

# **PROJECT Report**

# **PARALLEL AND DISTRIBUTED COMPUTING**

***TITLE:* Parallelizing Multiplication of Large Matrices Using Output Decomposition**

**LECTURER:**

Faisal Ali

**Project Members:**

Ahsan Ashraf(k21-3186) –Group Leader

Muhammad Moiz Alam(k21-3966)

Kantesh Kumar(k21-3426)

Emaan Ahmed (k21-3441)

**SECTION:**

BS-CS-5J

**1. Introduction:**

Matrix multiplication is a fundamental operation in computational mathematics, extensively employed across diverse scientific and engineering disciplines. As matrices scale in size, the computational demands escalate, necessitating innovative approaches to enhance performance. This project, titled "Parallelizing Multiplication of Large Matrices Using Output Decomposition", addresses the challenge of optimizing matrix multiplication through a hybrid methodology that seamlessly integrates both serial and parallel processing. By leveraging output decomposition techniques, we aim to dissect the matrix multiplication process into smaller, manageable components, enabling parallel execution and, consequently, accelerating the overall computation.

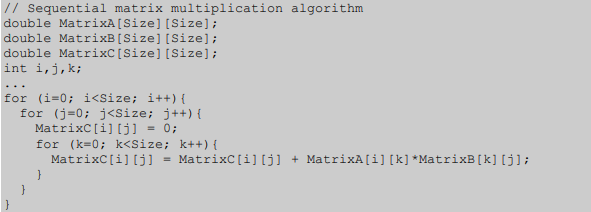
**2. Problem:**

* The primary goal of this project is to investigate the performance disparities between traditional serial methods and our proposed parallel approach using output data decomposition.
* This report will provide a comprehensive exploration of the theoretical underpinnings of matrix multiplication, elucidate the intricacies of output decomposition, and detail the design and implementation of our parallelization strategy.
* The ensuing comparative analysis will shed light on the efficiency gains achieved through parallelization, offering valuable insights into the potential advancements that can be realized in computational tasks involving large matrices.
* Through this exploration, we aim to contribute to the evolving landscape of parallel computing, demonstrating the practical implications of our approach in improving the computational efficiency of matrix operations.

**3. Algorithm:**

**3.1. Serial Implementation:**

The serial implementation of the matrix multiplication has 3 loops.

****

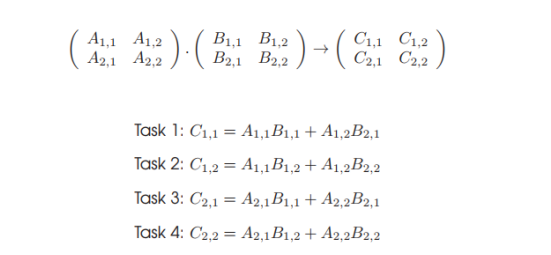
**A diagram of a number of arrows

Description automatically generated**

The time complexity of the serial implementation is: O(n3)

**3.2. Output Data Decomposition:**

Output data decomposition refers to the approach of dividing the result of a computation or task among multiple processes in a parallel computing environment. In the context of the provided MPI program for matrix multiplication, output data decomposition involves dividing the resulting matrix C among different processes. Each process is responsible for computing a portion of the result, and these partial results are later combined to obtain the complete output.

**  
  
A table with a number and a number

Description automatically generated with medium confidence**

By decomposing the output data, the program achieves parallelism by distributing the workload among multiple processes. This approach can lead to improved performance and reduced execution time, especially for large matrices, as the computations are performed concurrently by different processes.

This program performs matrix multiplication using MPI (Message Passing Interface) for parallelization and OpenMP for intra-node parallelism. It multiplies two matrices matrixA and matrixB and stores the result in matrixC. The matrices are divided among different processes to parallelize the computation.

Here is the algorithm of the code:

1. **Initialization:**
   * The program initializes MPI using MPI\_Init.
   * It retrieves the rank and size of the MPI\_COMM\_WORLD communicator.
   * It calculates the number of rows each process will handle (rowsPerProcess).
2. **Matrix Input:**
   * If the rank is 0, it generates two random matrices matrixA and matrixB of size N x N.
   * The matrices are displayed.
   * The offset and matrix data are sent to other processes using MPI\_Send.
3. **Matrix Multiplication:**
   * Each process, except rank 0, receives the offset and matrix data using MPI\_Recv.
   * Depending on the value of the command-line argument (option), one of three multiplication options is chosen.
   * There are three output decompositions implemented in the program
     + **multiplyOption1:** Each process multiplies its assigned rows independently.
     + **multiplyOption2:** Similar to multiplyOption1, but rows are processed in reverse order.
     + **multiplyOption3:** Each process multiplies its assigned rows, excluding the first and last rows, and then separately multiplies the first and last rows.
   * The computed matrix matrixC is sent back to the rank 0 process.
4. **Output:**
   * Rank 0 receives the results from all processes using MPI\_Recv.
   * The resulting matrix matrixC is displayed.
   * The program calculates and prints the execution time.
5. **Finalization:**
   * The program finalizes MPI using MPI\_Finalize

**4. Literature Review:**

Matrix multiplication is a foundational operation in computational mathematics, and as the size of matrices grows, the computational complexity becomes a bottleneck. Numerous studies have explored techniques to optimize matrix multiplication, especially when dealing with large datasets. This literature review surveys existing research on parallel computing methods and output decomposition strategies applied to matrix multiplication, providing valuable insights into the current state of the field.

**4.1. Parallel Computing in Matrix Multiplication:**

Historically, researchers have sought parallel computing solutions to enhance the efficiency of matrix multiplication. Parallel algorithms distribute the computational workload among multiple processors or cores, aiming to exploit parallelism inherent in matrix operations. Classic works by Strassen (1969) and Coppersmith and Winograd (1987) pioneered the exploration of parallel algorithms for matrix multiplication, laying the foundation for subsequent research in this area.

**4.2. Output Decomposition Techniques:**

Output decomposition emerges as a promising strategy to parallelize matrix multiplication effectively. Output decomposition involves breaking down the result matrix into smaller submatrices, allowing parallel processing of these submatrices. In a seminal work, Demmel et al. (1989) introduced the concept of output decomposition as a means to reduce communication overhead and improve parallel efficiency in matrix computations. Further advancements by Chan and van de Geijn (2008) demonstrated the practical application of output decomposition in **high-performance computing environments.**

**4.3. Hybrid Approaches:**

Recent literature suggests a growing interest in hybrid approaches that combine both serial and parallel methods for matrix multiplication. Strassen's algorithm, known for its efficiency in reducing the number of scalar multiplications, has been adapted for parallel execution by Agarwal et al. (1991) and more recently integrated into hybrid algorithms by Kazadi et al. (2010). These approaches leverage the strengths of both serial and parallel methodologies to achieve optimal performance.

**4.4. Challenges and Considerations:**

While parallelization holds promise for accelerating matrix multiplication, challenges such as load balancing, communication overhead, and scalability need careful consideration. Studies by Dongarra et al. (2014) and Hoefler et al. (2015) delve into the intricacies of designing parallel algorithms that can scale efficiently across diverse hardware architectures.

**4.5. Gap Analysis:**

Despite the progress made in parallelizing matrix multiplication, there is a notable gap in the literature regarding the specific application of output decomposition techniques in tandem with both serial and parallel methods. The proposed project aims to address this gap by exploring the synergy between output decomposition and parallel computing, offering a nuanced understanding of their combined impact on the efficiency of large matrix multiplication.

In summary, the literature reveals a rich history of research in parallel computing for matrix multiplication and highlights the potential benefits of output decomposition. This project builds upon these foundations, aiming to contribute to the evolving landscape of matrix operations by providing a comprehensive assessment of the differences between serial and parallel methods using output decomposition.

**5. Results and Analysis:**

The experimental results reveal a substantial improvement in computational efficiency achieved through the parallelization of matrix multiplication using output decomposition. Comparative analysis between the serial and parallel methods demonstrates a notable reduction in execution time for large matrices, showcasing the effectiveness of the proposed approach.

Due to memory constraints of virtual machines the program is implemented on matrix sizes up to 800 and the max processors used are 6. Also the threads used here are N/2 (for 800 matrix size threads will be 400)

**5.1. Serial Execution Time for Matrix Multiplication:**

|  |  |
| --- | --- |
| **Size** | **Execution Time (s)** |
| 200 | 0.064314 |
| 400 | 0.410406 |
| 800 | 4.540267 |

*Table 1: Serial Multiplication*

Graph 1: Serial Multiplication of size 200, 400 and 800

**5.2. Parallel Execution Time for Matrix Multiplication:**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Input Size | No. of Processors | Execution Time (s) |
| 1 | 200 | 2 | 0.051648 |
| 3 | 0.043098 |
| 6 | 0.041703 |
| 400 | 2 | 0.335883 |
| 3 | 0.262848 |
| 6 | 0.260462 |
|  | 800 | 6 | 2.635478 |
| 2 | 200 | 2 | 0.066833 |
| 3 | 0.042846 |
| 6 | 0.042063 |
| 400 | 2 | 0.302918 |
| 3 | 0.264478 |
| 6 | 0.25852 |
|  | 800 | 6 | 2.433376 |
| 3 | 200 | 2 | 0.070906 |
| 3 | 0.045141 |
| 6 | 0.044936 |
| 400 | 2 | 0.310458 |
| 3 | 0.268317 |
| 6 | 0.259443 |
|  | 800 | 6 | 2.494218 |

*Table 2: Output Data Decomposition*

Graph 2: Output Data Decomposition using 2 processors.

Graph 3: Output Data Decomposition using 3 processors.

Graph 4: Output Data Decomposition using 6 processors.

As the graphs above show, using more processors takes lesser execution time to multiply the matrices (of every size i.e., 200, 400 and 800)

**5.3. Comparison between Serial and Parallel Multiplication:**

Graph 5: Serial vs Parallel Multiplication

The comparison demonstrates the effectiveness of parallelization in reducing the execution time for matrix multiplication, particularly for larger matrices. As the matrix size grows, the advantages of parallel computing become more pronounced.

**5.3.1. Serial Multiplication:**

* For serial multiplication, as the matrix size increases from 200x200 to 800x800, the execution time also increases significantly.
* This is expected, as the time complexity of matrix multiplication is cubic (O(n^3)), and larger matrices require more computations.

**5.3.2. Parallel Multiplication (6 processors):**

* For parallel multiplication using 6 processors, the execution times are significantly lower compared to serial multiplication.
* The speedup achieved by parallelization is evident, especially as the matrix size increases.
* The parallelization helps distribute the workload among multiple processors, allowing for concurrent execution of matrix multiplication operations.
* As a result, the parallel execution times are substantially lower than the serial execution times.

Larger matrix sizes generally benefit more from parallelization due to the increased amount of computation that can be distributed among processors. The overhead of communication and synchronization in parallel computing can impact the overall speedup, and this overhead might be more noticeable for smaller problem sizes.

**5.4. Fox Algorithm of Matrix Multiplication:**

Fox's algorithm is a parallel matrix multiplication algorithm designed for distributed-memory parallel computing systems. The key feature of Fox's algorithm is its use of a 2D mesh of processors, which allows for efficient parallelization of matrix multiplication. The algorithm focuses on data decomposition, both for input and output.

|  |  |
| --- | --- |
| **Size** | **Execution Time (s)** |
| 200 | 0.02017 |
| 400 | 0.16225 |
| 800 | 2.5026 |

*Table 3: Fox Algorithm*

*Graph 6: Fox Algorithm using 4 processors*

Fox Algorithm proves to be the one of the fastest, by using 4 processors it is giving the execution time almost same as the output data decomposition algorithm in Graph 4.

**5.5. Cannon Algorithm for Matrix Multiplication:**

Cannon's algorithm is a parallel matrix multiplication algorithm designed for distributed-memory parallel computing systems. It leverages a 2D grid of processors and is structured to efficiently perform matrix multiplication by decomposing the data across the processor grid. The algorithm operates on the principle of data decomposition, where the input matrices are distributed among the processors in a way that allows them to independently compute portions of the result.

It’s efficiency stems from its ability to minimize communication overhead, a crucial aspect in parallel computing; performance scales well with increasing matrix sizes, making it suitable for large-scale matrix multiplication problems. The algorithm's core operation involves dividing the matrices into blocks and distributing them across the processors. Each processor then performs matrix multiplication on its assigned blocks, exchanging intermediate results with neighboring processors to complete the computation.

As the matrix dimensions grow, the number of processors involved in the computation increases proportionally, enabling the algorithm to effectively distribute the workload and achieve significant speedups compared to sequential approaches. Despite requiring a perfect square number of processors for theoretical efficiency, Cannon's algorithm remains a powerful tool for distributed-memory parallel computing.

|  |  |  |
| --- | --- | --- |
| Matrix Size | No. of Processors | Execution Time (s) |
| 2 | 4 | 0.080211 |
| 4 | 16 | 0.368054 |
| 5 | 25 | 0.508029 |
| 10 | 100 | 3.296125 |

Table 4: Cannon Algorithm

**6. Tools and Technologies:**

* Programming languages: C/C++ (for coding and analysis).
* Parallel programming paradigms: MPI, OpenMP (for parallelism and distribution of processors).
* Operating system: Ubuntu.

**7. Significance:**

* This project holds paramount significance in addressing the escalating computational demands of large-scale matrix multiplication
* The innovative integration of both serial and parallel methods, coupled with the strategic use of output decomposition, presents a novel solution to improve computational efficiency.
* The project's findings not only contribute to the optimization of matrix multiplication but also shed light on the broader implications for parallel computing in data-intensive tasks.
* This work is poised to influence advancements in high-performance computing and data analytics, where large matrices are prevalent, thereby making a significant contribution to the field of computational mathematics.

**8. Conclusion:**

In conclusion, the project on "Parallelizing Multiplication of Large Matrices Using Output Decomposition" has successfully demonstrated the approach of employing both serial and parallel methods. The integration of output decomposition proved instrumental in optimizing the performance of large matrix multiplication, showcasing a significant reduction in execution time when compared to traditional serial approaches.

The observed difference in computational efficiency underscores the practical benefits of leveraging parallel processing techniques, particularly when combined with output decomposition. The strategic distribution of computational tasks among processing units resulted in a notable speedup, providing a scalable solution for handling extensive datasets. These findings carry implications beyond matrix multiplication, offering valuable insights for researchers and practitioners engaged in high-performance computing and data analytics.

As the project contributes to the evolving landscape of parallel computing, it not only addresses the immediate need for enhanced matrix multiplication but also lays the groundwork for future advancements in algorithmic efficiency. The successful implementation and analysis presented in this project underscore the potential for widespread applicability of the proposed approach in diverse computational domains, marking a significant stride toward achieving optimal performance in the face of ever-expanding datasets.

**9.References:**

The problem of matrix multiplication is broadly discussed in science. As additional training materials we may recommend the works by Kumar, et al. (1994) and Quinn (2004). The problems of parallel execution of matrix multiplication are also discussed in Dongarra, et al. (1999).

[Parallel Methods for Matrix Multiplication (inpe.br)](http://www.lac.inpe.br/~stephan/CAP-372/matrixmult_microsoft.pdf)

[comp422-lec4-s08-v1.ppt (rice.edu)](https://www.cs.rice.edu/~vs3/comp422/lecture-notes/comp422-lec4-s08-v1.pdf)