e-ISSN: 2349-9745
p-ISSN: 2393-8161
Scientific Journal Impact Factor (SJIF): 1.711
International Journal of Modern Trends in Engineering and Research
www.ijmter.com

# Testing of Matrices Multiplication Methods on Different Processors

Jaymit Pandya[1], Jay Vala[2], Chetan Chudasama[3], Dhara Monaka[4]
[1]*Assistant Professor, IT Dept, GCET Engg College, jaymitpandya@gcet.ac.in*
[2]*Assistant Professor, IT Dept, GCET Engg College, jayvala@gcet.ac.in*
[3]*Assistant Professor, CE Dept, MBICT, chetanfriends30@gmail.com*
[4]*Assistant Professor, BCA Dept, NMC*

**Abstract –** There are many algorithms we found for matrices multiplication. Until now it has been found that complexity of matrix multiplication is $O(n^3)$. Though Further research found that this complexity can be decreased. This paper focus on the algorithm and its complexity of matrices multiplication methods.

**Keywords-** Matrices, multiplication, strassen, coppersmith and winograd

## I. INTRODUCTION

The product of two matrices is one of the most basic operations in mathematics and computer science. Here, we foucs on mainly three methods of matrices multiplication. 1) Conventional Method, 2)Strassen's Mehtod and 3)Coppersmith and Winograd.

Until 1969, it was thought that complexity of matrices multiplication is of order $n^3$ though some of researchers was of opinion that it is of order $n^2$ . Following Table 1 shows Reducing complexities found as time passed.

Table 1. Complexity of Various Algorithms for Matrices Multiplication[9,10,11]

| Name of Method | Complexity |
|---|---|
| Conventional Method | $n^3$ |
| Strassen | $n^{2.808}$ |
| Pan | $\omega < 2.796$ |
| Bini | $\omega < 2.78$ |
| Schonhage | $\omega < 2.548$ |
| Romani | $\omega < 2.517$ |
| Coppersmith & Winograd | $\omega < 2.496$ |
| Strassen's New Method | $\omega < 2.479$ |
| Coppersmith & Winograd | $\omega < 2.376$ |

From the above table it can be seen that tighter lower bound decreases. It can also be seen that it was only up gradation of Strassen's Method and Coppersmith & Winograd. In this paper, we have considered only three methods. They are considered in a view that which methods gives better result when general matrices multiplication is done. Of course, it can be said that competitors are only last two methods but for providing base Conventional method is considered[12,14].

## II. MATRICES MULTIPLICAITON METHODS

### 1. Conventional Method of Matrix Multiplication

If A is an n × m matrix and B is an m × p matrix,

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

Multiplication can be given by following formula[13].

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} .$$

Calculation of complexity is as follows.

Here is an algorithm for doing this, assuming that a, b, and c have been declared as square two-dimensional arrays of integers, and n is the length of a row or column in the arrays[13]:

```
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        c[i][j]= 0;
        for (int k = 0; k < n; k++)
        {
        c[i][j] += a[i][k] * b[k][j];
        }
     }
}
```

Let's analyze the number of operations performed by this algorithm in terms of the matrix size n, and use that analysis to come up with a big-O expression for the time complexity of the algorithm as a whole[13].

First, recognize that since we have triply-nested for loops, each of which individually loops n times, the total number of iterations of the outermost loop will be n, the total number of iterations of the middle loop will be n * n = $n^2$, and the total number of iterations of the innermost loop will be n * n * n = $n^3$.

**Multiplication**
The only multiplication appears in the innermost loop. Since this loop will iterate a total of n3 times, we will perform exactly $n^3$ multiplication operations.

**Addition**
There are two addition operations in the innermost loop: c[i][j] += a[i][k] * b[k][j] and k++. Each of these will happen $n^3$ times, giving us $2n^3$ additions generated by the innermost loop. The middle loop

 

has one addition operation (j++) which happens $n^2$ times, since the middle loop iterates a total of $n^2$ times.

The outermost loop has one addition operation as well (i++) which happens n times, since the outermost loop iterates a total of n times.

Adding these up, we get an exact value for the number of addition operations:$2n^3+ n2+ n$

**Assignment**
The += operation in the innermost loop happens n3 times, since it's inside the loop. The k = 0 assignment happens only once for each complete execution of the innermost loop, so it executes only n2 times. Similarly, the c[i][j] = 0 assignment, being within the middle loop, happens $n^2$ times. The j = 0 assignment happens only once for each complete execution of the middle loop, so it executes n times. The i = 0 assignment happens only once, when the outermost loop first starts running. Adding these up gives us an exact expression for the number of assignments:$n^3+ 2n^2+ n + 1$

Now, using the exact expressions above, let's come up with big-O expressions for the time complexity of each kind of operation as a function of n. There are exactly $n^3$ multiplications, so there are $O(n^3)$ multiplications.

There are exactly $2n^3+ n^2+ n$ additions, so, dropping all but the largest term and discarding its coefficient, there are $O(n^3)$ additions.

There are exactly $n^3+ 2n^2+ n + 1$ assignments, so, dropping all but the largest term and discarding its coefficient, there are $O(n^3)$ assignments.

$O(n^3)$ multiplications + $O(n^3)$ additions + $O(n^3)$assignments = $O(n^3)$ overall time complexity[13].

## 2. Strassen's Method for Matrix Multiplication

Strassen's Algorithm for matrix multiplication is seen as improvement over conventional method. Assuming that we have following matrices.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{12} & C_{22} \end{bmatrix}$$

&

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Calculations for resulting matrix can be done as follows.

 

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$
$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$
$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$
$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$

Based on this, we can find out values for final matrix as follows.

$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

**Complexity**

It is well known that complexity of Strassen's method is of order 2.808, which is improvement over conventional method. Even Second Version of Strassen's method gave lower complexity of order 2.479[2,3,4]. This method is for 2*2 matrices. If matrices are of higher size than they are divided to 2*2 matrices using block method and then result is computed.

## 3. Coppersmith & Winograd Method for Matrix Multiplication

The original algorithm is divided into two parts. Easy Version and Complicated Version. For the sake of complete ness here we have shown snapshots of both.

Following Figure 1 Shows Original Easy version of coppersmith and winograd algorithm.

$$\sum_{i=1}^{q} \lambda^{-2}\left(x_0^{[0]} + \lambda x_i^{[1]}\right)\left(y_0^{[0]} + \lambda y_i^{[1]}\right)\left(z_0^{[0]} + \lambda z_i^{[1]}\right)$$
$$- \lambda^{-3}\left(x_0^{[0]} + \lambda^2 \sum x_i^{[1]}\right)\left(y_0^{[0]} + \lambda^2 \sum y_i^{[1]}\right)\left(z_0^{[0]} + \lambda^2 \sum z_i^{[1]}\right)$$
$$+ \left(\lambda^{-3} - q\lambda^{-2}\right)\left(x_0^{[0]}\right)\left(y_0^{[0]}\right)\left(z_0^{[0]}\right) \qquad (5)$$
$$= \sum_{i=1}^{q}\left(x_0^{[0]} y_i^{[1]} z_i^{[1]} + x_i^{[1]} y_0^{[0]} z_i^{[1]} + x_i^{[1]} y_i^{[1]} z_0^{[0]}\right) + O(\lambda).$$

We have brought the factors $\lambda^{-3}$, $\left(\lambda^{-3} - q\lambda^{-2}\right)$ outside in order to reflect the symmetry.

**Note.** This is equivalent to the *bilinear* algorithm

$$M_i = \left(x_0^{[0]} + \lambda x_i^{[1]}\right)\left(y_0^{[0]} + \lambda y_i^{[1]}\right), \qquad i = 1, 2, \ldots, q$$
$$M_{q+1} = \left(x_0^{[0]} + \lambda^2 \sum x_i^{[1]}\right)\left(y_0^{[0]} + \lambda^2 \sum y_i^{[1]}\right)$$
$$M_{q+2} = \left(x_0^{[0]}\right)\left(y_0^{[0]}\right)$$
$$v_i^{[1]} \equiv x_0^{[0]} y_i^{[1]} + x_i^{[1]} y_0^{[0]} = \lambda^{-1} M_i - \lambda^{-1} M_{q+1} + O(\lambda), \qquad i = 1, 2, \ldots, q$$
$$v_0^{[0]} \equiv \sum_{i=i}^{q} x_i^{[1]} y_i^{[1]} = \sum_{i=1}^{q} \lambda^{-2} M_i - \lambda^{-3} M_{q+1} + \left(\lambda^{-3} - q\lambda^{-2}\right) M_{q+2} + O(\lambda)$$

where $v_j^{[J]}$ is the dual to the variable $z_j^{[J]}$, and the equivalence is gotten by identifying coefficients of $z_j^{[J]}$ in both sides of (5).

Figure 1: Easy Version of Coppersmith & Winograd Method[1]

 

Following Figure 2 Shows Original complex version of coppersmith and winograd algorithm.

$$\sum_{i=1}^{q} \lambda^{-2}\left(x_0^{[0]} + \lambda x_i^{[1]}\right)\left(y_0^{[0]} + \lambda y_i^{[1]}\right)\left(z_0^{[0]} + \lambda z_i^{[1]}\right)$$
$$- \lambda^{-3}\left(x_0^{[0]} + \lambda^2 \sum x_i^{[1]}\right)\left(y_0^{[0]} + \lambda^2 \sum y_i^{[1]}\right)\left(z_0^{[0]} + \lambda^2 \sum z_i^{[1]}\right)$$
$$+ \left[\lambda^{-3} - q\lambda^{-2}\right]\left(x_0^{[0]} + \lambda^3 x_{q+1}^{[2]}\right)\left(y_0^{[0]} + \lambda^3 y_{q+1}^{[2]}\right)\left(z_0^{[0]} + \lambda^3 z_{q+1}^{[2]}\right)$$
$$= \sum_{i=1}^{q}\left(x_0^{[0]} y_i^{[1]} z_i^{[1]} + x_i^{[1]} y_0^{[0]} z_i^{[1]} + x_i^{[1]} y_i^{[1]} z_0^{[0]}\right) +$$
$$x_0^{[0]} y_0^{[0]} z_{q+1}^{[2]} + x_0^{[0]} y_{q+1}^{[2]} z_0^{[0]} + x_{q+1}^{[2]} y_0^{[0]} z_0^{[0]} + O(\lambda).$$

Figure 2: Complex Version of Coppersmith & Winograd Method[1]

**Complexity**

Consider that this is just snapshot of original algorithm. Even if we take any of the two version, it can be said that complexity will be of order 2.496[7,8]. Though it has been improved to have complexity of order 2.376[5,6]. It can be said that it is lower than of the methods described above.

## III. ANALSIS OF RESULTS WITH RESPECT TO TIME TAKEN ON DIFFERENT HARDWARES

To analyze above three algorithm, we have taken matrices of following size. 2*2, 4*4, 6*6, 8*8, 10*10, 20*20, 40*40, 100*100, 500*500, 1000*1000…… upto 10000*10000.

Hardware (Processors) taking for these computations are Pentium P4, Dual Core, Core2Duo, i3, i5 & i7.

Noting that, we have to divide matrix bigger than 2*2 are divided using block method.

Taking all into consideration, we have found that all processors do not show any fluctuations in time taken for computation upto the size of 500*500 size.

But as size increases, time taken for computation increases slightly. But this time is in fractional milliseconds, i.e. for 10000*10000 time taken is around 0.02. While for 8000*8000 it is similar. But if fast processors like i5 or i7 are taken then time taken is less as compared to processors like P4.

## IV. CONCLUSION

From above discussion, it can be said that if small matrices are there, then all the methods give almost equal time. It should be noted that this is true for any processor used. If size of the matrix is increased then it show very less time. From this it can be said that because of speed of processors these computations are done so fast that actual difference of speed due to algorithm can't be seen. But it should also be noted that if these procedures are to be done repeatedly with bigger matrices then use of methods like strassen and coppersmith & winograd can make difference.

## REFERENCES

[1] Coppersmith *and* Winograd, 1987; D. Coppersmith, S. Winograd. Matrix Multiplication via Arithmetic Progressions.
[2] BILMES, J., ASANOVIC, K., CHIN, C.,ANDDEMMEL, J. 1997. Optimizing matrix multiply using PHiPAC:a portable, high-performance, Ansi C coding methodology. In International Conference on Supercomput-ing.
[3] BLACKFORD, L. S., DEMMEL, J., DONGARRA, J., DUFF, I., HAMMARLING, S., HENRY, G., HEROUX, M.,KAUFMAN, L., LUMSDAINE, A., PETITET, A., POZO, R., REMINGTON, K.,ANDWHALEY, R. C. 2002. An updated set of basic linear algebra subprograms (BLAS).ACM Transaction in Mathemathical Soft-ware 28,2, 135–151.

 

[4] BODRATO, M. 2010. A Strassen-like matrix multiplication suited for squaring and higher power computa-tion. InISSAC '10: Proceedings of the 2010 international symposium on Symbolic and algebraic compu-tation. ACM, New York, NY, USA. http://bodrato.it/papers/#ISSAC2010

[5]Aho, Ullman. he design and analysis of computer algorithms.

[6]N . Alon, A. Shpilka, and C. Umans. On sunflowers and matrix multiplication. ECCC TR11-067, 18.2011

[7] Junjie Li Sanjay Ranka Sartaj Sahni Strassen's Matrix Multiplication on GPUs, https://www.cise.ufl.edu/~sahni/papers/strassen.pdf

[8] D. Bailey, K. Lee, and H. Simon, Using Strassen's algorithm to accelerate the solution of linearsystems,Jr. of Supercomputing, 4, 357-371, 1990

[9] B. Boyer, C. Pernet, and W. Zhou, Memory efficient scheduling of Strassen-Winograd's matrixmultiplication algorithm,ACM ISSAC, 2009

[10] https://www.cise.ufl.edu/~sahni/papers/strassen.pdf

[11] http://www.cs.toronto.edu/~yuvalf/Limitations.pdf

[12] http://bioinfo.ict.ac.cn/~dbu/AlgorithmCourses/Lectures/MAndersonSBarman2009.pdf

[13] http://www.cs.mcgill.ca/~pnguyen/251F09/matrix-mult.pdf

[14] *D. Bini, M. Capovani, F. Romani, and G. Lotti.O(n2:7799)complexity for n*n approximate matrix multiplication.Inf. Process. Lett., 8(5):234–235, 1979*

 