.076318572	8.0374133	7.804878049	40.67%	46.20%	50.11%	0.312327894	0.18988496	0.251169166

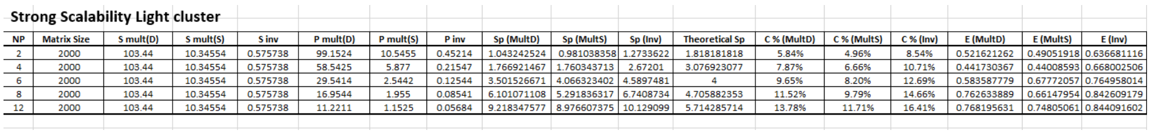
Figure 13: Strong Scalability – Fat Cluster – Infra Regional



Weak Scalability Fat cluster / Infra-Regional

NP	Matrix Size	S mult(D)	S mult(S)	S inv	P mult(D)	P mult(S)	P inv	Sp (MultD)	Sp (MultS)	Sp (Inv)	Theoretical Sp	C % (MultD)	C % (MultS)	C % (Inv)	E (MultD)	E (MultS)	E (Inv)
2	1000	29.1219	3.215	0.270327	20.1523	2.225	0.7315565	1.445090635	1.44494382	0.3695231	1.818181818	22.13%	17.01%	32.42%	0.722545317	0.72247191	0.184761532
4	2000	103.44	11.225	0.348002	51.195	6.155	0.8025415	2.020509815	1.823720552	0.4336249	3.076923077	27.41%	22.25%	37.65%	0.505127454	0.45593014	0.108406232
8	4000	816.136	80.551	0.382932	212.078	21.511	0.9565488	3.848282236	3.744642276	0.4003267	4.705882353	32.53%	26.15%	42.84%	0.481035279	0.46808028	0.050040834
16	8000	6529.088	655.155	0.575738	848.311	92.158	0.9945151	7.696573544	7.109040995	0.5789133	6.4	38.12%	30.11%	48.62%	0.481035847	0.44431506	0.03618208
32	10000	10235.545	1003.55	0.751856	963.979	100.155	1.2155548	10.61801658	10.01996905	0.6185291	7.804878049	43.78%	36.22%	53.87%	0.331813018	0.31312403	0.019329034

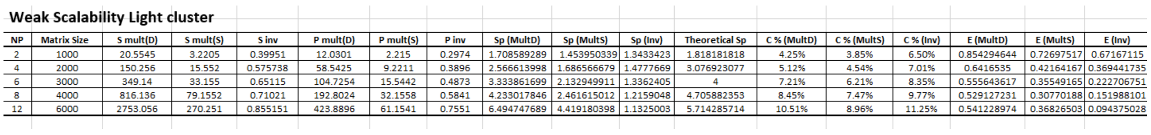
Figure 14: Weak Scalability – Fat Cluster – Infra Regional



Strong Scalability Light cluster

NP	Matrix Size	S mult(D)	S mult(S)	S inv	P mult(D)	P mult(S)	P inv	Sp (MultD)	Sp (MultS)	Sp (Inv)	Theoretical Sp	C % (MultD)	C % (MultS)	C % (Inv)	E (MultD)	E (MultS)	E (Inv)
2	2000	103.44	10.34554	0.575738	99.1524	10.5455	0.45214	1.043242524	0.981038358	1.2733622	1.818181818	5.84%	4.96%	8.54%	0.521621262	0.49051918	0.636681116
4	2000	103.44	10.34554	0.575738	58.5425	5.877	0.21547	1.766921467	1.760343713	2.67201	3.076923077	7.87%	6.66%	10.71%	0.441730367	0.44008593	0.668002506
6	2000	103.44	10.34554	0.575738	29.5414	2.5442	0.12544	3.501526671	4.066323402	4.5897481	4	9.65%	8.20%	12.69%	0.583587779	0.67772057	0.764958014
8	2000	103.44	10.34554	0.575738	16.9544	1.955	0.08541	6.101071108	5.291836317	6.7408734	4.705882353	11.52%	9.79%	14.66%	0.762633889	0.66147954	0.842609179
12	2000	103.44	10.34554	0.575738	11.2211	1.1525	0.05684	9.218347577	8.976607375	10.129099	5.714285714	13.78%	11.71%	16.41%	0.768195631	0.74805061	0.844091602

Figure 15: Strong Scalability – Light Cluster – Intra Regional



Weak Scalability Light cluster

NP	Matrix Size	S mult(D)	S mult(S)	S inv	P mult(D)	P mult(S)	P inv	Sp (MultD)	Sp (MultS)	Sp (Inv)	Theoretical Sp	C % (MultD)	C % (MultS)	C % (Inv)	E (MultD)	E (MultS)	E (Inv)
2	1000	20.5545	3.2205	0.39951	12.0301	2.215	0.2974	1.708589289	1.453950339	1.3433423	1.818181818	4.25%	3.85%	6.50%	0.854294644	0.72697517	0.67167115
4	2000	150.256	15.552	0.575738	58.5425	9.2211	0.3896	2.566613998	1.686566679	1.4777669	3.076923077	5.12%	4.54%	7.01%	0.6416535	0.42164167	0.369441735
6	3000	349.14	33.155	0.65115	104.7254	15.5442	0.4873	3.333861699	2.132949911	1.3362405	4	7.21%	6.21%	8.35%	0.555643617	0.35549165	0.222706751
8	4000	816.136	79.1552	0.71021	192.8024	32.1558	0.5841	4.233017846	2.461615012	1.2159048	4.705882353	8.45%	7.47%	9.77%	0.529127231	0.30770188	0.151988101
12	6000	2753.056	270.251	0.855151	423.8896	61.1541	0.7551	6.494747689	4.419180398	1.1325003	5.714285714	10.51%	8.96%	11.25%	0.541228974	0.36826503	0.094375028

Figure 16: Weak Scalability – Light Cluster – Intra Regional