Review Article

A Survey on Serial and Parallel Optimization Techniques Applicable for Matrix Multiplication Algorithm

Yajnaseni Dash*, Sanjay Kumar and V.K. Patle

School of Studies in Computer Science & IT, Pt. Ravishankar Shukla University, Raipur, Chhattisgarh, 492010, India

Address for Correspondence

School of Studies in Computer Science & IT, Pt. Ravishankar Shukla University, Raipur, Chhattisgarh, 492010, India.
E-mail: yajnasenidash@gmail.com

ABSTRACT

Parallel algorithms play an imperative role in the high performance computing environment. Dividing a task into the smaller tasks and assigning them to different processors for parallel execution are the two key concepts to evaluate the performance of parallel algorithms. Performance enhancement is essential in large scientific applications, where dense matrix multiplication algorithm is extensively used. Thus, optimization of this algorithm, both for serial and parallel execution would provide upsurge performance. In this paper, a brief systematic survey on serial and parallel optimization techniques applied on matrix multiplication algorithm is carried out.

Keywords: Parallel algorithm, Parallel processing, Matrix multiplication algorithm, Serial optimization, Parallel optimization.

INTRODUCTION

Programming on multiprocessor system using divide and conquer technique is called parallel programming. The parallel program is composed of various active processes all at once solving a particular problem. This paper focuses upon the previous research work which has been done to optimize the matrix multiplication algorithm on serial and parallel platforms. This paper focuses on different aspects of optimizing the matrix multiplication algorithm.

What is parallel algorithm?

An idealized parallel algorithm is that which is written for Parallel Random Access Machines (PRAM) model with no communication overhead[1].

Conventional uniprocessor computer system has been modeled as Random Access Machines (RAM) by Sheperdson and Sturgis in 1963[2] whereas parallel computers with zero synchronization and no memory access overhead have been modeled as PRAM in 1978[3]. If there is a set of k concurrent processes and if k=1 then it is called sequential algorithm. Sequential algorithm runs on uniprocessor machine. If

there is a set of k concurrent processes and if k>1 then it is called parallel algorithm. Parallel algorithm runs on parallel computers[1].

Matrix multiplication algorithm

Matrix is an extremely significant mean in conveying and discussing problems which arise from real life scenarios. It will be effortless to manipulate and obtain more information by managing the data in matrix form. Multiplication is one of the essential operations on matrices. A square matrix of order n x n is an arrangement of set of elements in n rows and n columns[4].

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} \times B_{kj}, \; 0 \leq i < m, \; 0 \leq j < 1 \ldots \text{(Eq.1)}$$

If we multiply the matrix *A* of the dimension *m x n* by the matrix *B* with size *n x l*, we store the result in matrix C (C= A x B) with dimension *m x l* with each element defined according to the expression (Eq. 1).

Literature survey of matrix multiplication algorithm on serial and parallel platform

Parallel matrix multiplication has been explored and investigated extensively in the previous two decades. There are diverse approaches to optimize the matrix multiplication algorithm. In this section, a brief overview on existing matrix multiplication algorithm is carried out.

Serial matrix multiplication optimization

Matrix multiplication is an exceptionally imperative essence in several numerical linear algebra algorithms and is one of the most studied problems in high-performance computing.

Several approaches have been proposed to optimize matrix multiplication by improving spatial and temporal locality. Blocking or tiling is one such fundamental technique[5]. Regardless of its generalization, blocking is architecture dependent whereas cache oblivious algorithms[6] are an architecture independent substitute to the blocked algorithms. The divide and conquer paradigm is used by cache oblivious algorithms. Authors in[7] made a comparison between cache oblivious and cache conscious algorithm. They found that even highly optimized cache oblivious programs perform significantly slower than cache conscious counterparts based on blocking. Chatterjee *et al.* bestowed a proposal regarding recursive array layouts and fast matrix multiplication. According to them cache oblivious method is to utilizing a recursive structure for the matrices[8]. Conversely, conventional implementations of the Basic Linear Algebra Subroutines (BLAS) libraries[9-16] are primarily based on the blocking approach and hence require optimization on a particular hardware platform. The BLAS routines provide standard building blocks to perform basic vector and matrix operations. As the BLAS are proficient, convenient, and extensively accessible, they are generally used in the development of high quality linear algebra software e.g. LAPACK.

Linear Algebra Package (LAPACK) is a standard software library which provides routines to solve systems of linear equations, linear least squares, problems of eigen value and singular value decomposition (SVD)[17]. Consequently, automatic optimization of matrix multiplication on different platforms has been a dynamic area of research. One such instance is Automatically Tuned Linear Algebra Software (ATLAS)[18] which provides C and Fortran77 interfaces to a portably efficient BLAS implementation. ATLAS automatically produces optimized numerical software for a given processor architecture as a part of the software installation process [19]. Another high performance implementation of matrix multiplication for a variety of architectures was presented in the GotoBLAS library[20]. In

table 1 serial matrix multiplication optimization methods are presented.

Parallel matrix multiplication optimization

Parallel matrix multiplication has also been systematically explored over the past three decades. Therefore, several parallel matrix multiplication algorithms have been proposed for distributed memory, shared memory and hybrid platforms. Here, only the algorithms designed for distributed memory platforms are focused.

Different layouts such as 1D Layout and 2D Layout were used for the optimization purpose. 1D Layout[21] was found to be much slower than serial. Here firstly, a bus connected machine without broadcast was considered and only one pair of processors can communicate at a time (ethernet). Secondly, a machine with processors on a ring was considered and all processors may communicate with nearest neighbors simultaneously. The efficiency of nearest neighbor communication on a ring (or bus with broadcast) is 1/(1 + O (p/n)). In 2D Layout processors are considered in 2D grid (physical or logical) can communicate with 4 nearest neighbors which broadcast along rows and columns. 2D Layout includes canon's matrix, scalable universal matrix multiply and recursive layouts. Cannon[22] introduced the first efficient distributed algorithm for two-dimensional meshes parallel matrix multiplication providing theoretically optimal communication cost in 1969. This algorithm is suitable for homogenous 2D grids but its extension is difficult to heterogeneous 2D grids. Constant storage requirements and independency of number of processors are the major advantage of the algorithm[23]. However cannon's matrix is hard to generalize with efficiency of 1/(1+O (sqrt (p)/n)). Fox's algorithm[24] was extended in PUMMA (Parallel Universal Matrix Multiplication Algorithm)[25] and Broadcast-Multiply-Roll (BiMMeR)[26] for a common 2D processor grid by using block cyclic data distribution and torus wrap data layout respectively. The 3D algorithm[27] prepares the p processors as $p^{1/3}\times p^{1/3} \times p^{1/3}$ 3D mesh and accomplishes a factor of $p^{1/6}$ less communication cost than 2D parallel matrix multiplication algorithms. But 3D algorithm need $p^{1/3}$ extra copies of the matrices which would be a significant problem on some platforms. For instance, on one million cores, the 3D algorithm will require 100 extra copies of the matrices. Hence, this algorithm is only convenient to relatively smaller matrices. Agarwal *et al.*[28] in 1994 proposed a different method for improving the performance of parallel matrix multiplication by overlapping communication and computation. The Scalable Universal Matrix Multiplication Algorithm (SUMMA)[29] is a very useful algorithm which needs less workspace and overcomes the necessity of a square 2D grid. It is slightly less efficient, but simpler and easier to generalize with efficiency 1/(1 + O (log p * p/(b*n2) + log p * sqrt (p)/n)). The smaller value of b produces less memory and has lower efficiency whereas the larger value of b produces more memory and has higher efficiency. It uses Scalable Linear Algebra Library for Distributed Memory Concurrent Computers (ScaLAPACK)[30], which is one of the most conventional parallel numerical linear algebra packages. DIMMA (Distribution Independent Matrix Multiplication Algorithm)[31] is related to SUMMA but uses a different pipelined communication scheme for overlapping communication and computation. A novel matrix multiplication algorithm suitable for clusters and scalable shared memory systems (SRUMMA)[32] has equivalent algorithmic efficiency with Cannon's algorithm on clusters and shared memory systems. It uses block checkerboard distribution of the matrices and overlaps communication with computations by using remote memory access (RMA) communication rather than message passing. Authors presented 2.5D algorithm[33]

to generalize the 3D algorithm by parameterizes the extent of the third dimension of the processor arrangement: $\frac{p^{1/2}}{c} \times \frac{p^{1/2}}{c} \times c$, $c \in [1, p^{1/3}]$. Simultaneously, it is predictable that exascale systems will have a considerably shrinking memory space per core[34]. Therefore, the 2.5D algorithm cannot be scalable on future exascale systems. Ali *et al*.[35] proposed the performance analysis of the matrix multiplication algorithms through Message passing Interface (MPI). In a previous study, matrix multiplication problem has also been studied to recognize the effect of problem size on parallelism. But this study was limited to a single multicore processor only and that was too implemented in Open Multi-Processing (OMP) environment[36]. In[37] a hierarchical optimization was presented to improve the communication cost and the overall performance on large-scale platforms. They applied their approach on SUMMA for optimization of message-passing parallel algorithms for execution on large scale distributed memory systems. Authors[38] analyzed the impact of different block size (M.K and K.N) on the performance. They used various parameter values for K and predefined values of the parameters M and N, for testing algorithm behavior in different cache regions. In[39] a technique was implemented to improve parallel execution of auto generated OpenMP programs by considering architecture of on chip cache memory. Several studies[40-42] were carried out to evaluate the performance of matrix multiplication algorithm on multicore processors by using OMP. They found that the parallel algorithms with small data set perform worse than sequential algorithms. However as the size of the data set increases the execution of parallel algorithms bestows the best outcome than sequential execution.

CONCLUSION

Optimization techniques can speed up the multi-core parallel execution by reducing the number of memory accesses, using the features of granularity and scalability of the algorithm and improving the algorithm appropriate to hardware architecture and organization. In this paper an extensive survey of published research work is carried out which will be beneficial for the researchers to get the deep insight into this topic for further analysis and exploration.

REFERENCES

1. Hwang K, Jotwani N (2001), "Advanced Computer Architecture", Tata McGraw Hill education Private Limited, Second Edition.
2. Shepherdson J. C., Sturgis H. E (1963), Computability of Recursive Functions. *J. ACM* 10, 2 (April 1963), 217-255. DOI=10.1145/321160.321170
3. Fortune S, James Wyllie (1978, Parallelism in random access machines. In Proceedings of the tenth annual ACM symposium on Theory of computing (STOC '78). ACM, New York, NY, USA, pp. 114-118.
4. Horn, Johnson (2013), Matrix Analysis (2nd ed.), Cambridge University Press.
5. Gustavson FG (2012), Cache blocking for linear algebra algorithms. Parallel processing and applied mathematics. In: Lecture Notes in Computer Science, vol 7203, Springer, Berlin, pp. 122-132.
6. Frigo M, Leiserson CE, Prokop H, Ramachandran S (1999), Cache-oblivious algorithms. In: Proceedings of the 40th annual symposium on foundations of computer science, FOCS '99. *IEEE Computer Society*, Washington, DC, USA, pp. 285.
7. Yotov K, Roeder T, Pingali K, Gunnels J, Gustavson F (2007), An experimental comparison of cache-oblivious and cache-conscious programs. In: Proceedings of the nineteenth annual ACM symposium on parallel algorithms and architectures, SPAA' 07ACM, New York, NY, USA, pp. 93-104.
8. Chatterjee S, Lebeck AR, Patnala PK, Mithuna T (2002), Recursive array layouts

and fast matrix multiplication. IEEE Trans Parallel Distrib Syst 13(11), pp. 1105-1123.
9. Basic Linear Algebra Routines (BLAS), Available online at: http://www.netlib.org/blas/.
10. Lawson C. L., Hanson R. J., Kincaid D., and Krogh F. T., Basic Linear Algebra Subprograms for *FORTRAN usage, ACM Trans*. Math. Soft., 5 (1979), pp. 308-323.
11. Dongarra J. J, J. Du Croz, S. Hammarling, and R. J. Hanson, An extended set of FORTRAN Basic Linear Algebra Subprograms, *ACM Trans*. Math. Soft., 14 (1988), pp. 1-17.
12. Dongarra J. J., Croz JD, Hammarling S., Hanson R. J., Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subprograms, *ACM Trans*. Math. Soft., 14 (1988), pp. 18-32.
13. Dongarra J. J., Croz JD, Duff I. S., and Hammarling S., A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans*. Math. Soft., 16 (1990), pp. 1-17.
14. Dongarra J. J., Croz JD, Duff I. S., Hammarling S., Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans*. Math. Soft., 16 (1990), pp. 18-28.
15. 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., Whaley R. C., An Updated Set of Basic Linear Algebra Subprograms (BLAS), *ACM Trans*. Math. Soft., 28-2 (2002), pp. 135—151.
16. Dongarra J., Basic Linear Algebra Subprograms Technical Forum Standard, *International Journal of High Performance Applications and Supercomputing*, 16(1) (2002), pp. 1-111, and International Journal of High Performance Applications and Supercomputing, 16(2) (2002), pp. 115-199.
17. http://www.netlib.org/lapack/lapacke.html
18. http://math-atlas.sourceforge.net/
19. Clint WR, Dongarra JJ (1998), Automatically tuned linear algebra software. Proceedings of the 1998 ACM/IEEE conference on supercomputing. Supercomputing'98 *IEEE Computer Society*, Washington, DC, USA, pp. 1-27.
20. Goto K, van De Geijn RA (2008), Anatomy of high-performance matrix multiplication. *ACM Trans* Math Softw 34(3), pp.1-25.
21. Dekel E., D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms", *SIAM Journal on Computing*", vol.10, 1981, pp. 657-673
22. Lynn Elliot Cannon, A cellular computer to implement the Kalman Filter Algorithm, Technical report, Ph.D. Thesis, Montana State University, 14 July 1969.
23. Gupta, H.; Sadayappan, P.: Communication Efficient Matrix-Multiplication on Hypercubes, dbpubs.stanford.edu.
24. Fox GC, Otto SW, Hey AJG (1987), Matrix algorithms on a hypercube I: matrix multiplication. *Parallel Comput* 4(1), pp.17-31.
25. Choi J, Walker DW, Dongarra J (1994), PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurr Pract Exp* 6(7), pp.543-570.
26. Huss-Lederman S, Jacobson E, Tsao A, Zhang G (1994), Matrix multiplication on the Intel Touchstone Delta. *Concurr Pract Exp* 6(7), pp.571-594.
27. Agarwal RC, Balle SM, Gustavson FG, Joshi M, Palkar P (1995), A three-dimensional approach to parallel matrix multiplication. *IBM J Res Dev* 39(5), pp. 575–582.
28. Agarwal RC, Gustavson FG, ZubairM (1994), A High-performance matrix-multiplication algorithm on a distributed-memory parallel computer, using overlapped communication. *IBM J Res Dev* 38(6), pp. 673–681.
29. van de Geijn RA, Jerrell W (1997), SUMMA: scalable universal matrix multiplication algorithm. *Concurr Pract Exp* 9(4), pp.255–274.
30. Blackford LS, Choi J, Cleary A, D'Azeuedo E, Demmel J, Dhillon I, Hammarling S, Henry G, Petitet A, Stanley K, Walker D, Whaley RC (1997), ScaLAPACK user's guide. Society for industrial and applied mathematics, Philadelphia.
31. Choi J (1997), A new parallel matrix multiplication algorithm on distributed-memory concurrent computers. In: High Performance Computing on the Information Superhighway, HPC, Asia '97, pp. 224–229.

32. KrishnanM, Nieplocha J (2004), SRUMMA: a matrix multiplication algorithm suitable for clusters and scalable shared memory systems. In: Proceedings of parallel and distributed processing symposium.
33. Solomonik E, Demmel J (2011), Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In: Euro-Par (2), Lecture Notes in Computer Science, vol 6853. Springer, Berlin, pp 90-109.
34. U.S.Department of Energy (2011): "Exascale Programming Challenges". ASCR Exascale Programming Challenges Workshop.
35. Ali J, Khan RZ, (2012), Performance Analysis of Matrix Multiplication Algorithms Using MPI, *International Journal of Computer Science and Information Technologies* (IJCSIT), Vol. 3 (1), pp. 3103 - 3106.
36. Patel R, Kumar S (2012), "Effect of problem size on parallelism", Proc. of 2nd International conference on Biomedical Engineering & Assistive Technologies at NIT Jalandhar, pp. 418-420.
37. Hasanov K, Quintin JN, Lastovetsky A (2014), "Hierarchical approach to optimization of parallel matrix multiplication on large-scale platforms" , *J Supercomputing, Springer*, 2014.
38. Ristov S., Gusev M., Velkoski G. (2014), Optimal Block Size for Matrix Multiplication Using Blocking, MIPRO 2014, 26-30 May 2014, Opatija, Croatia, pp. 295-300.
39. Dheeraj D, Nitish, B, Ramesh S, (2012), "Optimization of Automatic Conversion of Serial C to Parallel OpenMP," Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2012 International Conference, pp.309-314.
40. Sharma SK, Gupta K (2012), "Performance Analysis of Parallel Algorithms on Multi-core System using OpenMP Programming Approaches", *International Journal of Computer Science, Engineering and Information Technology* (IJCSEIT), Vol.2, No.5.
41. Kathavate S, Srinath N.K. (2014), "Efficiency of Parallel Algorithms on Multi Core Systems Using OpenMP," *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 3, Issue 10, October 2014. pp. 8237-8241.
42. Kulkarni P, Pathare S, "Performance analysis of parallel algorithm over sequential using OpenMP," *IOSR Journal of Computer Engineering* (IOSR-JCE), Volume 16, Issue 2, pp. 58-62.

Table 1. Studies on serial matrix multiplication optimization

Author and Year	Reference	Method Used	Description
Lawson *et al.* (1979), Dongarra *et al.* (1988, 1990, 2002) and Blackford *et al.* (2002)	[9-16]	BLAS	*Level 1:* Perform scalar, vector and vector-vector operations. *Level 2:* Perform matrix-vector operations. It is very efficient on vector computers, but not suitable to computers with a hierarchy of memory (i.e., cache memory). *Level 3:* Used for matrix-matrix operations.
Clint and Dongarra (1998)	[18-19]	ATLAS	ATLAS can be used by researchers requiring fast linear algebra routines as it provides optimized libraries.
Frigo *et al.* (1999)	[6]	Cache oblivious algorithms	Cache oblivious algorithms are not depending on the architecture.
Chatterjee *et al.*(2002)	[8]	Recursive Array Layouts	Authors observed a basic qualitative difference between the standard algorithm and the fast ones in terms of the benefits of using recursive layouts.
Yotov *et al.* (2007)	[7]	Cache oblivious and cache conscious algorithms	Compared both cache oblivious and cache conscious programs experimentally and found that cache conscious programs perform better than cache oblivious programs based on blocking.
Goto and van De Geijn (2008)	[20]	GotoBLAS	Analysis of high performance matrix multiplication in GotoBLAS library.
Gustavson (2012)	[5]	Blocking	Blocking is dependent on the architecture.