Design and Analysis of Parallel Matrix Multiplication

Ei Ei Myint¹, Lai Lai Win Kyi²
¹Department of Information Technology
Mandalay Technological University, Mandalay, Myanmar
¹eikhaingsoe20@gmail.com, ²laeae83@gmail.com

Abstract—Problem in science and engineering are often large, complex and highly irregular. Many type of problem can be converted into matrix form. Multiplication of large matrices requires a lot of time as its complexity is O(n³). Because most current applications require higher computational throughputs with minimum time, parallel algorithm is developed. If matrix multiplication is done in the efficient way, many problems can be solved easily and efficiently. As a result, this paper tends to implement parallel matrix multiplication in Java threads. The application distributes the products of rows and columns on different servers. One server and four clients are run to find the product of matrix multiplication. The server distributes the determine blocks of rows and columns on the registered clients. The clients return their product blocks to a server, which calculate the final product of matrix multiplication. In this paper, a theoretical analysis for the performance and evaluation of parallel matrix multiplication algorithms is carried out. And an experimental analysis is performed to support the theoretical analysis results by calculating 100×100 to 500×500 matrices.

Keywords: Parallel Matrix Multiplication, Java threads

I. INTRODUCTION

Matrix Multiplication is used as building block in many of applications covering nearly all subject areas. Like physics makes use of matrices in various domains, for example in geometrical optics and matrix mechanics; the latter led to studying in more detail matrices with an infinite number of rows and columns. Graph theory uses matrices to keep track of distances between pairs of vertices in a graph. Computer graphics uses matrices to project 3-dimensional space onto a 2-dimensional screen. Matrix calculus generalizes classical analytical concept such as derivatives of functions or exponentials to matrices etc [1].

According to the above reasons, matrix multiplication is commonly used in many areas. Multiplication of large matrices requires a lot of computation time as its complexity is O(n³), where n is the dimension of the matrix. Because most current applications require higher computational throughputs, many algorithms based on sequential and parallel approaches were developed to improve the performance of matrix multiplication [2].

The parallel processing is the processing of program instructions by dividing them among multiple processors with the objective of running a program in less time. In serial processing, next instruction will not be executed until the previous instruction completed. Parallel processing demands on concurrent execution of many programs in computer. The highest level of parallel processing is conducted among multiple programs through multi-programming, time sharing and multiprocessing. This level requires development of the parallel process able algorithms. The implementation of parallel algorithms depends on the efficient allocation of limited hardware-software resources to multiple programs being used to solve large manipulation problems [6].

The research tends to implement parallel matrix multiplication with the efficient way and analysis performance comparison of execution time for the sequential and parallel matrix multiplication. The rest of this paper is organized as follows. Section II presents the related work for this research. Section III presents matrix multiplication techniques. It describes sequential algorithm and parallel algorithms. Section IV describes design and implementation of the proposed system. Section V closes the paper with conclusion.

II. RELATED WORKS

M Ismail proposed “concurrent matrix multiplication on multi-Core processors”. In this paper, the matrix multiplication algorithm is presented using a newly developed parallel programming model SPC³ PM(Serial, Parallel and Concurrent Core to Core Programming Model) for general purpose multi-core processors. With the concurrent function of SPC3 PM, the programmer can execute a simple and standard matrix multiplication algorithm concurrently on multi-core processors in much less time than that of standard parallel OpenMP. The proposed approach also shows more scalability, better and uniform speedup and better utilization of available cores than that of OpenMP. This SPC3 PM will be further worked out for the introduction of some more parallel and concurrent functions and synchronizing tools [1].

Z. Aliqadi, discussed “Performance Analysis and Evaluation of Parallel Matrix Multiplication Algorithms”. In this paper, a theoretical analysis for the performance and evaluation of the parallel matrix multiplication algorithms is carried out. Moreover, an experimental analysis is performed to support the theoretical analysis results. Recommendations are made based on this analysis to select the proper parallel multiplication algorithms. This analysis has shown that systolic algorithm is considered the best algorithm that produced a high efficiency and then followed by puma, dimma and then suma [2].

Merima Delić and his colleagues performed the analysis of parallel matrix multiplication using C# threading. Threaded computation was distributed to all available cores in the shared memory system. Number of threads created to parallelize for-loops is equal to the number of cores in the system. The main aim of this work was to accomplish parallelization similar to what Microsoft .NET V4.o provided in their parallel extensions library and make it easy to use [3].

Sandip. K.Bhagat proposed parallel algorithms for matrix multiplication. This paper describes the various algorithms of Matrix Multiplication that can be implemented on parallel processing environment. Which includes Strassen’s Matrix Multiplication which can be done in O(n².₈₁) using sequential
computing. SUMMA algorithm works in \(O(n^2)\) in the parallel environment. And Matrix Multiplication can be done on SIMD computer in \(O(n \log n)\) times. It is concluded that Strassen’s algorithm gives better performance in sequential computing. The SUMMA algorithm and SIMD matrix multiplication can be only implemented in parallel environment [6].

If we do matrix multiplication in the efficiently way many problems can be solved easily and efficiently. Above of all researches, parallel method is faster than sequential method. As these results, this paper tends to implement parallel matrix multiplication in an efficient way using java RMI threads.

III. DESCRIPTION OF THE PROPOSED ALGORITHM

This section presents four portions. Firstly, the operation of matrix multiplication is described. And sequential method and parallel method in SIMD for matrix multiplication are discussed. Lastly Performance Evaluation is mentioned.

A. Matrix Operation

Consider the product of matrix multiplication

\[
C = A \times B
\]

where \(A, B, C\) are matrices of size \(n \times n\) as are as follows:

\[
a_{ij} \quad n \text{ columns, } j\text{ changes} \\
n \times m \quad n \text{ rows, } i\text{ changes}
\]

\[
A \times B = C \\
(m \times n) \times (n \times p) = (m \times p)
\]

For notation purpose it has been referred has m-by-n matrix as “(m x n)”. Similarly, the multiplication of an m-by-n matrix with n-by-p matrix is denoted as “(m x n) \times (n x p)”. By Initial assumption, \((n \times kn) \times (kn \times n)\) Matrix Multiply would require, at minimum, \(k\)-times the duration of an \((n \times n) \times (n \times n)\) Matrix Multiply. Based on assumption, the notation is given as:

\[
[A_11] \times [B_11] = [A_11 \times B_11]
\]

Whereas:

\[
[A_{i1} \ldots A_{i1}] \times [B_{11}] = [A_{i1} \times B_{11} + A_{i2} \times B_{21} + \ldots + A_{ii}]
\]

\[
[B_{12}] \\
[\ldots] \\
[B_{1n}]
\]

there would ultimately be \(k\)-times the number of \(A_{1x1}B_{1x1}\) operations (as well as \(k\)-times the number of addition operations, assuming that the resulting matrix had its element values initialized to the value of zero) involved in an \((n \times kn) \times (kn \times n)\) Matrix Multiply, versus an \((n \times n) \times (n \times n)\) Matrix Multiply, based on the example given in the above [5].

B. Sequential Method

Suppose we want to multiply two \(n \times n\) matrices \(A\) and \(B\), resulting in matrix \(C\). The standard sequential algorithm (without cache blocking) is usually written as three nested loops on \(i\) (row number in \(C\) and \(A\)), then \(j\) (column number in \(C\) and \(B\)), and then \(k\) (inner index, column of \(A\) and row of \(B\)). However, we can nest the loops in an order and still get the same answer. Using the order \(ijk\) instead of \(ij\), we get the following code:

**Algorithm**:

```
C = zeros(n,n);
for k = 1:n
    for i = 1:n
        for j = 1:n
            C(i,j) = C(i,j) + A(i,k) * B(k,j);
        end
    end
end
```

In serial processing, the \(n^3\) cumulative multiplications are carried out by a serially coded program with three levels of DO loops corresponding to three indices to be used. The time complexity of this sequential program is proportional to \(n^8\) [6].

C. Parallel method

To implement the matrix multiplication in parallel the matrix \(A\) is decomposed into several multiple rows depending on the number of processors by using the following equation.

\[
\text{No. of rows} = \frac{\text{Size of the matrix}}{\text{Number of the processors}}
\]

![Figure 1: Operation of Parallel Matrix Multiplication](http://ijccer.org)

The operation of parallel matrix multiplication is illustrated in figure 1. Under this operation, the following steps are performed.

- Step 1. Master PC reads data from user.
- Step 2. Server decomposes the matrix \(A\) into multiple rows.
- Step 3. Server sends the partitioned matrix \(A\) and matrix \(B\) to slaves.
- Step 4. Every process performs its local multiplication.
- Step 5. All slave processes send back their result.
- Step 7. Master PC display the results [4].

D. Performance Evaluation

Table 1 and 2 analyze the number of operation in sequential and parallel matrix multiplication. Here, \(n \times n\) matrix are multiplied for 2, 4 parallel / concurrent threads respectively.
### Table 1: Theoretical analysis of sequential matrix multiplication

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Multiplication</th>
<th>Addition</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2×2</td>
<td>8</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>4×4</td>
<td>64</td>
<td>48</td>
<td>112</td>
</tr>
<tr>
<td>8×8</td>
<td>512</td>
<td>448</td>
<td>960</td>
</tr>
<tr>
<td>16×16</td>
<td>4096</td>
<td>3840</td>
<td>7936</td>
</tr>
<tr>
<td>32×32</td>
<td>32768</td>
<td>31744</td>
<td>64512</td>
</tr>
<tr>
<td>64×64</td>
<td>262144</td>
<td>258048</td>
<td>520192</td>
</tr>
</tbody>
</table>

### Table 2: Theoretical analysis of parallel matrix multiplication

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>no. of processors</th>
<th>Multiplication</th>
<th>Addition</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2×2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4×4</td>
<td>2</td>
<td>32</td>
<td>24</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>16</td>
<td>12</td>
<td>28</td>
</tr>
<tr>
<td>8×8</td>
<td>2</td>
<td>256</td>
<td>224</td>
<td>480</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>128</td>
<td>112</td>
<td>240</td>
</tr>
<tr>
<td>16×16</td>
<td>2</td>
<td>2048</td>
<td>1920</td>
<td>3968</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1024</td>
<td>960</td>
<td>1984</td>
</tr>
<tr>
<td>32×32</td>
<td>2</td>
<td>16384</td>
<td>15872</td>
<td>32256</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8192</td>
<td>7936</td>
<td>16128</td>
</tr>
<tr>
<td>64×64</td>
<td>2</td>
<td>131072</td>
<td>129024</td>
<td>260096</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>56636</td>
<td>64512</td>
<td>130048</td>
</tr>
</tbody>
</table>

According to the above tables, the more the number of processors increases, the less the number of operations will be.

## IV. DESIGN AND IMPLEMENTATION OF THE PROPOSED SYSTEM

This section presents the design and implementation of sequential and parallel matrix multiplication system. And the performance analysis of these two algorithms is discussed.

### A. Design of the Proposed System

The proposed system is illustrated in figure 2. There are two portions: sequential side and parallel side.

![Figure 2: Sequential and Parallel Matrix Multiplication](image)

Firstly Server generates two \((n \times m)\) matrices by using random generators. In sequential side, Server calculates the matrix multiplication and display the results and store the execution time. On the other hand, in parallel, Server will distribute number of rows from first matrix and its corresponding columns of the second matrix on clients and calculates concurrently matrix multiplication using Java threads. Server will waits results from clients and append it in result matrix. And display the results and store the execution time. Finally this system compares the two execution times of the sequential and parallel matrix multiplication.

### B. Implementation of the proposed system

In this section, the sequential and parallel matrix multiplication algorithms are implemented with the help of Java Programming Language.

The home page for calculating sequential and parallel method is illustrated in figure 3. Under this form, the user can select the desired matrix multiplication method whether sequential or parallel. And the desired matrices sizes are defined in textboxes. Here, \(250\times250\) matrices are calculated as an example under sequential method.
Figure 4 illustrated the multiplication of $250 \times 250$ matrices under the parallel consideration. In parallel computing, four servers are used to solve the problem. These servers receive the partitioned matrix $A$ and matrix $B$ from main frame and multiply these matrices. This is shown in figure 5, figure 6, figure 7 and figure 8 respectively. Finally the result will be displayed in the main form as shown in figure 9.
Figure 9: Results of Parallel Matrix Multiplication

The view graph button in main form tends to show the performance comparison of sequential and parallel matrix multiplication. It is illustrated in figure 10.

Figure 10: Performance Comparison of Sequential and Parallel Matrix Multiplication

Table 3 lists the execution time of sequential and parallel matrix multiplication for 100 × 100, 200 × 200, 300 × 300, 400 × 400 and 500 × 500.

Table 3: Execution times of sequential and parallel Matrix Multiplication

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Sequential execution time (ms)</th>
<th>Parallel execution time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100×100</td>
<td>19.1</td>
<td>2.2</td>
</tr>
<tr>
<td>200×200</td>
<td>48.1</td>
<td>16.9</td>
</tr>
<tr>
<td>300×300</td>
<td>160.8</td>
<td>85.1</td>
</tr>
<tr>
<td>400×400</td>
<td>436.7</td>
<td>179</td>
</tr>
<tr>
<td>500×500</td>
<td>787.1</td>
<td>270</td>
</tr>
</tbody>
</table>

The performance comparison of sequential and parallel matrix multiplication is illustrated in Figure 11. According to the graph, the execution time in parallel is over two times faster than in sequential.

V. CONCLUSION

A theoretical analysis for the performance of sequential and parallel algorithms is carried out using one and four processors. This analysis has shown that parallel algorithm has high efficiency than sequential algorithm. The experimental results are implemented through this theoretical analysis with the help of java programming language. It has been shown that execution time decreases over two times for matrix multiplication of 100x100, 200x200, 300x00, 400x400 and 500x500 matrix sizes with four servers. Future work will apply this implementation on any practical application like weather prediction, database systems, data compression and others, with increasing the number of clients.

ACKNOWLEDGMENT

First of all, the author is highly grateful to Dr. Myint Thein, the Pro-Rector of the Mandalay Technological University for his permission for completion of this paper. The author wants to express her gratitude to Dr. Lai Lai Win Kyi for her help and advice regarding of this topic and excellent guidance, valuable suggestions and advices. The author is deeply thankful to all teachers from Department of Information Technology at Mandalay Technological University for their overall supporting during the writing of this paper.

REFERENCES

[3] Merima Delič, Edina Omerović, Azemina Sokolić Novica Nosović Faculty of Electrical Engineering, University of Sarajevo Sarajevo, Bosnia and Herzegovina “Performance Analysis of the Parallelized Matrix Multiplication Using C# Multiplication on Grid Enabled PC Cluster”.
[5] IME-Computer Science and Engineering (Final Year) , Annamalai University Chidambaram, India-608002. “Parallel Matrix Multiplication Implementation In Distributed Environment Through RMI”.

Figure 11: Performance Comparison of Sequential and Parallel Matrix Multiplication