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

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

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

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 are an architecture independent substitute to the blocked algorithms. The divide and conquer paradigm is used by cache oblivious algorithms. Authors in 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. Conversely, conventional implementations of the Basic Linear Algebra Subroutines (BLAS) libraries 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). 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) 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. Another high performance implementation of matrix multiplication for a variety of architectures was presented in the GotoBLAS library.
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 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 cannon’s matrix, scalable universal matrix multiply and recursive layouts. Cannon 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. However cannon’s matrix is hard to generalize with efficiency of $1/(1+O(\sqrt{p/n}))$. Fox’s algorithm was extended in PUMMA (Parallel Universal Matrix Multiplication Algorithm) and Broadcast-Multiply-Roll (BiMMeR) for a common 2D processor grid by using block cyclic data distribution and torus wrap data layout respectively. The 3D algorithm 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. 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) 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 $1/(1 + O(\log p * p/(b*n^2) + \log p * \sqrt{p/n}))$. 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), which is one of the most conventional parallel numerical linear algebra packages. DIMMA (Distribution Independent Matrix Multiplication Algorithm) 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) 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.
to generalize the 3D algorithm by parameterizes the extent of the third dimension of the processor arrangement: \( \mathbb{E}^{1/2} \times \mathbb{E}^{1/2} \times c, c \in [1, p^{1/3}] \). 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**

17. http://www.netlib.org/lapack/lapacke.html


Table 1. Studies on serial matrix multiplication optimization

<table>
<thead>
<tr>
<th>Author and Year</th>
<th>Reference</th>
<th>Method Used</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clint and Dongarra (1998)</td>
<td>[18-19]</td>
<td>ATLAS</td>
<td>ATLAS can be used by researchers requiring fast linear algebra routines as it provides optimized libraries.</td>
</tr>
<tr>
<td>Frigo et al. (1999)</td>
<td>[6]</td>
<td>Cache oblivious algorithms</td>
<td>Cache oblivious algorithms are not depending on the architecture.</td>
</tr>
<tr>
<td>Chatterjee et al. (2002)</td>
<td>[8]</td>
<td>Recursive Array Layouts</td>
<td>Authors observed a basic qualitative difference between the standard algorithm and the fast ones in terms of the benefits of using recursive layouts.</td>
</tr>
<tr>
<td>Yotov et al. (2007)</td>
<td>[7]</td>
<td>Cache oblivious and cache conscious algorithms</td>
<td>Compared both cache oblivious and cache conscious programs experimentally and found that cache conscious programs perform better than cache oblivious programs based on blocking.</td>
</tr>
</tbody>
</table>