EFFECT OF MEMORY ON THE PERFORMANCE OF ONE-STEP RECURSION PARALLEL STRASSEN'S ALGORITHM ON DUAL CORE PROCESSOR

A. J. KAWU[1*], A. U. YAHAYA[2] AND SAMINU I. BALA.[3]

[1,2]Department of Computer Science, Gombe State University, Gombe, Nigeria
[3]Department of Mathematical Science, Bayero University, Kano, Nigeria
Corresponding Author: ahmadkawujibir@gmail.com

ABSTRACT

The main aim of this research work is to compare the speedup obtainable from executing One-Level Recursion parallel Strassen's matrix multiplication algorithm on a dual core with 2G and 4G memories relative to parallel conventional algorithm. This is to determine the effect of additional memory on the performance of the algorithm. The speedup obtainable by Strassen's Algorithm is at the expense of additional memory space for intermediate matrices. Even though additional memory improve performance, there is need to experimentally investigate how additional memory influence the performance of the new algorithm. The execution time of parallel implementations of the algorithms on dual core processor with 2G and 4G memories were measured, analyzed and interpreted. The programs were written in C++/OpenMP. The result of the research work showed a minor improvement in performance of the parallel One-Level Recursion Strassen's algorithm for large matrix size on the 4G memory. It was also observed that conventional algorithm did not show any significant performance improvement with increased memory. The result suggested that with large amount of memory One-Level Recursion, Strassen's algorithm performs a little better for larger matrices. The result will serve as a motivation for further research to determine the role played by cache memory in performance of the new algorithm.

Keywords: Strassen's, OpenMP, Speedup, One-Level Recursion.

INTRODUCTION

Matrix multiplication has wide application in linear algebra and it is one of the most studied problems in high performance computing. Matrix multiplication has numerous applications in engineering, image processing, computer graphics (Aliyu et al., 2019), graph algorithms, combinatorial optimization, machine learning, digital control (Singh, Chander and Bhatt 2019), Sonar systems, Relational Database Management Systems (Rathore and Kumar, 2014).

Thottethodi et al. (1998) stated that Strassen's algorithm gained its lower arithmetic complexity at the expense of locality of reference which makes it difficult to implement on modern machines with hierarchical memory. This is mainly caused by the intermediate matrices created by a deep level of recursion of the Strassen algorithm. They suggested early truncation of the recursion to reduce the amount of temporary storage requirement. Tang (2019) also stated that there is need to balance space-time requirement for an algorithm to achieve higher performance on modern shared memory multicore architectures. Strassen algorithm has the expense of memory allocation due to recursive overhead for creating additional storage of the required sub matrices (Zin Oo, and Chaikan, 2019).

Conventional Matrix Multiplication

Multiplying two square matrices of size n x n using the Naïve (conventional) matrix multiplication cost n^3 multiplications as shown in equation (1).

If $A=[a_{i,j}]$ is an $m \ x \ n$ matrix and $B=[b_{j,k}]$ is an n x p matrix, then the matrix product $C=AB$ is the $m \ x \ p$ matrix $C=[c_{i,k}]$ defined by

$$c_{i,k}=\sum_{j=1}^{n} a_{i,j}b_{j,k} \quad 1 \le i \le m, 1 \le k \le p \ . \qquad (1)$$

The definition of the matrix multiplication operation is very simple, all of which simplifies its understanding. Given a matrix $A^{(m \ x \ r)}$ of m rows and r columns, where each element is denoted as $a_{i,j}$ with $1 \le i \le m, \ and \ 1 \le j \le r$; and a matrix $B^{(r \ x \ n)}$ of r rows and n columns, where each element is denoted as $b_{i,j}$ with $1 \le i \le r \ and \ 1 \le j \le n$; matrix C resulting from the multiplication operation of A and B matrices $C = A \ x \ B$, is such that each of its element is denoted as $c_{i,j} \ with 1 \le i \le m, \ and 1 \le j \le n$ and it is computed as

$$C_{i,j}=\sum_{k=1}^{r} a_{i,k} x \ b_{k,j} \qquad (2)$$

This multiplication has the asymptotic cost of $O\ (n^3)$

Developing ideas and algorithms to reduce this cost has been one of the central problems in algebraic complexity since the 1960s. Strassen's discovery in 1969 of an algorithm for multiplying 2 x 2 matrices using 7 (instead of 8) multiplication was the key event in the history of fast matrix multiplication algorithms. According to Ikenmeyer and Lysikov (2019), the discovery of Strassen's matrix multiplication algorithm is a great achievement in computational linear algebra. Several variants of Strassen's algorithms such as Winograd's, Laderman's, Hopcoft and Kerr's were developed to reduce the complexity cost of the conventional matrix multiplication method. According to Dammel et al. (2012) matrix multiplication is a core kernel of high-performance computing. It is an inherently parallelizable task, and algorithms which take advantage of parallel architectures to achieve much higher performance than serial implementations. The fundamental role of matrix multiplication is evidenced by its inclusion in portable libraries such as level 3 Basic Linear Algebra Subroutine (BLAS) which can then be used as a building block in building routines such as Linear Algebra

Package (LAPACK), Automatically Tuned Linear Algebra Software (ATLAS), and so on. . According to Gu et ‘al. (2020) improving matrix multiplication is still an active research area.

Recent developments in multi core processors have led to renewed interest in parallelization of matrix multiplication algorithm (Rathore and Kumar (2014), Kathavate and Srinath (2014), Dash, Kumar and Patle (2015), Sharma and Munir (2015) and Thakur and Kumar (2016).

Strassen’s Matrix Multiplication

Strassen’s algorithm is an improvement over the conventional algorithm in the case of multiplying 2 x 2 matrices, because it uses only seven scalar multiplications as opposed to the usual eight. This algorithm has the asymptotic complexity of $O\ (n^{\log_2 7})$ The algorithm is defined using Divide and Conquer as follows:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} x \begin{bmatrix} e & f \\ g & h \end{bmatrix} \tag{3}$$

$$= \left| \begin{matrix} P1 + P4 - P5 + P7 & P8 + P5 \\ P2 + P4 & P1 + P3 - P2 + P6 \end{matrix} \right| \tag{4}$$

$$\begin{aligned} p1 &= a*(f-h) \\ p2 &= (a+b)*h \\ p3 &= (c+d)*e \\ p4 &= d*(g-e) \\ p5 &= (a+d)(e+h) \\ p6 &= (b-d)(g-h) \\ p7 &= (a-c)(e+f) \end{aligned} \tag{5}$$

In terms of P1 through P7:

$$\begin{aligned} r &= p5+p4-p2+p6 \\ s &= p1+p2 \\ t &= p3+p4 \\ u &= p5+p1-p3-p7 \end{aligned} \tag{6}$$

The Strassen’s algorithm can thus be written as:

1. Partition matrices A and B into sub matrices; add and subtract to form terms as shown in Equations (5) and (6).
2. Perform 7 recursive multiplications.
3. Add and subtract terms to form C.

Speedup

According to El-Nashr (2011) speedup is defined as the ratio of serial execution time to a parallel execution time. It is used to express how many times a parallel program works faster than its serial version used to solve the same problem. Many conflicting parameters such as parallel overhead, hardware architecture, programming paradigm, programming style, memory hierarchy may negatively affect the execution time of a parallel program. This would make its execution time larger than that of the serial version and thus any parallelization gain will be lost. Speedup can be computed as

$$S(p) = \frac{execution\ time\ u\sin g\ one\ processor}{execution\ time\ u\sin g\ multiprocessor} = \frac{t_s}{t_p} \tag{2}$$

Where t_s is the execution time on a single processor and t_p is execution time on a multiprocessor.

Related Work

Lee et ‘al. (2018) proposed to speed up distributed machine learning by using coded matrix multiplication to overcome the problem of slow system due communication bottle neck or system failure.
Kumar et al. (1995) attempt to reduce the amount of memory required by Strassen algorithm from $O(7^n)$ to $O(4^n)$ for multiplying $2^n \, x \, 2^n$ matrices. This reduction was made possible through the reuse of working storage.
Mathew and Kumar (2012) compared the execution time complexity and space complexity between Strassen’s and conventional algorithms for matrix multiplication where they found that the Strassen’s algorithm is more efficient than conventional algorithm on large sizes of matrices. Their result also showed that the Strassen’s algorithm needs more memory allocation.
Saravanan et al. (2012) implemented the Naïve and Strassen’s matrix multiplication algorithms on a Dual core (using Visual Studio 2005). Their result showed optimizing the code using OpenMP increased the performance of the parallel algorithm over the sequential implementation.
Karstadt and Schwartz (2017) in their attempt to find a little faster matrix multiplication algorithm based on Strassen’s-like matrix multiplication algorithm presented a theoretical analysis for finding the optimal number of recursive steps without tuning. Their result showed that the new algorithm outperforms Strassen’s-Winograd algorithm for matrix size n>=32.
Kawu et al. (2017) proposed the One-Level Recursion Strassen’s algorithm to reduce the amount of temporary storage requirement by stopping the recursion very early.
Karstadt and Schwartz (2020) propose to improve the leading coefficient of Strassen’s and other fast matrix multiplication algorithms by extending the Bodrato’s intermediate representation method for matrix squaring to an alternative basis multiplication method.
In this research we focus on determining the performance of One-Level Recursion parallel Strassen’s algorithm on multi core processor. The objective of the research is to compare the speedup obtainable by running parallel Strassen’s and parallel conventional matrix multiplication programs on Dual core processor with 2G and 4G RAM (Random Access Memory). The programs were written in C++/OpenMp
The remaining part of the paper is organized as follows: section 2 is the

methodology, section 3 presents the results obtained, section 4 is discussion of the result, section 5 is the conclusion, section 6 is the acknowledgement.

MATERIALS AND METHODS

C++ Programs optimized with OpenMp library directives were developed to implement the parallel conventional and parallel One-Level Recursion Strassen's multiplication algorithms. The programs were run on Intel T4500 Dual core processor. The specifications of the system and the required software are shown in Table 1.

Table 1: Hardware and Software requirements.

Processor	Intel T4500 Dual Core
CPU speed	2.30GHz
RAM	2GB/4GB
Operating System	Windows 7
Software	Visual Studio 2010

Matrices of different sizes ranging from 100 x 100 to 2000 x 2000 were used in the implementation of the programs. For the input matrices A and B of size n x n, the elements will be generated automatically within the program where the user is only required to enter the dimension of the two matrices and the elements will be generated randomly using the Ran() function of C++. The generated elements were multiplied by 1000 and converted to integer. Since the same program would be run for all dimensions of the matrices. Static padding (zeros would be added to make the matrix dimension a power of 2 is use to handle odd size matrices. For instance, if a matrix dimension of 5 is entered by a user, zeros are added to make the matrix 8 x 8) is used for a matrix size that is not a power of 2.

The sequential programs for both conventional and Strassen's matrix multiplication algorithms were written in Visual Studio 2010.

In the Strassen program, the matrices A, B and C are partitioned in to four sub matrices each. The matrix C is calculated using the algorithm shown in section 1.3.

The parallel programs were obtained by adding OpenMP library routines. The programs are parallelized by Loop level parallelism using #pragma parallel for directive.

Calculation of execution time for both the parallel conventional and the parallel One-Level Recursion programs will commence after creation of the matrices and stop immediately after the multiplication is completed and all allocated spaces on the heap were released.

Each program is run three times and the average execution time is calculated.

RESULTS

This work focuses on investigating speedup gain due to increasing memory size during the execution of One-Level Recursion parallel Strassen's algorithm. The Conventional and One-level recursion programs were run on Intel Dual core with 2GB and 4GB memory. The result is presented below.

Table 2: Execution Time (in seconds) of Parallel Conventional Strassen's algorithms on 2GB memory.

Matrix size	Conventional Algorithm	Strassen's Algorithm
100 x 100	0.06	0.04
200 x 200	0.29	0.22
500 x 500	4.09	1.57
1000 x 1000	37.42	26.36
2000 x 2000	107.43	93.46

The execution time of the parallel Strassen's algorithm is better than the conventional algorithm (Table 2). This is due the inherently parallelizable nature of the Strassen's algorithm.
The execution time of the parallel Strassen's algorithm improves by about 20% for matrix size 500 and 2000. However, a -0.8% improved was recorded for matrix size 1000 (Table 3).

Table 3: Execution Time (in seconds) of Parallel Conventional Strassen's algorithms on 4GB memory.

Matrix size	Conventional Algorithm	Strassen's Algorithm
100 x 100	0.07	0.04
200 x 200	0.31	0.20
500 x 500	4.40	1.16
1000 x 1000	33.20	26.44
2000 x 2000	111.38	74.54

The execution time of the parallel one-level recursion Strassen's algorithm reduces significantly for matrix size above 500 (figure 1)

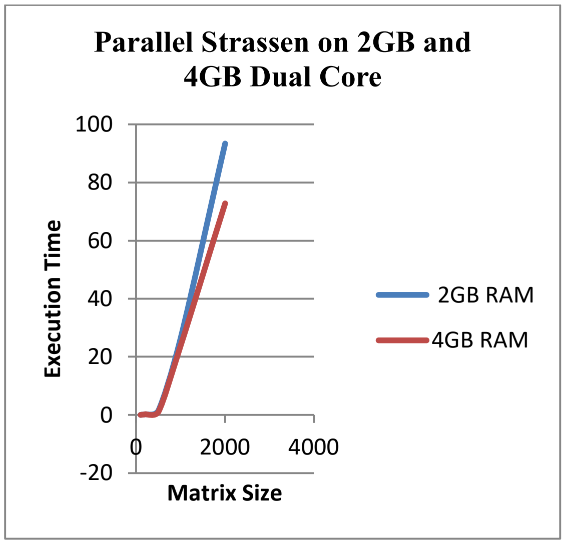

Figure 1: Comparison of the execution time of parallel Strassen's algorithm on Dual Core with 2GB and 4GB RAM.

DISCUSSION

The overall result of the experiment carried out to determine the effect of increasing memory size on the performance of One-Level Recursion parallel Strassen's algorithm was presented in section 3 above. As seen in Tables 2 and 3, there is a marginal reduction in the execution time of parallel Strassen's algorithm as compared to the conventional program. The loss of improvement observed for matrix size of 1000 may be due to saturation of the L2 cache of the processor. The L2 cache has a size of 1MB and is shared by the two cores. The overall result shows a minor improvement in execution time when the memory size is doubled. This may due the difficulty of implementing Strassen's algorithm on computers with memory hierarchy like the Intel dual core used in this research work.

Acknowledgement

This research work is the extension of our paper presented at NIGERCON IEEE conference in 2017, in Owerri, Nigeria.

REFERENCES

Dash, Y., Kumar, S., &Patle V. K. (2015). *A Survey on Serial and Parallel Optimization Techniques Applicable for MatrixMultiplication Algorithm*.American Journal of Computer Science and Engineering Survey.Vol. 3(1).071-077.

El-Nashar, A. I. (2011). *To Parallelize or not to Parallelize, speed up issues.*International Journal of Distributed and Parallel Systems Vol.2, No.2.

Gu, Z., Moreira, J., Edelsohn, D. and Azad, A. (2020). arXiv2002.11302v1[cs.DC].

Huss-Lederman, S. Jacobson, E. M., Johnson, J. R., Tsoa, A. & Turnbull T. (1996). *Implementation of Strassen's Algorithm for Matrix Multiplication.*

Karstadt, E. & Schwartz, O. (2017). Matrix Multiplication, a little faster. In proceedings of SPAA' 17, *Washington DC, USA, July 24 – 26, 2017.*

Karstadt, E. & Schwartz, O. (2020). Matrix Multiplication, a little faster. *Journal of ACM, Vol. 67. No.1. article 1.*

Kawu, A. J., Yahaya, A. U., & Bala, S. I. (2017). Performance of One-Level Recursion Parallel Strassen's Algorithm on Dual Core Processor. *IEEE NIGERCON 2017.*

Kumar, B., Huang, C. H., Sadayyappan, P., and Johnson, R. W. (1995). A tensor Product Formulation of Strassen's Matrix Multiplication Algorithm with Memory Reduction. *Scientific Programming Vol. 4 pp 275 – 289*.

Lee, K., Lam. M., Pedarsani, R., Papailiopoulus, D., and Ramchoudran, K. (2018). *arXiv:1512.02673v3[cs.DC].*

Mathew, J. and Kumar R. V. (2012). *Comparative Study of Strassen's Matrix Multiplication Algorithm. International Journal of Computer Science and Technology.Vol. 3(1). Pp. 749- 754.*

Rathore, Y. S., & Kumar, D. (2014). *Performance Evaluation of Matrix Multiplication Using Mix Mode Optimization Technique and OpenMP for Multi-core Processors*. *IOSR Journal of Engineering.Vol. 04 (3).*

Saravan, V., Radhakrishna M., Basavesh, A.S. and Kathori B. (2012). A Comparative Study on Performance Benefits of Multicore CPU Using OpenMP**.** *International Journal of computer science issues, Vol. 9, Issue 1, No. 2.*

Sharma, R. &Munir, S. (2015). Time Efficient Implementation of Matrix Multiplication for Signal Processing. International Journal *of Artificial Intelligence and Mechatronics.Vol. 4(2).*

Tang, Y. (2019). Improving Space-Time Efficiency of Processor-Oblivious Matrix Multiplication Algorithms. *arXiv:1911.05328v1[cs.DC].*

Thabet, K. and Al-Ghuribi, S. (2012). Matrix Multiplication Algorithms.*International Journal of Computer Science and Network Security, Vol. 12, No. 2***.**

Thottethodi, M., Chatterjee, S. & Lebeck, A. R. (1998). *Tuning Strassen's Matrix Multiplication for Memory Efficiency*.Unpublished.

Thakur, V. & Kumar S. (2016). An Analytical Study on Effect of Parallelism on Multi-core. International Journal of Science Engineering and Technology Research. Vol. 4 (8) pp. 2664.

Zin Oo. N., & Chaikan, P. (2019). Efficient Implementation of Strassen's Algorithm for Memory Allocatio Using AVX Intrinsic on Multicore Architectures. 34th International Technical Conference on Circuits/Systems Computer and Communication. IEEE.