Bima Journal of Science and Technology, Vol. 4(2) Dec, 2020 ISSN: 2536-6041

SCALABILITY EVALUATION OF ONE-LEVEL RECURSION PARALLEL STRASSEN'S ALGORITHM ON MULTICORE PROCESSORS

[1*]AHMAD JIBIR KAWU, [1]AISHATU YAHAYA UMAR AND [2]SAMINU I. BALA

[1]Department of computer science, Gombe State University
[3]Department of Mathematical Sciences, Bayero University, Kano

Corresponding Author: ahmadkawujibir@gmail.com

ABSTRACT

Scalability plays a key role in determining the potential worthiness of a parallel system . It can be used to determine the performance of a parallel system on a large volume of data given a certain number of processors from known performance on fewer processors. Scalability is a measure of the ability of a parallel system (algorithm + architecture) to decrease the computation time in proportion to the number of available processors. The upper bound of scalability is the minimum number of processors for which the speedup takes the minimum value. Common parallel algorithm evaluation metric like Amdahl's law and Gustafson –Barsis's have the draw back that they do not take into account the parallel overhead and therefore overestimate the speedup. This makes them unsuitable for investigating scalability. The main characteristics of scalability are speedup and efficiency. The objective of this paper is to evaluate the scalability of One-Level Recursion parallel Strassen's algorithm on multicore processors (Intel Dual core and AMD Quad-core). This algorithm is a variant of the Strassen's algorithm where recursion stops at level one. Thus, as a new algorithm, there is need to determine its scalability. This is done by experimentally determining the fraction of sequential code of a parallel algorithm and the parallel overhead using Karp-Flatt metric (to determine whether they are responsible for or against speedup). The results obtained showed that both parallel overhead and fraction of sequential code have negligible impact on the performance of the One-Level Recursion Strassen algorithm. Furthermore, a super linear speedup was observed on the quad-core processor. This will help in prediction and decision making of parallel system combination for cost effective and efficient performance.

Keywords: One-level recursion, Scalability, Partitioning, Parallel Overhead, Isoefficiency

INTRODUCTION

The wide application of matrix multiplication and its inherently embarrassingly parallelizable nature attracts the attention of many researchers. Different matrix multiplication algorithms with different asymptotic complexities, for example, Stassen's, Winograd, and Copper smith have been developed. Research is still active in trying to develop techniques to reduce the time and space complexity of the available matrix multiplication algorithms (Gu *et al*., 2020). Availability of affordable and ever increasing number of parallel computing systems has attracted a lot of interest in to parallelize and hence reduce the execution time of matrix multiplication algorithm (Sarmah, *et al*., 2019). A scalable system shows a steady increase in speedup in such a rate that the efficiency is maintained as the number of processors increases (Hwang, 2001; Perlin *et al*., 2016). But the parallel overhead increases inversely

proportional to efficiency as the number of processors increases.. Increasing the problem size is a way to maintain efficiency (Spiliotis *et al*, 2020). According to Kalinov (2006) a parallel system is a combination of parallel algorithms and parallel architectures However; parallel systems come with some overhead cost and bottlenecks. Parallel overheads include communication cost, redundant computation, imbalanced workload, and architecture overhead. Thus there is need to check the effectiveness of a parallel algorithm by investigating its scalability as more processors and data are added to the system (Fan, *et al*., 2020).

According to Amdahl's law, there is an upper bound to the speedup obtainable by increasing number of processors (Wilkinson and Allen, 2005). Amdahl's law and Gustafson –Barsis's are also use to evaluate the performance of a parallel system. These metrics have the draw back that makes them unsuitable for investigating scalability. The drawback is that they do not take into account the parallel overhead and therefore overestimate the speedup. Other metrics for determination of scalability of parallel systems are available. Common metrics for testing scalability are Karp-Flatt and Isoefficiency metric. The Karp-Flatt metric decides whether the principal barrier to speedup is due to sequential code or parallel overhead (Harwell and Gini, 2019). Isoefficiency determines the degree of scalability of a parallel system by deciding the rate at which input and number of processors should be increased to maintain a constant efficiency (Spiliotis *et al*, 2020). We chose to use Karp-Flatt metric because it can be experimentally used to determine the principal barrier to scalability from the values of speedup. But isoefficiency has the drawback of not having a generic method of computation (Prasad *et al*., 2018; Shudler *et al*. 2017).

Speedup is defined as the ratio of serial execution time to a parallel execution time (El-Nashr, 2011). It expresses how many times a parallel program works faster than its serial counterpart used to solve the same problem. Speedup is mainly affected by parameters such as programming paradigm, parallel overhead, and hardware architecture.

Speedup *S* is mathematically defined in Eq. 1;

$$S = \frac{ts}{tp} \qquad (1)$$

Where *ts* is the serial execution time and *tp* is the parallel execution time.

Efficiency is the measure of the fraction of time for which a processing element is effectively and usefully employed.

Efficiency *E* is defined mathematically in Eq. 2;

$$E = \frac{S}{p} \qquad (2)$$

Where, *S* is the speedup and *p* is the number of processors.

Efficiency of a sequential or parallel matrix multiplication strongly depends on the hardware architecture.

In this research work, we try to experimentally determine the scalability of our new algorithm (one-step recursion parallel Strassen's algorithm) on multicore processors. The algorithm was presented in a seminar paper (Kawu, *et al*, 2017). The algorithm was run on a dual core processor to determine its speedup on the multicore processors. However, in this work, we investigated how increasing number of processors and input data affects speedup (scalability). The number of processors is

increased from two to four. The parallel programs were run on these processors. The speedup obtained by running the parallel program on Dual and Quad core processors was used to investigate the scalability of the parallel algorithm on the multicore processors. That is running the algorithm on a higher number of processors to see whether increasing the number of processors has effect on the performance of the algorithm. Portion of a parallel program that must be run on a single processor while all other processors remain idle is called the sequential portion of the parallel program. Additional burden due to parallel execution of a program such as communication cost and synchronization are called parallel overhead. These factors may inhibit the performance of a parallel program. The recorded speedup was used to determine the sequential portion of the program that may inhibit linear speedup as more processors are added. We also use the speedup to calculate the parallel overhead of the parallel program.

The performance of a given parallel algorithm for a specific problem on a given number of processors is not sufficient to provide answers to several question such as how can the program perform on a different architecture? How does changing processor speed and communication speed affect the performance of the parallel system? In our previous work Kawu *et al* (2017), we investigated the speedup of our algorithm on a dual core processor. In our recent work kawu *et al* (2020), we investigated the effect of adding more memory on the performance of the algorithm on a dual core processor. But in this paper we investigated the scalability of our algorithm when more processor is added. We reference some work done to evaluate the scalability of several algorithms below.

Arrigoni and Massini (2019) proposed a new cache oblivious algorithm for the A^tA multiplication. The algorithm is a variant of Strassen algorithm. They implemented the algorithm using MPI paradigm on a Galileo cluster and tested the performance of the system on a subset of the cluster nodes. Their result showed a good scalability and speedup. They further observed that sequential portion of the algorithm and parallel overhead have negligible impact on the performance of the algorithm. Iqbal *et al* (2020) used the Karp-Flat metric to determine the sequential portion of parallel algorithm they run a cluster of four computers using Hadoop and Stark services. This was to allow them compare the scalability of the algorithm on the two parallel architectures. Furthermore, Spiliotis *et al*., (2020) use the Karp-Flatt metric to determine the serial fraction of their parallel algorithm in order to measure the parallel overhead of the algorithm.

Peng *et al*. (2008) compares the performance of Intel Core 2 Duo, an Intel Pentium D and AMD Athlon 64 x 2 processor using multi-program and multithreaded workloads. Their result showed better scalability for processors with fast cache-to-cache communication, large L2 cache, fast L2 to core latency and fair cache resource sharing. The major challenge of the system is the sharing of the L2 cache by more than one core when they have different demands from cache memory. In a separate work, Artemov *et al*. (2019) compared the scalability and efficiency of three parallel algorithms RINCH, LIF and IRSI for parallel inverse factorization of block-sparse Hermitian positive definite matrices. They reported that LIF is the most efficient and scalable algorithm. Höfinger and Haunchmid (2017) proposed a simple procedure for estimating parallel overhead

of a program based on run-time records. Misra *et al*. (2018) investigated the scalability of Stark (a distributed matrix multiplication of large and distributed matrices using Spark framework). They generated three test cases, each containing a different set of two matrices of dimensions equal to (4096 x 4096), (8192 x 8192) and (16384 x 16384). They experimentally showed that Stark has a strong scalability with increasing matrix size where they multiplied two 16384 x 16384 matrices.

Sokolinsky (2017) presented a new high-level parallel computational model named BSF- Bulk Synchronous Farm; the model was intended to determine the scalability of a multiprocessor system with distributed memory. Chen *et al*., (2020) investigated the scalability of machine learning algorithms. They chose four kinds of parallel machine learning algorithms: (1) Asynchronous parallel SGD algorithm (2) Parallel mode average SGD algorithm (3) Decentralization optimization algorithm (4) Dual coordinate optimization algorithm. Their result showed that character learning dataset decides the scalability of the machine learning algorithms.

Stefanes, Rubert and Soares (2020) demonstrated the scalability of coarse grained parallel algorithms for the maximum matching problem in a convex bipartite graph. Marowka (2020) examined the practical performance Cormen's Quicksort parallel algorithm with different programming approaches. The author compared the capacity for theoretical prediction of an algorithm's performance with predictions based on combination of theoretical and practical analyses. Kathavate and Srinate (2014) analyzed the efficiency of a parallel matrix multiplication program on a multicore system. Their result showed that maximum speedup obtained is less than number of processors.

The general assumption that engaging more number of processors in parallel program to solve a problem will continue to increase the speed of execution need to be investigated. Scalability analysis can be used to choose the best algorithm-architecture combination for a problem under different constraints on the growth of the problem size and the number of processors. It can also be used to predict performance of a parallel algorithm and parallel architectures for a large number of processors from a known performance from fewer processors (Kumar and Gupta,1994; Spilitios, *et al*.,2020). The objective of this paper is to evaluate the scalability of One-Level Recursion parallel Strassen's algorithm on multicore processors. This will help in prediction and decision making of parallel system combination for cost effective and efficient performance. The remaining of the paper is organized as follows: Section 2 gives a brief explanation of method and materials used for the implementation of the sequential and parallel programs. Section 3 presents the results of running the programs on Intel Dual Core and AMD quad-core processors and the calculated parallel overhead, fraction of sequential code and efficiency. The final part, which is Section 4 and 5 highlight the implication of the result, drawback and motivation for further research.

MATERIALS AND METHODS

A C++ programs for the implementation of sequential and parallel Strassen's matrix multiplication algorithms were run on Dual and Quad core processors. The execution time for different matrix sizes (100, 200, 500, 1000, and 2000) were recorded. The sequential program was optimized with OpenMP library routines. The optimized

program was also run on the Dual and Quad core processors and execution time recorded. The system specifications of the two multicore processors are given in Table 1.

Table 1: Hardware and Software Specifications.

Processor	**Intel T4500 Dual Core**	**AMDa A6-3240M Quad-core**
CPU speed	2.30GHz	1500Mhz
RAM	2GB/4GB	4GB
Operating System	Windows 7	Windows 7
Software	Visual Studio 2010	Visual Studio 2010

Input Design

For the input matrices A and B of size n x n, the elements were generated automatically within the program since it is very cumbersome to enter using keyboard, a matrix of size 100 not to talk of 1000, 2000 and so on. The user may only be required to enter the dimension of the two matrices since the same program would be run for all dimensions of the matrices.

Output Design

The output matrix C will neither be displayed nor written to a text file. It is only the execution time that is recorded. A block of code that display the result was provided (in console application only) to check if the matrix multiplication algorithms are giving the correct result.

Execution Time

Calculation of execution time for both the serial and the parallel programs will commence after creation of the matrices. And stops immediately after the multiplication is completed and all allocated spaces on the heap are released.

Parallel Program Design

As stated in section 2.1, the input matrices would be created by a single processor. Depending on the number of available cores a number of threads will be spawned and the matrices would be initialized and partitioned among the cores. Each core will multiply the portion allocated to it. The parallel program was written in Visual Studio 2010 (C++) using the OpenMP library routines. The choice of the programming language is based on the hardware available for testing the program; that is the available hardware is Multi-core processors and OpenMP is suitable for programming Shared memory multiprocessors.

Partitioning

Partitioning is the breaking down of the problem into discrete “chunks” of work that can be distributed to multiple processors. In this research work, the matrices A and B are partitioned into 4 for the Strassen’s algorithm. If a matrix is a power of 2 its dimension is maintained while if it is not a power of 2 it is extended to the next power of 2, for example a dimension of 100 will be extended to 128, 900 to 1024, and so on the new elements are filled with zeros. That is static padding was employed.

Parallel Implementation

To implement parallelism, the sequential programs were modified by adding loop level parallel construct (for example, #pragma omp parallel for) to the program. Each loop would be parallelized. Depending on the number of available processors, threads would be specified using omp_set_num_threads () library routine.

The two input matrices A and B were generated automatically inside the program using the Ran() function of C++. The generated random numbers were multiplied by 1000 and converted to integer. Memory space was allocated to the matrices A, B, the product matrix C and all the intermediate matrices of the Strassen's algorithm on the heap using the new operator. Calculation of the execution time starts immediately after the creation of matrices and stops when all processors finish their work and all allocated spaces are released. Each program was run three times and average execution time calculated.

Scalability

To determine the scalability of the Strassen's algorithm, we are going to calculate the fraction of the sequential portion of the parallel program and also calculate the parallel overhead of the program. This will enable us see whether the sequential portion is growing with increasing number of data input and number of processors or reducing. In addition, we will also see how the parallel overhead increase with increase in the number of processors.

We shall also consider efficiency metric to determine how increasing number of processing elements affects efficiency.

To determine the serial fraction of program using speedup, let use the following notations:

S is speedup, e serial fraction, ts serial execution time, tp parallel execution time.

Parallel execution time tp is

$$tp = \frac{ts(1-e)}{p} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

$$S = \frac{ts}{tse+\frac{ts(1-e)}{p}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(4)$$

$$S = \frac{1}{e+\frac{1-e}{p}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5)$$

$$\frac{1}{S} = e + \frac{1-e}{p} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(6)$$

$$e = \frac{\frac{1}{s}-\frac{1}{p}}{1-\frac{1}{p}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(7)$$

The parallel overhead is the extra time spent in executing a parallel program by a number of processors.

Let *to* be parallel overhead,

$$ptp = ts + to \quad \ldots\ldots\ldots\ldots\ldots\ldots (8)$$

$$to = ptp - ts \quad \ldots\ldots\ldots\ldots\ldots\ldots(9)$$

RESULTS

The section presents the result of running the sequential and parallel Strassen's programs on Dual and Quad core processors. The main focus of the research is to determine the scalability of One-Level Recursion Strassen's algorithm. The speedup obtained on Dual core increases linearly up to a matrix size of 1000 where the speedup reduced drastically and then increased at 2000. However, on Quad core processor the speedup increases at 1000. This may be due to sufficient cache memory (Table 2). The efficiency of the parallel programs increases with increasing size of input matrices. For matrices size from 500 and more, efficiency above 100% was recorded except for a matrix size of 1000 where the efficiency falls below 100% (Table 3) on the dual core processor.

Table 2: Comparison of Speedup between Sequential and Parallel Strassen's on Dual versus Quad-core

Mat. size	Exec.Time seq.(s)	Exec.Time duo(s)	Exec.Time quad(s)	Speedup duo	Speed-up quad
100	0.06	0.04	0.03	1.5	2
200	0.33	0.2	0.12	1.65	2.75
500	3.27	1.16	0.63	2.82	5.19
1000	41.42	23.79	5.36	1.75	7.73
2000	193.73	72.77	41.56	2.66	4.66

Table 3: Efficiency (%) of parallel Strassen's on Duo and Quad Cores.

Mat. Size	Speed-up duo	Speed-up quad	Efficiency duo	Efficiency quad
100	1.5	2	75	50
200	1.65	2.75	82.5	68.75
500	2.82	5.19	141	129.75
1000	1.74	7.73	87	193.25
2000	2.66	4.66	133	116.5

The serial fraction of the parallel One-Level Recursion program calculated from equation 3 to 7 is presented in Table 4.

Table 4: Fraction of sequential portion of the parallel program

Matrix size	Dual Core	Quad Core
100	0.34	0.33
200	0.22	0.15
500	-0.3	-0.08
1000	0.14	-0.16
2000	-0.26	-0.05

The parallel overhead of running the algorithm on Dual and Quad core processors calculated from equation 8 and 9 shows a negligible overhead on both processors for matrix size above 200 (Table 5).

Table 5: Parallel Overhead

Matrix size	Dual Core	Quad Core
100	0.02	0.06
200	0.11	0.15
500	-0.13	-0.75
1000	-6.17	-37.45
2000	-6.18	-27.49

DISCUSSION

This paper focuses on evaluation of the scalability of One-Level recursion parallel Strassen algorithm on multicore processors. C++ programs were developed for sequential and parallel algorithms. The parallel programs were optimized with OpenMP library routines. The programs were run on dual and quad core multiprocessors. The overall result obtained showed a significant scalability for medium and large matrix sizes. The effect of sequential fraction of code is significant in small matrix sizes but negligible in larger matrices.

For smaller matrices, the sequential execution time is very large. However for larger matrices (1000 x 1000) the sequential portion is negative (Table 2) because matrix multiplication is embarrassingly parallel. This result agrees with the findings of Thakur and Kumar (2016).. Besides large shared L2 cache also improves performance of a multi- core processor (Peng *et al*., 2008). The sequential portion reduces between matrix size 100 to 500 (Table 2). Both the parallel overhead (Table 3) and the sequential portion reduces with increasing matrix size. From matrix size 500 they become negligible. The negative sequential portion and parallel overhead in tables 4 and 5 shows that they are negligible. The negative values of both sequential portion of

code and parallel overhead signifies a super linear speedup where speedup is higher than number of processors. This result is in agreement with the work of Arrigoni and Massini (2019).

CONCLUSION

The results presented and discussed above indicate that one-Level Recursion parallel Strassen's algorithm is scalable and has negligible sequential portion and parallel overhead as matrix size and number of processors increases. This result can be used in selection of hardware and algorithm combination for solving a particular parallel program. An important drawback of this research is that even though the scalability of the parallel algorithm supposed to be tested on a large number of processors like 5, 6, 7, 8, and so on. we believed that doing this on a dual and quad core will give an insight into the scalability of multicore processors and motivates further research. In addition, there is need for further research to determine the real cause for the drastic fall in execution time for a matrix size of 1000 on the dual core processor.

REFERENCE

Arrigoni, V. and Massini, A. (2019). Fast Strassen-based A^tA Parallel Multiplication.

Artemov, A.G., Rudberg, E. and Rubensson, E. H. (2019). Parallelization and Scalability Analysis of the Inverse Factorization Using Chunks and Task Programming Model.

Cheng, D., Zhang, H., Xia, F. Li, Y. and Zhang Y. (2020).The Scalability for Parallel Machine Learning Training Algorithm: Dataset Matters.

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.

Fan W., HE, K., LI, Q and WANG. Y. (2020). Graph Algorithms: Parallelization And Scalability Science China Information Science Vol. 63 203101:2.

Gu, Z., Moreira, J., Edelsohn, D. and Azad, A. (2020). Bandwidth-Optimized Parallel Algorithms for Sparse Matrix-Matrix Multiplication using Propagation Blocking.

Harwell, J. and Gini, M. (2019). Swarm Engineering Through Quantitative Measurement of Swarm Robotic Principles in 10,000 Robot Swarm.

Höfinger, S. and Haunchmid, E. (2017). Modeling Parallel Overhead from Simple Run-time Record. Springer.

Hwang, K. (2001). Advanced Computer Architecture: Parallelism, Scalability, Programmabity. Tata-Mc-GrawHill). New Delhi.

Iqbal, W., Khan, N., Mahmood, A. and Errati, A. (2020). Canny Edge Detection Hough Transform for High Resolution Video Streams Using Hadoop and Stark. Cluster Computing.

Kalinov A. (2006) Measuring the Scalability of Heterogeneous Parallel Systems. In: Wyrzykowski R., Dongarra J., Meyer N., Waśniewski J. (eds) Parallel Processing and Applied Mathematics. PPAM 2005. Lecture

Notes in Computer Science, vol 3911. Springer, Berlin, Heidelberg.

Kathavate, S. and Srinate, S.K. (2014). Efficiency of Parallel Algorithms on Multicore Systems Using OpenMP. International Journal of Advanced Research in Computer and Communication Enineering. Vol. 3 Issue 10.

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

Kawu, A. J., Yahaya, A. U., and Bala, S. I.(2020). Effect Of Memory On The Performance of One-Step Recursion Parallel Strassen's Algorithm On Dual Core Processor. Bima Journal of Science and Technology, Vol. 4(1).

Kumar, V. and Gupta, A. (1991). Analyzing Scalability of Parallel Algorithms and Architecture. Proceedings of 5th International Conference on Super Computing pp 396 – 405.

Misra, C., Bhattacharya, S., and Gosh, k. S. (2015). Stark: Fast and Scalable Strassen's Matrix Multiplication using Apache Spark.

Morowka, A. (2020). On the Performance Difference Theory and Practice for Parallel Algorithms. Journal of Parallel and Distributed Computing.

Peng, L, Peir, J, Prakash, T., Staelin, C., Chen, Y., Koppelmanl, D. (2008). Memory Hierarchy Performance Measurement of Commercial dual-core desktop processors Journal of Systems Architecture. pp 816–828. Elservier.

Perlin, N., Zysman, J. P., and Kirtman, B. P. (2016). Practical scalability assesment for parallel scientific numerical applications.

Prasad, S., Gupta, A., Rosenberg, A., Sussman, A. and Weems, C.(2018). Parallel and Distributed Computing (1st Ed.). Paris. Springer.

Sarmah, S. Bhuyan, M.P. Deka, . V Rahman, M. Sarma, P. Sarma, S.K.(2019). Measuring The Performance of Multi-Core Architecture Using Openmp. International Journal Of Scientific and Technology Research Volume 8, Issue 10.

Shuler, S., Calotois, M., Hoefler, T. and Wolf, F. (2017). Isoefficiency in Practice: Configuring and Understanding the Performance of Task-Based Applications. ACM Digital Library.

Sokolinsky, L. B. (2017). Analytical Estimation of Iterative Numerical Algorithms on Distributed Memory Multiprocessors.

Spilitios, I.M., Bekakos, M. P.. and Boutalis, Y. S. (2020). Parallel Implementation of the Image Block Representation Using OpenMp. Journal of Parallel Computing 137 pp 134-147.

Stefanes, M.A., Rubert, D.P., and Soares, J. (2020). Scalable Parallel Algorithm for Maximum Matching and

Hamiltonian Circuit in Convex Bipartite Graphs. Elservier.

Thakur V. and Kumar, S. (2016). Impact of Parallelism on Dualcore. International Journal of Science Engineering and Technology Research, Vol. 5 Issue 8, pp 2664 – 2667.

Trobec, R., Shivnik, B., Bulic, P. and Robic, B. (2018). Introduction to Parallel Computing (From Algorithm to Programming on the State-of-the-Art Platform). Springer. Switzerland.

Wilkinson, B. and Allen, M. (2005). Parallel Programming Techniques and Applications using networked Workstations and Parallel computers. U.S.A. Pearson Prentice Hall.