Reviewing Matrix Multiplication

Neelam Amien, Abida Naseem

Abstract—Algorithm written using different methods can still give same result. One of the ways to determine whether a solution is optimal or not is to determine how much time does it take to solve the specific problem. The problem that is targeted in this paper is Matrix multiplication that is widely used in many scientific computations. Different solutions of this problem are evaluated in this paper and all of them are compared on the basis of their time complexity. After comparing five most known algorithms for matrix multiplication we concluded that *coppersmith-winograd* algorithm is fastest in terms of time.

Index Terms—Algorithm, Complexity, Coppersmith, Winograd, Comparison, Matrix multiplication, Strassen

1 Introduction

As we all know algorithm is the set of instructions to be followed by a computer or machine to solve a specific problem. This set of instructions for the same problem can vary according to the thinking of the one devising it. It means same problem can have different solution but same result depending on the method adopted to approach the solution. It depends on the user which solution he uses to proceed with his problem depending on his requirement (less space, less time, less resources consumption).

Most of the algorithms now a day are judged based on their time complexity that is the time an algorithm takes to solve some specific problem. Gone are the days when space complexity was also considered as an important factor in efficiency of the algorithm because memory is now very cheaper.

One other element that is considered while examining the efficiency of an algorithm is its usage of processors. Some algorithms mostly small uses one processor to compute the problem and provide the results(Sequential Execution).While algorithm with complex statements and having sub-problems in them require more than one processor to compute their sub problems simultaneously on different CPU's(Parallel Execution).

In this paper we will re-examine the algorithms of matrix multiplication and judge them on the basis of key factor of their efficiency that is time complexity.

Matrix multiplication is the base of many computational problems including theory of networks, transforming co-ordinate systems, modeling of population, translation, scaling and rotation of graphics, quantum mechanics, counting number of walks between the two nodes of a graph and many more. Matrix multiplication is basically correspondence to linear map compositions**Error! Reference source not found.**.

In Matrix Multiplication two Matrix A and B are multiplied providing that the count of matrix A columns must be equal to count of matrix B rows and result is stored in a C matrix having rows count of matrix A(matrix 1) and column count of B (matrix 2).

$$\underset{A}{\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}} \underset{B}{\begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix}} = \underset{C}{\begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix}}$$

As mentioned earlier many solutions have been provided by different authors to solve the problem of matrix multiplication efficiently. But time is the most expensive commodity now a days. An algorithm that works faster occupying more memory is lot better than an algorithm that is slower but occupies less computer memory that is why we will discuss some well-known approaches in this paper and will find the one which is optimal in terms of time.

1. Naïve approach of matrix Multiplication
2. PUMMA
3. SUMMA
4. Strassen's Algorithm
5. Canon's Algorithm
6. Coppersmith-Winograd algorithm

We will test each of the above algorithms. Our paper is arranged in following manner: in section 2 we give an evaluation study related to matrix multiplication, in section 3 we give a brief review of naïve approach of matrix multiplication, in section 4 we present SUMMA algorithm. Similarly, section 5 presents PUMMA, in section 6 Strassen's algorithm is presented, Section 7 present Canon's algorithm, and Section 8 displays Coppersmith-Winograd algorithm, Comparison of above algorithms is presented in Section 9, Section 10 evaluates Strassen and CW algorithm briefly. Finally, conclusion and future work is discussed in Section 11.

2 Related Works

In [1], the authors of the paper were interested in reduction of multiplication cost for small size matrices. Observing the preceding effort of Mezzarobba, Fischer, Probert, and Smith, in a similar domain, they kept Strassen, Pan, Laderman, Winograd, algorithms for small matrices as base, shown the way to utilize these merit algorithms in a better course of action. They elaborated the usage of their computed results by illustrating and creating codes of multiplication on several

• *First Author is residing in District Mandi Bahauddin, 50400, Pakistan, E-mail: neelamamien881@gmail.com.*
• *Second Author is residing at Mitha Tiwana, District Khushab, 41250, Pakistan, E-mail: abidanaseem3719@gmail.com*

areas, such as differential operators, integers, linear operators of recurrence and polynomials. In **Error! Reference source not found.**, the authors examined the execution and extensibility of several of matrix multiplication algorithms of parallel distribution and envisioned the circumstances under which those formulation are superior to the other formulations. They concluded that the Gustafson-Kessel algorithm presented in their paper defeated all other algorithms devised for a specific number of processors and sizes of matrices.

In **Error! Reference source not found.**, the authors tried to make Strassen's algorithm memory efficient. First, the algorithm centrally uses a Morton (nonstandard array layout) based on a quad-tree partitioning of the matrix. Second, they selected the recursion trimming point dynamically to lower distance without influencing the algorithm's performance. Each strategy is critical for execution, and these combination of their codes increases their effectiveness. After comparing their algorithm with other implementation performance wise they showed that their implementation often defeats the substitute techniques (to 25%).

In [14], the authors wrote about the production of an efficient and easy implementation of algorithm of Strassen of matrix-multiplication. The idea designed by them can substitute the level III BLAS multiplication of matrix method. Optimal performance and low memory size were achieved for all sizes of matrices. Authors also reported that algorithm of Strassen is practical approach for big size matrices.

In **Error! Reference source not found.**, a new distribution scheme was proposed by the author for Strassen algorithm for parallel multiplication of matrix on clusters of heterogeneous type. So their implementation not only achieved load balancing but also reduced the count of total operation. Consequently they succeeded in achieving nearly 21.7% speedup in comparison to the classical parallel heterogeneous clustering environment Strassen's algorithm.

3 Naive Approaches

Naive matrix multiplication refers to the approach that we use for multiplication of matrices. Explicitly, suppose two Matrix A and B are multiplied providing

Case 1: Matrix A and matrix B are matrices of same size(square) and result is stored in a C matrix which will also be square matrix. Where a11, b11, c11 are first elements of each matrix A, B, and C respectively. Similarly a12, b12, and c12 are the second element of first row and first element of second column in matrix A, B and C respectively and so on.

$$\overset{A}{\begin{bmatrix} a11 & a12 \\ a21 & a22 \end{bmatrix}}\overset{B}{\begin{bmatrix} b11 & b12 \\ b21 & b22 \end{bmatrix}} = \overset{C}{\begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix}}$$

Case 2: That first matrix is a row matrix and second matrix is a column matrix the product will be a 1x1 matrix.

$$\overset{A}{\begin{bmatrix} a11 & a12 \end{bmatrix}}\overset{B}{\begin{bmatrix} b11 \\ b21 \end{bmatrix}} = \overset{C}{\begin{bmatrix} c11 \end{bmatrix}}$$

Case 3: That first matrix is a column matrix and second matrix is a row matrix and the product will be an outer product matrix

$$\overset{A}{\begin{bmatrix} a11 \\ a21 \end{bmatrix}}\overset{B}{\begin{bmatrix} b11 & b12 \end{bmatrix}} = \overset{C}{\begin{bmatrix} c11 & c12 \\ c21 & c22 \end{bmatrix}}$$

Where,
A = (Apk)1 <= p <= n,1 <= k <= m is a nxm matrix
B = (kl)1 <= k <= n,1 <= l <= p is a nxp matrix
C = (cpl) 1 <= p <= m, 1 <= l <= p

3.1 Complexity of Naïve approach

Time complexity of naïve approach for all the three cases discussed as follows.

As per case 1 when variables m, n, p are equal the multiplications needed would be n^3. Similarly in case 2 where n can be arbitrary while m and p are equal than required multiplications would be N. In case 3 where n=1 considering m and p arbitrary the required multiplications would be N. So the time complexity for the naïve approach using 4 addition and 8 multiplication using master theorem is $O(n^3)$.

4 Summa

The summa (Scalable Universal Matrix Multiplication Algorithm) is more practical algorithm that requires less workspace. This algorithm is less efficient but is quite easy to generalize. It also uses shift algorithm and broadcast technique. This algorithm completes its process in four simpler steps: first, owner of partial row and column 0 broadcast row and column along its process column and row respectively. Second, owner of partial row and column 1 broadcast row and column along its process column and row respectively. Third, owner of partial row and column 2 broadcast row and column along its process column and row respectively and at the end the sum of all entries are computed in a single matrix for a 3x3 matrix multiplication. Basic algorithm for Scalable universal matrix multiplication**Error! Reference source not found.**:

```
SUMMA
for k = 0 to n − 1 do
for all i = 1 to $p_{row}$ do
Owner of A(i, k) broadcasts it to whole processor row;
end for
for all j = 1 to $p_{column}$ do
owner of B(k, j) broadcasts it to whole processor column;
end for
A(i, k)=$A_{column}$
B(k, j)=$B_{row}$
C = C + $A_{column}$ * $B_{row}$
end for
```

where, p_{row} and p_{column} are processor grids and $p_{row} = p_{column}$. So, the time complexity of SUMMA algorithm become $T(p) = 2 * n^3/p + \alpha * \log p * n/b + \beta * \log p * n^2/s$. This can be generalized as n^3 with α= starting cost of message, β=bandwidth.

5 Pumma

PUMMA (Parallel Universal Matrix Multiplication Algorithm) is used for performing matrix multiplication operations on systems distributed memory. In a distributive memory systems there are more matrix multiplication than an ordinary sequential system. In this algorithm multiple processors are used to compute the result of multiplication of Matrices A B. Parallel matrix multiplication is a displacement algorithm .in which data distribution in block cycles is displayed over two-dimensional processor grid[13].

The general matrix multiplication see routine for optimal pure block decomposition. Routnes xGEMM is an example which is optimal whenever considering the routine is isolated.

```
PUMMA
Do k = 0, Mb_ 1
With each column across different template
Prado l= 0, Mb -- 1
Kp  | (k+1, Lb) |
Prado j ==0  Nb -- 1
F ( I,j )= F(I,j ) + M(I,kp ) . N ( k ,p, j )
Terminate Prado
Role a leftword
End do.
```

The processor template's column at the initial of the step is multiplied with block broadcast containing the required routine.

```
PUMMA Contd.
Include mp I .h
#--define F(I,j)  (f[j*1df+I])
#--define g (i,j )  ( f[j*1dg+I])
#--define H ( I,j) (g[j*1dc +I])
#--define min (x,y) ( ( x) <((y) ? (X) :(y))
Int I one =1;
double d one = ,0;
d_zero = 0,0;
void pdgmm(m,n ,k,ng,alpha,f,Idf,g , I dg,beta,h,idg ,mf ,nf,mg,ng,mh, nh,commrow ,commoncol,work1,work20)
Int m, n, k, ng, mf[], nf[] mg [], ng[] mg[], nh[], Idf,idg ,idh ;
Double *f, *g, *h;
alpha, beta,
*Work1,*work2;
Mpi_comm comm_row
{Int myrow ,my col,
nprow , npcol,
i,j,kk,iwrk,icurrow,icurcol,
ii,jj
double xtemp;
double xp
For (c=o;c<n_h[[ mycolumn] ; c++)
For (d=0 ; d< m_h[myrow]; d++)
H (c,d) = beta *H (c,d);
```

As per its parallel nature PUMMA Algorithm also shows time complexity of $O(n^3)$.

6 Strassen's Algorithm

In 1960 Strassen provided a method of matrix multiplication and claimed that this algorithm has less time complexity than that of classical approach to matrix multiplication. Basically this algorithm reduced the multiplication operation that cannot be solved in linear time and increased the addition operation that are solvable in linear time hence decreased the time complexity**Error! Reference source not found.**.

So this algorithm basically reduces the recursive calls to 7(that was originally 8 in classical approach.)It divides the matrices to sub matrices of N/2xN/2 size and compute their result

$$\overset{A}{\begin{bmatrix} l11 & l12 \\ l21 & l22 \end{bmatrix}} \overset{B}{\begin{bmatrix} m11 & m12 \\ m21 & m22 \end{bmatrix}} = \overset{R}{\begin{bmatrix} r11 & r12 \\ r21 & r22 \end{bmatrix}}$$

Where,
r11=s5+ s4 -s2+ s6,
r12=s1+s2,
r21=s3+s4,
r22=s1 +s5- s3- s7, and
s1 = l11(m12-m22),
s2 = (l11+l12) m22,
s3 = (l21+l22) m11,
s4 = l22(m21-m11),
s5 = (l11+l22) (m11+m22),
s6 = (l12-l22) (m21+m22),
s7 = (l11-l21) (m11+m12)

As addition/subtraction of 2 matrices A and B takes $O(N^2)$ time. Its time of execution can be written as S(N)=7S(N/2) + O(N2). Time complexity of Strasssen algorithm is $O(n^{Log7}) \approx O(n^{2.8074})$ by Master Theorem.

7 Cannon's Algorithm

Cannon's matrix multiplication algorithm is an algorithm distributed for two-dimensional meshes that was given by Lynn Elliot Cannon in 1969.

It uses shift mechanism for multiplication of matrices and provide processes equal to order of matrix and operations equal to sqrt(processes).

$$\text{For A=}\begin{bmatrix} l11\ l12\ l13\ l14 \\ l21\ l22\ l23\ l24 \\ l31\ l32\ l33\ l34 \\ l41\ l42\ l43\ l44 \end{bmatrix}$$

$$\text{For B=}\begin{bmatrix} m11\ m12\ m13\ m14 \\ m21\ m22\ m23\ m24 \\ m31\ m32\ m33\ m34 \\ m41\ m42\ m43\ m44 \end{bmatrix}$$

Every element of matrix will be possessed by different processes hence total no of processes for above 4x4 matrix will be 16 and total number of steps will be square root (16) = 4.

Step1: we find A1 and B1 from matrix A and matrix B respectively by following process:

$$A=\begin{bmatrix} a11 & a12 & a13 & a14 \\ a21 & a22 & a23 & a24 \\ a31 & a32 & a33 & a34 \\ a41 & a42 & a43 & a44 \end{bmatrix}\begin{bmatrix} \text{0 left shift} \\ \text{1 left shift} \\ \text{2 left shift} \\ \text{3 left shift} \end{bmatrix}$$

$$A1=\begin{bmatrix} a11\ a12\ a13\ a14 \\ a22\ a23\ a24\ a21 \\ a33\ a34\ a31\ a32 \\ a42\ a43\ a44\ a41 \end{bmatrix}$$

For B1 we do 0,1,2,3 shifts from bottom respectively

$$B=\begin{bmatrix} b11\ b12\ b13\ b14 \\ b21\ b22\ b23\ b24 \\ b31\ b32\ b33\ b34 \\ b41\ b42\ b43\ b44 \end{bmatrix}, B1\begin{bmatrix} b11\ b22\ b33\ b24 \\ b21\ b32\ b43\ b34 \\ b31\ b42\ b13\ b44 \\ b41\ b12\ b23\ b14 \end{bmatrix}$$

C1[m][n] = A1[m][n] * B1[m][n] for m=i, n=j

Step 2: we find A2 from A1 and B2 from B1 using following process:

$$A2=\begin{bmatrix} l11\ l12\ l13\ l14 \\ l22\ l23\ l24\ l21 \\ l33\ l34\ l31\ l32 \\ l42\ l43\ l44\ l41 \end{bmatrix}\begin{bmatrix} \text{1 left shift} \\ \text{1 left shift} \\ \text{1 left shift} \\ \text{1 left shift} \end{bmatrix}$$

$$A2=\begin{bmatrix} l12 & l13 & l14 & l11 \\ l23 & l24 & l21 & 22 \\ l34 & l31 & l32 & l33 \\ l43 & l44 & l41 & l42 \end{bmatrix}$$

For B2 we will do 0,1,2,3 shifts from bottom of B1 respectively.
C2[m][n] = A2[m][n] * B2[m][n]

Step 3: A3, B3 will be obtained from A2 and B2 by the same procedure in step 2.
C3[m][n] = A3[m][n] * B3[m][n]

Step 4: A4, A3 will be obtained from A3 and B3 by the same procedure in step 2.
C4[i][j] = A4[i][j] * B4[i][j]
and finally, result will be C = C1+C2+C3+C4
Time complexity:

1) The maximum shift distance of matrix shift is $\sqrt{p} - 1$
2) Each $\sqrt{p}$ single-step shifts in the compute-and-shift phase takes $ts+twn^2/p$ time
3) By Multiplying $\sqrt{p}$ submatrices of size $(n/\sqrt{p})\times(n/\sqrt{p})$ total time taken is : n^3/p.

8 Coppersmith -Winograd

This Algorithm was given by two authors \ coppersmith and Winograd and it was the quickest matrix multiplication algorithm from 1990 to 2010. It multiplies p*p matrices in $O(p^{2.375477})$time.

This algorithm is an up gradation of the naive algorithm with O(n3) time and also taking the less time than the algorithm of Strassen which takes $O(n^{2.807355})$ time to execute**Error! Reference source not found.**.

Algorithm is rarely in use and impractical because of its large constant factors in their running time and there is possibility for the improvement of exponents further, which required it as exponent power should be at least 2 mean $m*m=m^2$ value in the computed result. For example, if input: M1={{1,2},{3,4}} and M2= {{3,2},{5,1}}, then the result={{13,4},{29,10}} along with the output: resultant matrix will be matching.

```
Coppersmith Winograd Algorithm:

1: Start
2: By taking the matrices of M1, M2, M3 as input of
   (n*n).
3: Choose matrices a[n][1] and randomly will be zero or
   1.
4: Calculate matrix multiplication M2*a, and M3
   multiply with a and M1*(M2*a) for computing the
   M1*(M2*a) M3*a…
5: Verify it if M1*(M2*a) – M3*a multiplication performs.
6: If it is 0 or false matrices multiplication will be correct
   otherwise program will end.
```

Andrew Stothers improved the algorithm in 2010 with display style mythical O $(n^{2.374})$ display style mythical O $(n^{2.374})$. In 2011 Virginia William's strengthen the bound of display style mythical O $(n^{2.3728642})$ by incorporating a mathematical shortcut from paper of Stothers incorporating her own observations and automatic computers optimization.

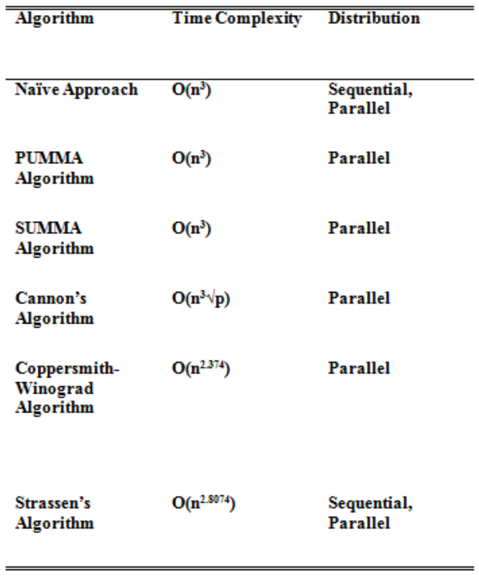
To prove theoretical time bounds the algorithm of this algorithm is often used as a base in other algorithms.Dissimilar to the Strassen algorithm, it is rarely in use now because its only useful for the matrices that are too large for modern hardware to handle (as making its galactic algorithm).The coppersmith Winograd algorithm has been re_derived using a group of theoretic construction by Robert Kleinberg , Henry Cohn , Chris Umans and Balazs Szegedy. They also demonstrated that any of two conjunctions implies that the ideal 2 as exponent of matrix multiplication has long been assumed.They were anyway unable to come up with a particular solution that would result in faster time.

9 Comparisons

Based on time complexities we can compare all the algorithms as follow:

TABLE 1

COMPARISON OF ALGORITHMS



Algorithm	Time Complexity	Distribution
Naïve Approach	$O(n^3)$	Sequential, Parallel
PUMMA Algorithm	$O(n^3)$	Parallel
SUMMA Algorithm	$O(n^3)$	Parallel
Cannon's Algorithm	$O(n^3\sqrt{p})$	Parallel
Coppersmith-Winograd Algorithm	$O(n^{2.374})$	Parallel
Strassen's Algorithm	$O(n^{2.8074})$	Sequential, Parallel

As we have now concluded that other algorithm except Strassen's and Coppersmith-Winograd are not efficient in terms of time complexity so in the next section we'll target the comparison of both the algorithms.

10 Strassen Vs Coppersmith-Winograd

In an algorithm for the dot product of matrices of any shapes and sizes it is very inappropriate approach to first fill the matrices with 0 and to the next power 2. Adding 0's in one row or column of each dimension is more appropriate, but the best approach is to divide into different sizes, using recurrences and shifting to Winograd's or the naïve approach for small matrices. It is certainly ineffective to use Strassen's recursion for 1×1 matrices.

The best exponent gradually dropped over the following few years. The best record now is still the 2.376 by Coppersmith and Winograd. However, this algorithm involves a lot of constant factors. So the Strassen's algorithm is the only practical approach.

Since the time complexity of Coppersmith-Winograd algorithm for multiplication of two matrices in O($n^{2.37}$). On the other hand Strassen's Method provide O($n^{2.80}$) time complexity for multiplication of two square matrices. Hence based on time complexity Coppersmith-Winograd algorithm is better than Strassen's algorithm.

11 Conclusion And Future Work

Based on their time complexities, we can conclude that Coppersmith-Winograd algorithm is the best algorithm from all other algorithms that are discussed in this article, so far. But if it comes to sizes of matrices the more practical approach is Strassen's algorithm because of involvement of huge constant factors.

In future, we will try to run these algorithms on specific hardware and report the results accordingly.

REFRENCES

[1] G. W. Stewart, "Matrix Algorithms," *Matrix Algorithms*, pp. 1–12, 1998, doi: 10.1137/1.9781611971408.

[2] Rivera, Cody, Jieyang Chen, Nan Xiong, Jing Zhang, Shuaiwen Leon Song, and Dingwen Tao. "TSM2X: High-performance tall-and-skinny matrix–matrix multiplication on GPUs." Journal of Parallel and Distributed Computing, 151: 70-85, 2021.

[3] Drevet, C, Islam, M and Schost, R. *"Optimization techniques for small matrix multiplication*". Science Direct. Theoretical Computer Science, 412, 2219–2236, 2011

[4] Robinson, Sara. "Toward an optimal algorithm for matrix multiplication." SIAM news 38, No. 9: 1-3, 2005.

[5] Li, Junjie, Sanjay Ranka, and Sartaj Sahni. "Strassen's matrix multiplication on GPUs." IEEE 17th International Conference on Parallel and Distributed Systems. IEEE, 2011.

[6] Petros Drineas, Ravi Kannan, and Michael W. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. SIAM Journal on Computing, 36(1):132–157, 2006.

[7] Gupta, A and ICumar, V. "Scalability of Parallel Algorithms for Matrix Multiplication". International Conference on Parallel Processing, 1993.

[8] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing, pages 1–6, New York, NY, USA, 1987. ACM Press.

[9] Alman, Josh, and Virginia Vassilevska Williams. "A refined laser method and faster matrix multiplication." In Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 522-539. Society for Industrial and Applied Mathematics, 2021.

[10] Thottethodi, M, Chatterjee, S and Lebeck, A. "Tuning Strassen's Matrix Multiplication for Memory Efficiency". ACM/IEEE SC98 Conference (SC'98), 1998.

[11] J. A. Gunnels, G. M. Henry, and R. A. van de Geijn, "A family of high-performance matrix multiplication algorithms," Lecture Notes in Computer Science (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), Vol. 2073, No. 2, pp. 51–60, 2001, doi: 10.1007/3-540-45545-0_15.

[12] Lederman, S, Jacobson, E, Johnson, J, Tsao, A and Turnbull, T. "Implementation of Strassen's algorithm for Matrix Multiplication". ACM/IEEE Conference on Supercomputing (SC'96), 1996.

[13] S. Huss-Lederman, E. M. Jacobson, A. Tsao, T. Turnbull, and J. R. Johnson, "Implementation of Strassen's algorithm for matrix multiplication," no. August, pp. 32-es, 1996, doi: 10.1145/369028.369096.

[14] R. A. Van De Geijn and J. Watts, "SUMMA: Scalable universal matrix multiplication algorithm," *Concurr. Pract. Exp.*, vol. 9, no. 4, pp. 255–274, 1997.

[15] D. W. Walker, "Accepted for publication in Concurrency: Practice and Experience (1994) PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers 1," Vol. 1, No. 1994, 2008.

[16] Ohtaki, Y. "Parallel Implementation of Strassen's Matrix Multiplication Algorithm for Heterogeneous Clusters". IEEE. 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 2004.