World Applied Sciences Journal 5 (2): 211-214, 2008
ISSN 1818-4952

Performance Analysis and Evaluation of Parallel Matrix Multiplication Algorithms

[1]Ziad A.A. Alqadi, [2]Musbah Aqel, and [3]Ibrahiem M. M. El Emary

[1]Faculty of Engineering, Al Balqa Applied University
[2]Faculty of Engineering, Applied Science University
[3]Faculty of Engineering, Al Ahliyya Amman University Amman, Jordan

Abstract: Multiplication of large matrices requires a lot of computation time as its complexity is O (n^3). Because most current applications require higher computational throughputs with minimum time, many sequential and parallel algorithms are developed. In this paper, a theoretical analysis for the performance and evaluation of the parallel matrix multiplication algorithms is carried out. However, 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.

Key words: Parallel processing · Matrix multiplication algorithms and Distributed systems

INTRODUCTION

Matrix multiplication is commonly used in the areas of graph theory, numerical algorithms, digital control, digital image processing and signal processing. Multiplication of large matrices requires a lot of computation time as its complexity is O (n^3), 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. Even with such improvements [1], for example, Strassen's algorithm for sequential matrix multiplication [14] has shown a limitation in performance. For this reason, parallel approaches have been examined for decades.

Most of parallel matrix multiplication algorithms use matrix decomposition that is based on the number of processors available. This includes the systolic algorithm [11], Cannon's algorithm [5], Fox and Otto's algorithm [3], PUMMA (Parallel Universal Matrix Multiplication) [12], SUMMA (Scalable Universal Matrix Multiplication) [8] and DIMMA (Distribution Independent Matrix Multiplication) [9, 10]. Each one of these algorithms uses the matrices that decomposed into sub-matrices. During execution process, a processor calculates a partial result using the sub-matrices that are currently accessed by it. It successively performs the same calculation on new sub-matrices, adding the new results to the previous. When the multiplication sub process is completed, the root processor assembles the partial results and generates the complete matrix multiplication result.

In order to make a proper selection for the given multiplication operation and to decide which is the best suitable algorithm that generates a high throughput with minimum time, a comparison analysis and a performance evaluation for the above mentioned algorithms is carried out using the same performance parameters and based on parallel processing. Moreover, these algorithms are implemented using C++ programming language.

PARALLEL COMPUTING PARADIGMS

Parallel computing process depends on how the processors are connected to a memory. The way of system connection can be classified into a shared memory system or distributed memory, each of these two types is discussed as follows:-

Shared Memory System: In such a system, a single address space exists, within it every memory location is given a unique address and the data stored in memory are accessible to each processor. The P_i processor reads the data written by P processor. Therefore, in order to enforce sequential consistency, it is necessary to use synchronization.

The Open MP is one of the popular programming languages executed on the shared memory system. It provides a portable, scalable and efficient approach to run parallel programs in C/C++ and FORTRAN [16, 17]. In OpenMP, a sequential programming language can be

Corresponding Author: Ziad A.A. Alqadi, Faculty of Engineering, Al Balqa Applied University

Table 1-A: Systolic algorithm [1]

Algorithm	Execution time
Transpose B matrix	$2n^2\ t_{fl}$
Send A, B matrices to the processors	$2m^2p\ t_{cm}$
Multiply the elements of A and B	$m^2n\ t_f$
Switch processors' B sub-matrix	$n^2\ t_c$
Generate the resulting matrix	$n^2\ t_f + n^2\ t_c$
Total execution time	$t_f(m^2n + 3n^2) + t_c(4n^2)$

Table 1-B: Cannon's algorithm [2]

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

Table 1-C: Fox's algorithm with square decomposition [3]

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

Table 1-D: Fox's algorithm with scattered decomposition [4]

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

Table 1-E: PUMMA (MBD2) [5]

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

Table 1-F: SUMMA [6]

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

Table 1-G: DIMMA [7]

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

parallelized with preprocessor compiler directives such as #pragma omp in C and $OMP in FORTRAN based with library support.

Distributed Memory System: In such a system, each processor has its own memory and can only access its local memory. The processors are connected with other processors via a high-speed communication network. Processors exchanges information with one another using send and receive operations. A common approach to programming multiprocessors is to use message-passing library routines in addition to conventional sequential program. [18, 19]

MPI (Message Passing Interface) is useful for a distributed memory systems since it provides a widely used standard of message passing program. It provides a practical, portable, efficient and flexible standard for message passing [13, 15]. In MPI, data is distributed among processors, where no data is shared and data is communicated by message passing.

Performance Evaluation: The MPI technique needs two kinds of time to complete the multiplication process, t_c and t_f. Where t_c represents the time it takes to communicate one datum between processors and t_f is the time needed to multiply or add elements of two matrices. It is assumed that $n \times n$ matrices are multiplied on a number of processors (p). Each processor holds the n^2/p elements and it was assumed that n^2/p is set to a new variable m^2. A summary of the execution time of every step of each algorithm [12, 15 and 17] is shown in Table (1).

THEORETICAL ANALYSIS

In order to evaluate the performance of any matrix multiplication based on using parallel processors and different algorithms, a theoretical analysis is carried out based on the following assumptions:

- 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)

Table 2: Algorithm analytical variables

Algorithm	f	C	q
(1)	$(m^2n + 3n^{2)}$	$(4n^{2)}$	$(m^2n + 3n^{2)} / (4n^{2)}$
(2)	$(5m^2n + 5n^2)$	$(2n^2 + 2mn)$	$(5m^2n + 5n^2) / (2n^2 + 2mn)$
(3)	$(3m^2n + n^2)$	$(2n^2 + mn(p+1)$	$(3m^2n + n^2) / (2n^2 + mn(p+1)$
(4)	$(m^2n + n^2)$	$(2n^2 + 2m^2n + mnp)$	$(m^2n + n^2) / (2n^2 + 2m^2n + mnp)$
(5)	$(m^2n + n^2)$	$(2n^2 + m^2$ root(p)(p+1)	$(m^2n + n^2) / (2n^2 + m^2$ root(p)(p+1)
(6)	$(m^2n + n^2)$	$(n^2 + 2mnp)$	$(m^2n + n^2) / (n^2 + 2mnp)$
(7)	$(m^2n + n^{2)}$	$(n^2 + 2mnp)$	$(m^2n + n^{2)} / (n^2 + 2mnp)$

However, to apply these analytical variables, some values are assumed as follows: Number of processors (p) = 4, Number of elements (n) = 600, (tf/tc) = 0.1. So, the following results are obtained for all algorithms as shown in table (3)

Table 3: Theoretical results

Algorithm	f	C	Q	SP
(1)	55080000	1440000	38.5	3.85
(2)	271800000	108720000	2.5	0.25
(3)	162360000	270720000	0.599	0.0599
(4)	54360000	109440000	0.4967	0.0497
(5)	54360000	1620000	33.555	3.355
(6)	54360000	1800000	30.2	3.02
(7)	54360000	1800000	30.2	3.02

Table 4: 600 * 600 matrix multiplication

No. of processors	systolic	cannon	fox	fox2	pumma	Summa	Dimma
1	251.69	530.33	390.2	506.49	271.12	149.54	165..97
4	65.5	408.12	236.32	267.244	79.583	45.241	49.39

From the results in Table (4), the performance factors (i.e. speedup and efficiency) for each algorithm can be calculated and the results are shown in table (5), taking into consideration that: Speedup (SP) = time using (1) processor/time and when using (n) processors=T (1)/T (P). Then, Efficiency (E) = SP/P (must be closed to 1)

Table 5: Efficiency factors

Algorithm	SP	E
systolic	3.843	0.961
cannon	1.299	0.325
Fox	1.651	0.413
fox2	1.895	0.474
pumma	3.406	0.852
summa	3.305	0.826
dimma	3.36	0.84

Again, the results in table (4) prove our theoretical conclusion. From the results in Table (5), it could be concluded that systolic algorithm will give the best performance (i.e. efficiency), then Pumma algorithm, then dimma and summa algorithm

So, larger q indicates that time is closer to minimum f*tf, thus increasing the speed up and the efficiency of the algorithm. SP must be closer to number of processors to achieve the highest efficiency. Using the above assumption and with reference to the information in Table(1) we can obtain f, c and q for each parallel matrix multiplication as shown in Table(2).

EXPERIMENTAL RESULTS

Table (4) shows the experimental results obtained by implementing each algorithm 100 times and Then, ten fastest five ones are taken and averaged

CONCLUSION AND FUTURE WORKS

A theoretical analysis for the performance of most seven used algorithms is carried out using one and four processors. 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 summa. However, these algorithms are implemented and run on one and four processors to evaluate their performance for a matrix multiplication. The experimental results are matched with the theoretical one. This analysis is useful for making a proper recommendation to select the best algorithm among others as a future works to be done by others.

REFERENCES

1. Ziad Alqadi and Amjad Abu-Jazzar, 2005. Analysis of program methods used for optimizing matrix multiplication, J. Eng., Vol. 15, NO. 1: 73-78.
2. Agarwal, R.C., F. Gustavson and M. Zubair, 1994. A high-performance matrix multiplication algorithm on a distributed memory parallel computer using overlapped communication, IBM J. Res. Develop., Volume 38, Number 6.
3. Agarwal, R.C., S.M. Balle, F.G. Gustavson, M. Joshi and P. Palkar, 1995. A 3-Dimensional Approach to Parallel Matrix Multiplication, IBM J.Res. Develop., Volume 39, Number 5, pp: 1-8, Sept.
4. Alpatov, P., G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, Robert van de Geijn and J. Wu, Plapack: Parallel Linear Algebra Package-Design Overview, Proceedings of SC 97, to appear.
5. Alpatov, P., G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, Robert van de Geijn and J. Wu, 1997. Plapack: Parallel Linear Algebra Package, Proceedings of the SIAM Parallel Processing Conference.
6. Anderson, E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen, 1990 Lapack: A Portable Linear Algebra Library for High Performance Computers, Proceedings of Supercomputing '90, IEEE Press, pp: 1-10.
7. Barnett, M., S. Gupta, D. Payne, L. Shuler, R. van de Geijn and J. Watts, 1994. Interprocessor Collective Communication Library (InterCom), Scalable High Performance Computing Conference.
8. Cannon, L.E., 1969. A Cellular Computer to Implement the Kalman Filter Algorithm, Ph.D. Thesis Montana State University.
9. Chtchelkanova, A., J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, 1995. Parallel Implementation of BLAS: General Techniques for Level 3 BLAS, TR-95-40, Department of Computer Sciences, University of Texas, Oct.
10. Choi J., J.J. Dongarra, R. Pozo and D.W. Walker, 1992. Scalapack: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers, Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation. IEEE Comput. Soc. Press, pp: 120-127.
11. Choi, J., J.J. Dongarra and D.W. Walker, Level 3 BLAS for distributed memory concurrent computers, 1992. CNRS-NSF Workshop on Environments and Tools for Parallel Scientific Computing, Saint Hilaire du Touvet, France, Sept. 7-8, Elsevier Sci. Publishers,
12. 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, Vol 6(7): 543-570.
13. Dongarra, J.J., J. Du Croz, S. Hammarling and I. Duff, 1990. A Set of Level 3 Basic Linear Algebra Subprograms, TOMS, Vol. 16, No. 1, pp: 1-16.
14. Dongarra, J.J., R.A. Van de Geijn and D.W. Walker, 1994. Scalability Issues Affecting the Design of a Dense Linear Algebra Library, J. Parallel and Distributed Computing, Vol. 22, No. 3, Sept., pp: 523-537.
15. Edwards, C., P. Geng, A. Patra and R. vande Geijn, 1995. Parallel matrix distributions: have we been doing it all wrong?, Tech. Report TR-95-40, Dept of Computer Sciences, The University of Texas at Austin.
16. Fox, G.C., M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.K. Salmon and D.W. Walker, 1988. Solving Problems on Concurrent Processors, Vol. 1, Prentice Hall, Englewood Cliffs, N.J.
17. Fox, G., S. Otto and A. Hey, 1987. Matrix algorithms on a hypercube I: matrix multiplication, Parallel Computing 3, pp: 17-31.
18. Gropp, W., E. Lusk and A. Skjellum, 1994. Using MPI: Portable Programming with the Message-Passing Interface, The MIT Press.
19. C.-T. Ho and S.L. Johnsson, 1986. Distributed Routing Algorithms for Broadcasting and Personalized Communication in Hypercubes, In Proceedings of the 1986 International Conference on Parallel Processing, pages 640-648, IEEE.