World Applied Sciences Journal 18 (10): 1455-1462, 2012
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2012.18.10.1818

Sub Tasks Matrix Multiplication Algorithm (STMMA)

[1]Hussam Hussein Abuazab, [1]Adel Omar Dahmane and [2]Habib Hamam

[1]Department of Electrical and Computer Engineering, Université Du Québec À Trois-Rivières, Trois-Rivieres, Quebec, Canada
[2]Faculty of Engineering, University De Moncton, Canada

Abstract: Several parallel matrix multiplications developed since four decades based on decomposing the multiplied matrices into smaller size blocks, the blocks will be distributed among the processors to run matrix multiplication in shorter time than when only one processor will run the whole matrix multiplications. All parallel matrix multiplications algorithms suffer from three points:

- The optimal size of the block of the decomposed matrices.
- The optimal number and size of the exchanged messages between the processors.
- Data dependency between the processors.

And some algorithms suffer from fourth point, which is load balance especially with non-square matrix multiplications. In this paper we will introduce new parallel matrix multiplications to overcome the above four drawbacks, which will result in vast difference in performance in terms of time and load balance. For example, for a matrices multiplication of 5000×5000, it consumes 2812 seconds using cannon algorithm, while only 712 seconds needed using STMMA to accomplish the same task using MPI library, which implies 4 times faster. We have used Processors Intel(R) Core(TM) i5 CPU 760 @2.80GHz 2.79 GHz, Installed memory (RAM) 4.00 GB, System type: 64-bit Operating System, Windows 7 Professional. We have coded the algorithms using Microsoft C++ ver. 6, with MPI Library.

Key words: Parallel matrix multiplication · Speedup · Efficiency

INTRODUCTION

Parallel matrix multiplication is one of the most important linear algebra operations that involved in so many fields like image processing, ocean studies and aerospace engineering and all numerical analysis fields.

The concept of parallel programming is to divide the total work among more than one processor in a way that minimizes total solution time and resources utilization (CPU and Memory).

This goal is pursued by equally distributing load to processors (i.e., load balancing), where each processor will do the part of the multiplication.

Parallel matrix multiplication has been investigated extensively since1969. All algorithms in this field till now depend on decomposing the matrices into several blocks. This includes the Systolic algorithm [2], Cannon's algorithm being developed in 1969 [3], Fox and Otto's algorithm [4], PUMMA (Parallel Universal Matrix Multiplication) [5], SUMMA (Scalable Universal Matrix Multiplication) [6] and DIMMA (Distribution Independent Matrix Multiplication) [7].

Several new algorithms being developed recently as the result of heavy demands on matrix multiplications, but still, these new algorithms using the decomposition of the matrices to be multiplied, one algorithm were developed recently by Sotiropoulos and Papaefstathiou in 2009 [8] and another algorithm being developed by Cai and Wei in 2008 [9], another algorithm being developed by Michael Bader, Sebastian Hanigk, Thomas Huckle, in 2007 [14], another algorithm being developed by Duc Kien NGUYEN, Ivan LAVALL´EE, Marc BUI, Quoc Trung HA,

Corresponding Author: Hussam Hussein Abuazab, Department of Electrical and Computer Engineering, Université Du Québec À Trois-Rivières, Trois-Rivieres, Quebec, Canada.

in 2005 [13], another algorithm being developed by James Demmel, Mark Hoemmen, Marghoob Mohiyuddin and Katherine Yelick, in 2008 [12], another algorithm being developed by Ardavan Pedram, Masoud Daneshtalab and Sied Mehdi Fakhraie, in 2006 [11], another algorithm being developed by Yunfei Du, Panfeng Wang, Hongyi Fu, Jia Jia, Haifang Zhou, Xuejun Yang, in 2007 [10]. In this research [10], new direction appears, which is to avoid the failure node in parallel matrix multiplications. Another new direction appears also in [9], which the hardware implementation of Parallel Matrix Multiplications.

Agarwal *et al*. [15] developed a three-dimensional (3D) matrix multiplication algorithm that overlaps communication with computation, but still it uses the theory of decomposing the multiplied matrices.

Manojkumar Krishnan and Jarek Nieplocha developed new approach called SRUMMA [16] differs from the other parallel matrix multiplication algorithms by the explicit use of shared memory and remote memory access (RMA) communication rather than message passing, but could not give up of the matrix decomposition.

The paper is organized as follows. The next section goes through the previous parallel matrices multiplication algorithms; Section 3 describes our algorithm (STMMA) and analyzes its efficiency model.

In Section 4, an experimental study of algorithm STMMA is presented. In Section 5 comparison with some selected algorithms will be presented. Finally, conclusions are given in Section 6.

Previous Algorithms

Systolic Algorithm: In this algorithm, matrices A, B are decomposed into submatrices of size $\sqrt{P} \times \sqrt{P}$ each, where P is number of processors. The basic idea of this algorithm is the data exchange and communication occurs between the nearest-neighbors.

Cannon's Algorithm: To multiply matrices A and B, the blocks of matrix B to circulate vertically and blocks of A horizontally in ring fashion, Blocks of both matrices must be initially aligned using circular shifts so that correct blocks meet as needed to generate the product matrix C. In fact, it replaces the traditional loop:

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,k} \times B_{k,j}$$

With the loop

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k) mod \sqrt{P}} \times B_{(i+j+k) mod \sqrt{P},j}$$

Fox's Algorithm with Square Decomposition: Fox's algorithm for multiplication organize C =A x B into submatrix on a P processors, as shown in the figure below, where the square multiplied matrices will be decomposed into square processor template.

The algorithm runs P times, in each turn, it broadcasts corresponding submatrix of A on each row of the processes, run local computation and then shift array B for the next turn computation. The main disadvantages, it is applied only for square matrices.

P0	P2
P1	P3

Fox's Algorithm with Scattered Decomposition: It is same Fox's algorithm, but the multiplied matrices will be scattered decomposed, where the matrix is divided into several sets of blocks. Each set of blocks contains as many elements as the number of processors and every element in a set of blocks is scattered according to the two-dimensional processor templates.

Pumma (MBD2): It an algorithm that performs parallel matrix multiplication by applying the transpose matrix on distributed memory concurrent computers.
It applies the following equation:

$$C = A^T.B,\ C = A.B^T,\ C = A^T.\ B^T$$

Summa: It is simply can be summarized by the equation:

$$C(i,j) = C(i,j) + \sum_k A(i,k) * B(k,j)$$

Dimma: The algorithm is based on two new ideas; it uses a modified pipelined communication scheme to overlap computation and communication effectively and exploits the LCM block concept to obtain the maximum performance of the sequential BLAS routine in each processor even when the block size is very small as well as very large [17].

New Algorithm (STMMA)

Algorithm description: Unlike previous algorithms, STMMA does not define new algorithm, instead, it follows the serial matrix multiplications shown below:

```
1: for I=0 to s-1 {
2:  for J=0 to s-1 {
3:      for K=0 to s-1 {
4:       CIJ= CIJ+ AIK ×BKJ
5:             }
6:      }
7: }
```

The main idea in this algorithm, the multiplied matrices A and B will not be decomposed into submatrices and then being distributed among processors, instead, both matrices A and be will be sent initially to each processor and each processor will execute subtask(s) of the matrix multiplication. For example for $A_{4x4} \times B_{4x4}$ to be executed on four processors P_0, P_1, P_2 and P_3, both matrices will be sent to all processors and each processor will produce part of the matrices multiplication result as shown of Fig. 1

Each processor will do separate independent task, to minimize time consuming for exchange intermediate results between the processors and to avoid keep any processor idle waiting results from other processors. So, P_0 will execute the following code:

```
1: for J=0 to 3 {
2:      for K=0 to 3 {
3:       C0J= C0J+ A0K ×BKJ
4:             }
5:      }
```

While P_1 will run the following code:

```
1: for J=0 to 3 {
2:      for K=0 to 3 {
3:       C1J= C1J+ A1K ×BKJ
4:             }
5:      }
```

While P_1 will execute the following code:

```
1: for J=0 to 3 {
2:      for K=0 to 3 {
3:       C2J= C2J+ A2K ×BKJ
4:             }
5:      }
```

While P_2 will execute the following code:

```
1: for J=0 to 3{
2:      for K=0 to 3 {
3:       C3J= C3J+ A3K ×BKJ
4:             }
5:      }
```

While P_3 will execute the following code:

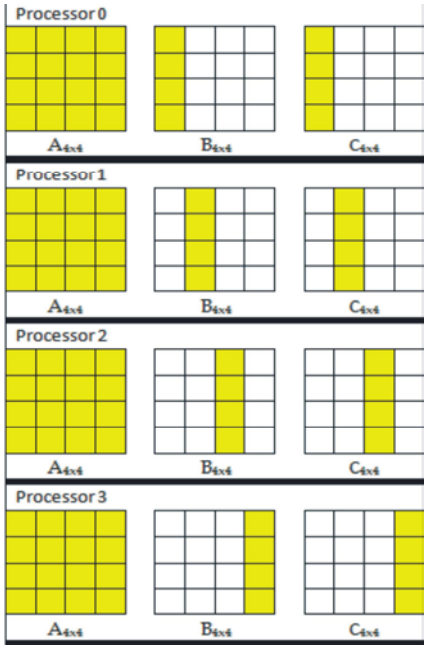

Fig. 1: Task distribution of $A_{4x4} \times B_{4x4,}$ where each processor will produce part of the result matrix C_{4x4}

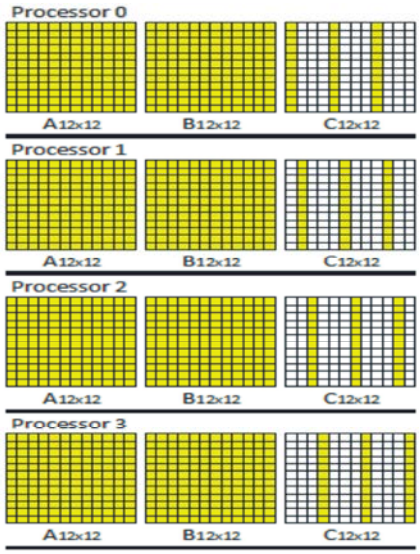

Fig. 2: Task distribution of $A_{12x12} \times B_{12x12,}$ where each processor will produce part of the result matrix $C_{12x12.}$

As we can see, no processor will wait entries from other processors and no processor will send some data to other processor, so we could minimize the data dependency and exchanged messages which has played backward role on the performance of the previous parallel matrix multiplications. To generalize the case mentioned above, for different matrices size, we need to consider

different code; for example for $A_{12x12} \times B_{12x12}$, processor P_0 will produce three tasks, that is to produce the first and fifth and night columns of the matrix $C_{12\times12}$, Fig. 2 shows task distribution of $A_{12x12} \times B_{12x12}$.

Processor 0 will execute the following code:

```
1: for J=0 to 11 {
2:      for K=0 to 11 {
3:      C0J = C0J + A0K ×BKJ
4:            }
5:      }

1: for J=0 to 11 {
2:      for K=0 to 11 {
3:      C4J = C4J + A0K ×BKJ
4:            }
5:      }

1: for J=0 to 11 {
2:      for K=0 to 11 {
3:      C8J = C8J + A0K ×BKJ
4:            }
5:      }
```

While processor P_1 will produce different three tasks, that is to produce the second and sixth and ninth rows of the matrix $C_{12\times12}$ and so on for the remaining processors.

The tasks to be scheduled independently – so not task will need to exchange data between processors – so there will be no time to waste by exchanging messages and no processor will stay idle waiting its input from other processors.

For another case, when we have to parallel multiply the matrices $A_{12x12} \times B_{12x12}$ and number of processors is eight, where 12 is not multiplex of 8. In this case, the tasks to be scheduled between the processors, again independently and with load balanced, so no processor will stay idle while other processors are still doing some tasks. So, we can run the following schedule:

Tasks to be performed		
P1	C[0][0]_C[0][11]	C[8][0]_C[8][5]
P2	C[1][0]_C[1][11]	C[8][6]_C[8][11]
P3	C[2][0]_C[2][11]	C[9][0]_C[9][5]
P4	C[3][0]_C[3][11]	C[9][6]_C[9][11]
P5	C[4][0]_C[4][11]	C[10][0]_C[10][5]
P6	C[5][0]_C[5][11]	C[10][6]_C[10][11]
P7	C[6][0]_C[6][11]	C[11][0]_C[11][5]
P8	C[7][0]_C[7][11]	C[11][6]_C[11][11]

So each processor produced 18 elements of the output matrix C, which satisfies the load balance between the processors and keep each task - and so the processor - independent from any other tasks, also, no exchanged messages.

For the non square matrices multiplications, the tasks to be scheduled within the three constraints:

- Task independency.
- Load balance.
- Processor efficiency.

For non-square matrix multiplication, the task distribution is shown in Fig. 3. In this case, the algorithm will define more independent task and will distribute them equally among the processors.

The other case that we are to test matrix multiplication where the size of the multiplied matrices is not multiplicand of the number of the processors, where defining the independent tasks will take diffident way. Fig. 4 shows task distribution of $A_{12x12} \times B_{12x18,}$ among 4 processors. So, for 18 columns of the result matrix C_{12x18}, each processor will produce 4 columns, where the remaining two columns will be produced either by:

- Two processors, one column for each processor, while the remaining two processors will stay idle, which implies not full load balance, at the last stage, as shown in Fig. 5.
- Four processors, where each processor will produce half of the column, which implies task dependency in this last stage which implies some message exchange and so time consuming at this last stage.

STMMA Efficiency: We will consider the below equation to calculate the efficiency of STMMA algorithm: $C_{m\times n}=A_{m\times k} \times B_{k\times n}$ So sequential time of matrix multiplication algorithm T_{SEQ} is N^3 (in case of square matrix multiplications where $m=n=k=N$). While the parallel time of the same square matrix multiplication algorithm (STMMA) T_{PRL} is N^3/p, while p is number of processors being used in the parallel processing.

The startup cost or latency is to be neglected in the network of sufficient bandwidth.

The speed up is the Sequential Time T_{SEQ} *to* parallel time T_{PRL}

Speed up = T_{SEQ} / T_{PRL}

While

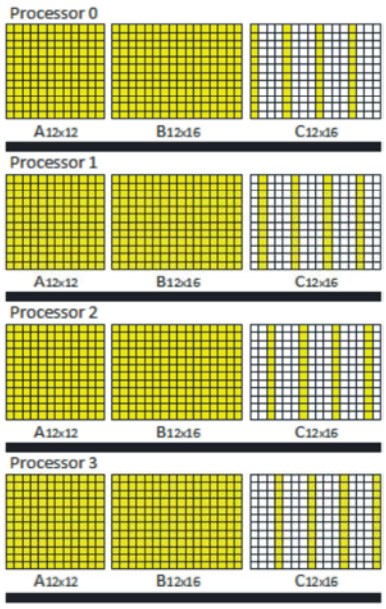

Fig. 3: Task distribution of $A_{12x12} \times B_{12x16,}$ where each processor will produce part of the result matrix $C_{12x16.}$

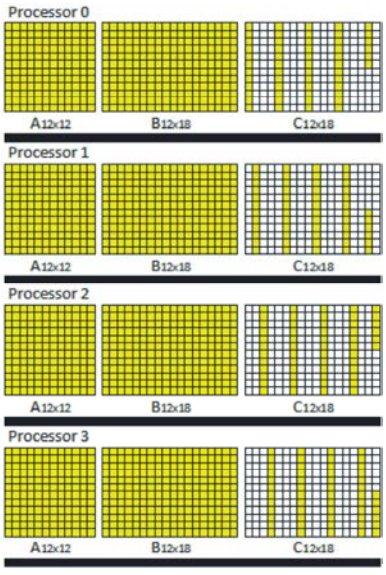

Fig. 4: Task distribution of $A_{12x12} \times B_{12x18,}$ where each processor will produce part of the result matrix $C_{12x18.}$

P3	C12×4	C12×8	C12×12	C12×16	Idle
P2	C12×3	C12×7	C12×11	C12×15	Idle
P1	C12×2	C12×6	C12×10	C12×14	C12×18
P0	C12×1	C12×5	C12×9	C12×13	C12×17
Time	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5

Fig. 5: Processors P2 and P3 will stay idle if we insist on full task dependency

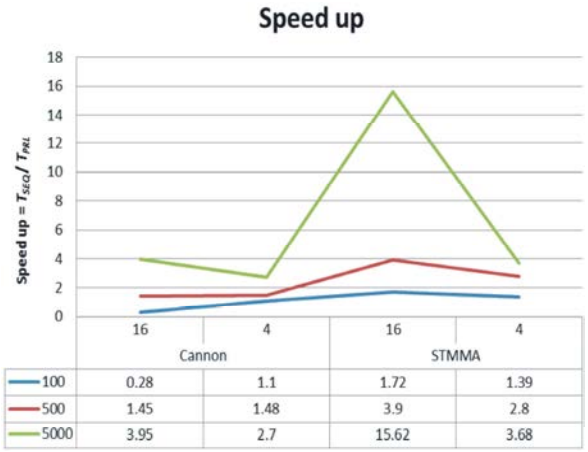

Fig. 6: Speed up of STMMA compared with Cannon

Efficiency (η) = Speedup / p

Experimental Study: Twelve experiments being conducted, for two different architecture, four processors and sixteen processors and three different matrices sizes being used, 100, 500 and 5000 square matrix.

Algorithms, the Cannon and STMMA being used on each test, for each matrix size as shown in Table 1, below.

The high parallel time value on test ID 2, reflects the time being spent for decomposing and exchanging the messages between the processors, which implies that parallel matrix multiplication is not suited for small matrix size. While Fig. 6 shows the performance of the new algorithm STMMA, in terms of Speed up.

Comparisons with Selected Algorithm: Since 1969 many sequential and parallel algorithms are developed for matrix multiplications. In fact, multiplication of large matrices requires a lot of computation time as its complexity is O (n3), so to measure the efficiency of an algorithm for matrix multiplication and to compare many algorithms properly, we will use CLUMEC, so we can carry out each algorithm for huge matrix size and so, we can compare between the new algorithm and the previous algorithms.

The comparison of STMMA will consider the following algorithms:

- Systolic algorithm.
- Cannon's algorithm.
- Fox's algorithm with square decomposition.
- Fox's algorithm with scattered decomposition
- PUMMA (MBD2)
- SUMMA
- DIMMA

Table 1: Comparison between the performance of STMMA and Cannon Algorithms for matrices of 100×100, 500×500 and 5000×5000

Comparison between the performance of STMMA and Cannon Algorithms for matrices multiplication												
Test ID	1	2	3	4	5	6	7	8	9	10	11	12
Algorithm	Cannon		STMMA		Cannon		STMMA		Cannon		STMMA	
Matrix Size (n)	100	100	100	100	500	500	500	500	5000	5000	5000	5000
Decomposition (number of blocks)	4	16	-	-	4	16	-	-	4	16	-	-
Number of processor	4	16	4	16	4	16	4	16	4	16	4	16
Ts (1 processor) (ms)	43				874				11121			
Tp (ms)	39	154	31	25	594	601	312	225	4102	2812	3051	712
Speed up	1.1	0.28	1.39	1.72	1.48	1.45	2.8	3.9	2.7	3.95	3.68	15.62
Efficiency(1-2)	27.5	1.75	34.75	10.75	9.25	9.06	70	24.375	67.5	24.675	25	97.625

Table 2: Systolic algorithm

Task	Execution time	STMMA
Transpose B matrix	$2n^2 t_f$	0
Send A, B matrices to the processors	$2m^2p\ t_c$	$2m^2p\ t_c$
Multiply the elements of A and B	$m^2n\ t_f$	$m^2n\ t_f$
Switch processors' B sub-matrix	$n^2\ t_c$	0
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(m^2\ n + 3n^2) + t_c(4n^2)$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 3: Cannon's algorithm

Task	Execution time	STMMA
Shift A, B matrices	$4n^2\ t_f$	0
Send A, B matrices to the processors	$2n^2\ t_c$	$2mnp\ t_c$
Multiply the elements of A and B	$n^2\ t_f$	$n^2\ t_f$
Shift A, B matrices	$2(mn\ t_c + 2m^2\ n\ t_f)$	0
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(5m^2\ n + 5n^2) + t_c(2n^2 + 2mn)$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 4: Fox's algorithm with square decomposition

Task	Execution time	STMMA
Send B matrix	$n^2\ t_c$	$2mnp\ t_c$
Broadcast the diagonal elements of A	$mnp\ t_c$	0
Multiply A and B	$m^2n\ t_f$	$n^2\ t_f$
Shift A, B matrices	$mn\ t_c + 2m^2\ n\ t_f$	0
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(3m^2\ n + n^2) + t_c(2n^2 + mn(p+1)$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 5: Fox's algorithm with scattered decomposition

Task	Execution time	STMMA
Scatter A	$n^2\ t_c$	0
Broadcast the diagonal elements of B	$mnp\ t_c$	$2mnp\ t_c$
Multiply A and B	$m^2n\ t_f$	$n^2\ t_f$
Switch processors' A submatrix	$mn\ t_c$	0
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(m^2\ n + n^2) + t_c(2n^2 + 2m^2\ n + mnp)$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 6: PUMMA (MBD2)

Task	Execution time	STMMA
Scatter A	$m^2p\ t_c$	0
Broadcast the diagonal elements of B	npt_c	0
Multiply A and B	$m^2n\ t_f$	$n^2\ t_f$
Switch processors' A submatrix	m^2 root(p) t_c	0
Generate the resulting matrix	$n^2\ t_f + n2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(m\ 2n + n^2) + t_c(2n^2 + m^2root(p)(p+1))$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 7: SUMMA

Task	Execution time	STMMA
Broadcast A and B	$2mnp\ t_c$	$2mnp\ t_c$
Multiply A and B	$m^2n\ t_f$	$m^2n\ t_f$
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(m\ 2n + n^2) + t_c(n^2 + 2mnp)$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 8: DIMMA

Task	Execution time	STMMA
Broadcast A and B	$2mnp\ t_c$	$2mnp\ t_c$
Multiply A and B	$m^2n\ t_f$	$m^2n\ t_f$
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$	$2mn\ t_c$
Total execution time	$t_f(m\ 2n + n^2) + t_c(n^2 + 2mnp)$	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

Table 9: STMMA

Task	Execution time
Broadcast A and B	$2mnp\ t_c$
Multiply A and B	$m^2n\ t_f$
Generate the resulting matrix	$2mn\ t_c$
Total execution time	$2mn\ t_c\ (1 + p) + m^2n\ t_f$

The above algorithms being compared theoretically and experimentally on a paper titled "Performance Analysis and Evaluation of Parallel Matrix Multiplication Algorithms" by Ziad A.A. Alqadi, Musbah Aqel and Ibrahiem M. M. El Emary; World Applied Sciences Journal 5 (2): 211-214, 2008, ISSN 1818-4952,© IDOSI Publications, 2008.

The theoretical part will be compared by the theoretical part of STMMA algorithm.

Through the theoretical analysis, the following symbols will be used[1]:

- f = number of arithmetic operations units
- tf = time per arithmetic operation << tc (time for communication)
- c = number of communication units
- q = f / c average number of flops per communication access.
- Minimum possible time = f* tf when no communication.
- Efficiency(speedup) SP=q*(tf/tc)
- f * tf + c* tc = f * tf * (1 + tc/tf * 1/q)
- m2 = n2/p

Also, the algorithms to be compared concerning the load distribution, where all the algorithms above - except STMMA algorithm - suffer from non-equivalent load distribution between the processors in case of non square matrix multiplications were being carried out.

CONCLUSION

In this paper, we have introduced new parallel matrix multiplication algorithm which perform matrix multiplication of 5000×5000 faster 4 times than Cannon algorithm. STMMA has better performance in the means of:

- Time for exchanged message being minimized to minimum, as each task to be completed by the same processor. The exchanged messages limited to only collect the last results, where there is no time consuming for exchange intermediate results.
- Load balance, where no processor stays idle any time, another point being recorded here is all the processors do same number of operations.

[1]Theses definitions appeared in several literatures

- Processor efficiency where no processor will stay idle waiting other processor output, as each processor has complete single task at a time.

This algorithm is to be extended for solving more problems like solving Laplace equation using Jacobi operations and other problems in heat transferring and fluid dynamic, where more efforts are needed to develop a mechanism to define the independent tasks.

REFERENCES

1. Bailey, D.H., K. Lee and H.D. Simon, 1991. Using Strassen's Algorithm to Accelerate the Solution of Linear Systems, Journal of Supercomputing, 4(4): 357-371.
2. Mathur, K. and S.L. Johnsson, 0000. Multiplication of Matrices of Arbitrary Shape on a Data Parallel Computer, Parallel Computing, 20: 919-952.
3. Cannon, L., 1969. Ph.D. thesis, Montana State University, Roseman, MN.
4. Fox, G., S. Otto and A. Hey, 1987. Matrix algorithms on a hypercube I: matrix multiplication,' Parallel Computing, 3: 17-31.
5. Choi, J., J.J. Dongarra and D.W. Walker, 1994. PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers. Concurrency: Practice and Experience, 6: 543-570.
6. Van De Geijin, R. and J. Watts, 1995. SUMMA: Scalable Universal Matrix Multiplication Algorithm LAPACK Working Note 99, Technical Report CS-95-286, University of Tennessee.
7. Jaeyoung Choi, 0000. A New Parallel Matrix Multiplication Algorithm on Distributed - Memeory Concurrent Computers.
8. Ioannis Sotiropoulos and Ioannis Papaefstathiou, 2009. A fast parallel matrix multiplication reconfigurable unit utilized in face recognitions system, 978-1-4244-3892-1/09/ ©2009 IEEE.
9. Zhaoquan Cai and Wenhong Wei, 2008. General Parallel Matrix Multiplication on the OTIS Network, 978-0-7695-3122-9/08 © 2008 IEEE.
10. Yunfei Du, Panfeng Wang, Hongyi Fu, Jia Jia, Haifang Zhou and Xuejun Yang, 2007. Building single fault survivable parallel algorithms for matrix operations using redundant parallel computation 0-7695-2983-6/07 © 2007 IEEE.
11. Ardavan Pedram, Masoud Daneshtalab and Sied Mehdi Fakhraie, 2006. An Efficient Parallel Architecture for Matrix Computations, 1-4244-0772-9/06 ©2006 IEEE.
12. James Demmel, Mark Hoemmen, Marghoob Mohiyuddin and Katherine Yelick, 2008. Avoiding Communication in Sparse Matrix Computations, 978-1-4244-1694-3/08 ©2008 IEEE.
13. Duc Kien Nguyen, Ivan Lavall´ EE, Marc Bui and H.A. Quoc Trung, 2005. A General Scalable Implementation of Fast Matrix Multiplication Algorithms on Distributed Memory Computers, 0-7695-2294-7/05 © 2005 IEEE.
14. Michael Bader, Sebastian Hanigk and Thomas Huckle, 2007. Parallelisation of Block-Recursive Matrix Multiplication in Prefix Computations, John von Neumann Institute for Computing, J¨ulich, NIC Series, ISBN 978-3-9810843-4-4, 38: 175-184.
15. Agarwal, R.C., S.M. Bale, F.G. Gustavson, M. Joshi, P. Palkar, 1995. A threedimensional approach to parallel matrix multiplication Ibm J. Res. Develop., 39(5).
16. Manojkumar Krishnan and Jarek Nieplocha, 2004. SRUMMA: A Matrix Multiplication Algorithm Suitable for Clusters and Scalable Shared Memory Systems Proceedings of the 18th International Parallel and Distributed Processing Symposium (C) 2004 IEEE.
17. Jaeyoung Choi, 1997. A New Parallel Matrix Multiplication Algorithm on Distributed-Memory Concurrent Computers UT, CS-97-369.