British Journal of Mathematics & Computer Science
10(2): 1-12, 2015, Article no.BJMCS.18772
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Determining the Effects of Cross-over Point on the Running Time of Strassen Matrix Multiplication Algorithm

S. C. Agu[1*] and T. A. Atabong[1]

[1]Department of Computer Science, Madonna University, Elele, Nigeria.

Article Information

DOI: 10.9734/BJMCS/2015/18772
Editor(s):
(1) Kewen Zhao, Director and Professor, Institute of Applied Mathematics and Information Sciences, Department of Mathematics, University of Qiongzhou, Sanya, P.R. China.
(2) Morteza Seddighin, Indiana University East Richmond, USA.
Reviewers:
(1) Grienggrai Rajchakit, Department of Mathematics, Mae jo University, Thailand.
(2) G. Y. Sheu, Accounting and Information Systems, Chang-Jung Christian University, Tainan, Taiwan.
Complete Peer review History: http://sciencedomain.org/review-history/10079

Short Research Article

Received: 09 May 2015
Accepted: 25 May 2015
Published: 07 July 2015

Abstract

This paper studies Strassen's algorithms for fast multiplication of two finite dimensional matrices. However, one pertinent issue that has deterred Strassen's scheme from been considered for practical usage is determining the cross-over point. In this light, large matrices with different sizes were randomly generated on which Strassen and conventional matrix multiplication algorithms were implemented in MATLAB R2008b. Two MATLAB built-in functions nextpow2 and pow2 were used for implementing padding techniques to ensure that the matrices are to the power of two. Three different experiments were carried out using five, four and three levels of recursion (divide and conquer algorithm) respectively to determine the suitable cut-off point n_0 which were used to evaluate the optimal running time for Strassen's algorithm. For each experiment, eight finite dimensional square matrices of real numbers were generated and iteratively multiplied. The experiment reveals that the cut-off point with five level of recursion optimized the Strassens time.

Keywords: Matrix multiplication; Strassen algorithm; conventional algorithm; divide and conquer algorithm; cross-over point; padding techniques; matlab statements.

1 Introduction

The central role of the conventional matrix multiplication algorithm as a building block in solving problems in algebra and scientific computations has generated a significant amount of research into techniques for improving the performance of this algorithm [1]. Since 1960, many matrix multiplication algorithms have been discovered to multiply matrices fast with the cost estimate yielding very complex algorithms that are

**Corresponding author: sndyaguu@gmail.com*;

impractical for many matrices of reasonable sizes [2]. The order of conventional algorithm has been established to be $O(n^3)$ and a way forward is to reduce this order to $O(n^\alpha)$ where $\alpha < 3$.

Strassen's divide and conquer matrix multiplication algorithm (MMA) was developed and found to be of order $O(n^{2.81})$ with a significant gain of 0.19(or 19%) in time and cost (see for example the work of Harvard) [3]. As a consequence, for sufficiently large values of n (in thousands), Strassen's algorithm will run faster on the one hand, as well as offers a significant improvement in cost efficiency on the other hand than the conventional algorithm for matrix multiplication.

Strassen's MMA has a number of variations including Winograd's variant of this algorithm. Both algorithms are known to provide efficient single- and double-precision Graphics Processing Unit (GPU) implementations [4]. The single-precision implementations of these two algorithms have been compared analytically using the arithmetic count, device-memory transactions, and device memory to multiprocessor data volume metrics. Most of the analysis that have been performed on these algorithms indicates that, for 16384 x 16384 matrices, single-precision implementation of Strassen's algorithm limited to four levels of recursion reduces the number of arithmetic operations by 41.3%, the number of transactions by 33.7%, and the volume by 29.2% relative to the best known GPU implementation of the classical $O(n^3)$ MMA. The corresponding reductions achieved by Winograd's variant are 41.3%, 35.1%, and 31.5% respectively.

Some researches on Strassen's algorithm and its application to multiplication considered special case of odd-sized representation of mathematical arrays of real numbers in rhomboidal form like structures (called rhotrices) basically by means of both padding and peeling approaches. A sequential recursive implementation as well as the modelling of a parallel framework implementation of the algorithm using java program has been done for rhotrices of relatively medium sizes [5].

There are newer and practically useful MMA, such as karatsuba's fast multiplication of large numbers, the Coppersmith-Winograd algorithm each of complexity $O(n^{2.376})$. The Ladderman non-commutative algorithm for multiplying 3 x 3 matrices using 23 multiplications developed in 1975 is still the best known for the 3 x 3 cases, even though its exponent is not as good as Strassen's [6]. Even though faster, these newer algorithms are significantly more complicated to implement than Strassen's [7]. A thorough investigation of the usefulness of these other techniques for an actual implementation has not yet been carried out and a lead way to the analysis of these algorithms, is to completely investigate the cross over point in Stressen's technique.

Strassen's algorithm is a recursive algorithm [3]. At some point in the recursion, once the matrices are small enough, we may want to switch from recursively calling Strassen's algorithm and just do a conventional matrix multiplication. That is, the proper way to apply Strassen's algorithm is to evade recurse on a "base case" of a 1 by 1 matrix, and switch earlier to use conventional matrix multiplication. The cross-over point between the two algorithms can be defined to be the value of n for recursive strassen's algorithm is stopped and switch to conventional matrix multiplication. Determining the cross-over point has been one pertinent issue that has deterred strassen's scheme from been considered for practical usage.

This paper seeks to implement the conventional algorithm and Strassen's algorithm and systematically determine their cross-over point by carrying out the following tasks: to use large matrices whose entries are randomly generated; to use Strassen algorithm to multiple two real n by n matrices; to stop the recursive strassen multiplication at some point n_0 and use the conventional method onward; to experimentally determine the value of n_0 that optimizes the running time of the algorithms; and to use padding technique to ensure the implementation of any square matrices, whose dimensions is not a power of 2.

2 Requirements and Methods

Matrix multiplication $C = A * B$ is the linear algebraic product of the matrices A and B [8]. More precisely, $For\ i \in N, j \in N$

$$C_{(i,j)} = \sum_{k=1}^{n} A_{(i,k)} B_{(k,j)} \quad (1)$$

For nonscalar A and B, the number of columns of A must equal the number of rows of B.

2.1 A Simple Divide and Conquer Algorithm

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub problem of the same related type until these becomes simple enough to solve directly [2]. The solutions to the sub problems are then combined to give a solution to the original problem. Basic design step:

Assuming n is a power of 2 and suppose A, B and C are divided into four $n/2$ x $n/2$ matrices.

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \text{x} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \quad (2)$$

then

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21}$$
$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22}$$
$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21}$$
$$C_{22} = A_{21} * B_{12} * A_{22} * B_{22}$$

Algorithm 1: A Divide and Conquer Algorithm

SQUARE MATRIX MULTIPLY RECURSIVE (SMMR)

$SMMR(A, B)$

1. $N = A.rows$
2. $Let\ C$ be a new n x n matrix
3. if $n = 1$
4. $C = a * b$
5. else partition A, B and C as in equation 2
6. $C_{11} = SMMR(A_{11}, B_{11}) + SMMR(A_{12}, B_{21})$
7. $C_{12} = SMMR(A_{11}, B_{12}) + SMMR(A_{12}, B_{22})$
8. $C_{21} = SMMR(A_{21}, B_{11}) + SMMR(A_{22}, B_{21})$
9. $C_{22} = SMMR(A_{21}, B_{12}) + SMMR(A_{22}, B_{22})$
10. Return

Algorithm 2: Strassen's Matrix Multiplication Algorithm (Strassen (A, B, n))

Assuming the size of the matrix is also of a power of two, the algorithm below holds;

1. if $n = 1$ output $C = A * B$
2. else
3. $m = n/2$
4. Compute $A_{11}, B_{11}, A_{12}, B_{12}, A_{21}, B_{21}, A_{22}, B_{22}$
5. $P1 = \text{Strassen}(A_{11}, B_{12} - B_{22}, \text{m})$
6. $P2 = \text{Strassen}(A_{11} + A_{12}, B_{22}, \text{m})$
7. $P3 = \text{Strassen}(A_{21} + A_{22}, B_{11}, \text{m})$
8. $P4 = \text{Strassen}(A_{22}, B_{21} - B_{11}, \text{m})$

9. $P5 = \text{Strassen}(A_{11} + A_{22}, B_{11} + B_{22}, \text{m})$
10. $P6 = \text{Strassen}(A_{12} - A_{22}, B_{21} + B_{22}, \text{m})$
11. $P7 = \text{Strassen}(A_{11} - A_{21}, B_{11} + B_{12}, \text{m})$
12. $C_{11} = P5 + P4 - P2 + P6$
13. $C_{12} = P1 + P2$
14. $C_{21} = P3 + P4$
15. $C_{22} = P1 + P5 - P3 - P7$
16. Output C
17. End If

If the matrix dimension is not 1, the matrices are divided by 2. This is the Divide part. Seven recursive calls are made for each sub-divisions. Then the Conquer stage combines the sub problems to get the overall result.

2.2 Odd Size Matrix

In order to apply a level of Strassen algorithm, the row and column size of the matrix must be of a power of two. If the matrix dimension is not to a power of two, padding is done to add or delete a particular number of rows and columns. Padding is done by embedding the input matrix in a larger matrix of zeros, Fig. 2.1

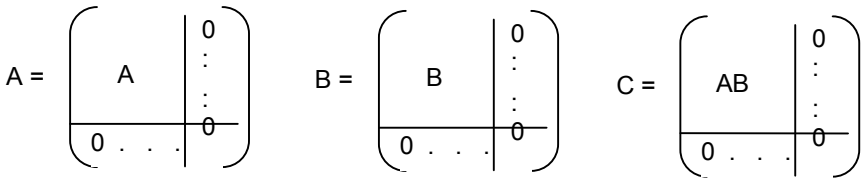

Fig. 2.1. Padding a matrix

The product C = AB would appear at the upper left block of the matrix which is independent of the extra rows and columns. After Strassen's algorithm is applied to the large matrix, the desired product is obtained by deleting the extra rows and columns. Any number of rows and columns can be added as long as the resulting matrix is of a power of two.

2.3 Implementation of Strassens Algorithm

The algorithm is implemented by writing a function and a script file that calls the function using MATLAB R2008b. The script file contains the 'input' statements used to get the test matrices, range of values for the test matrices and cut-off point from the user. The test matrices entered by the user is checked using the 'pow2' function, a MATLAB inbuilt function which returns the value of the matrix size in terms of the power of two. The implementation also requires another MATLAB inbuilt function 'nextpow2' which returns the smallest power of two that is greater than or equal to the absolute value of the size of matrix in terms of the power of two. The value returned by the 'nextpow2' is used to create a large matrix that is used during padding.

3 Procedures

The Strassen's algorithm involves two major processes: (1) The general process which involves inputting the matrix dimensions, padding, and the call to the Strassen function and (2) Strassen function which involves what happens when the Strassen function is called. The general process of how Strassen algorithm works is seen in the algorithm 3 [2].

Algorithm 3: How Strassen's Algorithm Works

1. Input the smallest dimension $n1$ of matrices A and B
2. Input the largest dimension $n2$ of matrices and A and B
3. Input the step dimension of matrices A and B
4. Input the cutoff($nmin$) for number of iterations
5. For $n = n1{:}\,step{:}\,n2$
6. Generate $A(n)$ and $B(n)$
7. If $n =$ Power of 2
8. Compute $C = Strassen(A, B, nmin)$
9. Else
10. $d =$ size of A
11. $s =$ next power of $2 > A$
12. $k = d \cup s$
13. New size of $A = k$
14. Compute $C = Strassen(A, B, nmin)$
15. Output C

The Strassen function call process is shown in the algorithm 4 [8] (See Matlab code in appendix 1.0

Algorithm 4: Strassen's Function Algorithm

1. If $n \leq nmin$
2. $C = A * B$
3. $Else$
4. $m = n/2\ ;\ i = 1{:}\,m\ ;\ j = m + 1{:}\,n$
5. $P1 = Strassen(A(I,I) + A(J,J), B(I,I) + B(J,J), nmin)$
6. $P2 = Strassen(A(J,I) + A(J,J), B(I,I), nmin)$
7. $P3 = Strassen(A(I,I), B(I,J) - B(J,J), nmin)$
8. $P4 = Strassen(A(J,J), B(J,I) - B(I,I), nmin)$
9. $P5 = Strassen(A(I,I) + A(I,J), B(J,J), nmin)$
10. $P6 = Strassen(A(J,I) - A(I,I), B(I,I) + B(I,J), nmin)$
11. $P7 = Strassen(A(I,J) - A(J,J), B(J,I) + B(J,J), nmin)$
12. Return C

MATLAB program (see appendix 1.0) was written to implement algorithms 3 and 4. The execution of the program presented the interface that was used in entering data as shown on Interface 3.1

```
>> test_strassen
Enter the smallest dimension n1 of the matrices A and B 512
Enter the largest dimension n2 of the matrices A and B 4096
Enter the step dimensions of the matrices A and B 512
Enter left range of interval 0
Enter right range of interval 1
Enter Cutoff level for 512 x 512 matrix 16
Enter Cutoff level for 1024 x 1024 matrix 32
Enter Cutoff level for 1536 x 1536 matrix 64
Enter Cutoff level for 2048 x 2048 matrix 64
Enter Cutoff level for 2560 x 2560 matrix 128
Enter Cutoff level for 3072 x 3072 matrix 128
Enter Cutoff level for 3584 x 3584 matrix 128
Enter Cutoff level for 4096 x 4096 matrix 128
```

Interface 3.1. Entering data in MATLAB

4 Results and Discussion

The MATLAB program written, was run in a Core I3 Lenoz computer with 4.00GB RAM and the following result was recorded. The experiment was iteratively carried out on eight different matrices in the steps of 512 starting from 512 by 512 matrices through 4096 by 4096 matrices. The Steps dimensions (SDs) of the matrices were manually entered and the Power2 dimensions (PDs) were generated. The SDs matrices whose dimensions are not a power of 2 were padded in the generated PDs. Tables 4.1, 4.2, 4.3 and Figs. 4.1, 4.2 and 4.3 respectively show the different number of recursions performed on the matrices used for the experiments, their cross-over points and the time complexities of their corresponding conventional and Strassen algorithms.

Table 4.1. Experiment with 5 recursions

SD (rows)	PD	Values of 5 levels of recursion (divide and conquer)					CT (CPU time in seconds)	ST (CPU time in seconds)
512	512	256	128	64	32	16	0.2105	0.0670
1024	1024	512	256	128	64	32	0.1005	0.2175
1536	2048	1024	512	256	128	64	0.7710	0.7223
2048	2048	1024	512	256	128	64	0.7896	0.7194
2560	4096	2048	1024	512	256	128	6.2496	3.9763
3072	4096	2048	1024	512	256	128	6.1280	4.0011
3584	4096	2048	1024	512	256	128	6.1645	3.9764
4096	4096	2048	1024	512	256	128	6.1256	4.0169

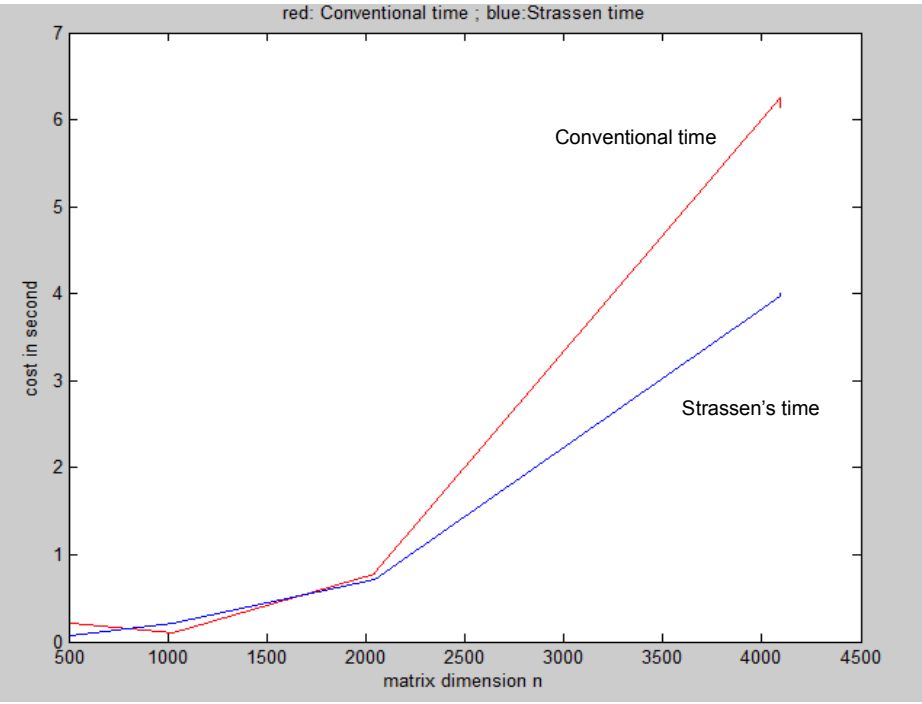

Fig. 4.1. Graph of experiment with five recursions

Table 4.1 shows that optimization n_0, the cross-over point, was made using five levels of recursions. For instance, it shows that for 512 by 512 matrices, the five values for the recursion: 256, 128, 64, 32 and 16 are the values of the divide and conquer algorithm. The last value 16 represents the cross-over point n_0 which indicates that the recursion stopped after the 16 by 16 matrices and switched over to conventional matrices to complete the strassen's time. (Note that the time used for the strassen's recursions and the conventional time after the cross-over add up to the total Strassen's time, ST). CT is the time for the execution of the conventional matrix multiplication. The table also shows that the values 32, 64, 64, 128, 128, 128 and 128 are the cross-over points n_0 for the matrix dimensions 1024, 1536, 2048, 2560, 3072, 3584 and 4096 with their respective values of divide and conquer algorithm. In the table the matrix dimension 1024 x 1024, the conventional algorithm completed the multiplication in 0.1005 seconds which is faster than Strassen's algorithm that completed the multiplication of the same matrix dimension in 0.2175. As the matrix dimensions increase to 2560 x 2560 upwards, the time taken to complete the conventional matrix multiplication of step dimensions (SDs) 2560, 3072, 3584 and 4096 are 6.2496, 6.1280, 6.1645 and 6.1256, whereas, Strassen's matrix multiplication completed the same SDs at 3.9763, 4.0011, 3.9764 and 4.0169. This shows that for sufficiently large values of n in thousands, Strassen's algorithm will run faster and offer an improvement than the conventional algorithm. The graph in Figure 4.1 indicates that the lines for the conventional algorithm completed the multiplication faster (lesser seconds) at the matrix dimensions of 512 x 512 and 1024 x 1024 up to 1536 x 1536 than the lines for Strassen's algorithm. As the matrix dimensions increase above 1536 x 1536, the line for Strassen's algorithm completed the multiplications in lesser seconds.

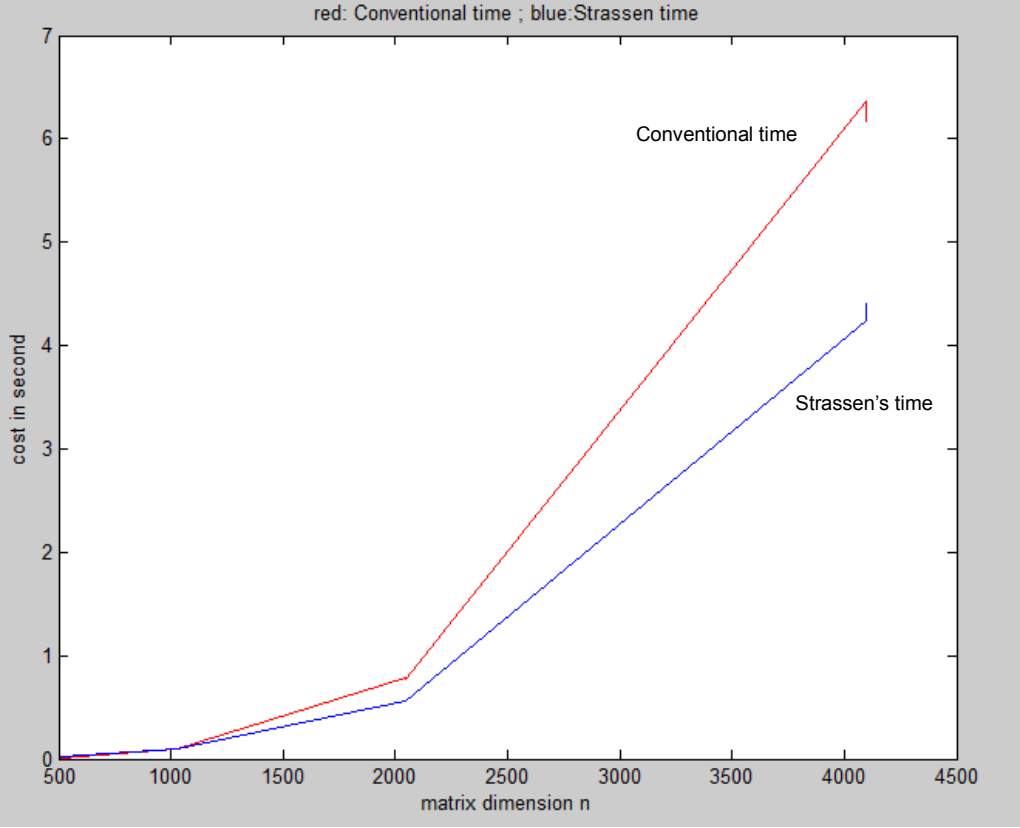

Fig. 4.2. Graph of experiment with four recursions

Table 4.2. Experiment with 4 recursions

SD (rows)	PD	Values of 4 levels of recursion (divide and conquer)				CT (CPU time in seconds)	ST (CPU time in seconds)
512	512	256	128	64	32	0.0140	0.0271
1024	1024	512	256	128	64	0.1004	0.0953
1536	2048	1024	512	256	128	0.7862	0.5593
2048	2048	1024	512	256	128	0.7722	0.5652
2560	4096	2048	1024	512	256	6.3685	4.2340
3072	4096	2048	1024	512	256	6.2326	4.4239
3584	4096	2048	1024	512	256	6.1589	4.3210
4096	4096	2048	1024	512	256	6.1807	4.2651

Tables 4.2 shows that four levels of recursions were used for the cutoff points and that there are additional costs in the time to complete the corresponding matrix multiplications. The graph in Fig. 4.2 shows that when the cutoff point undergoes four recursions, strassens algorithm performs faster even with the matrix dimensions of 512 by 512.

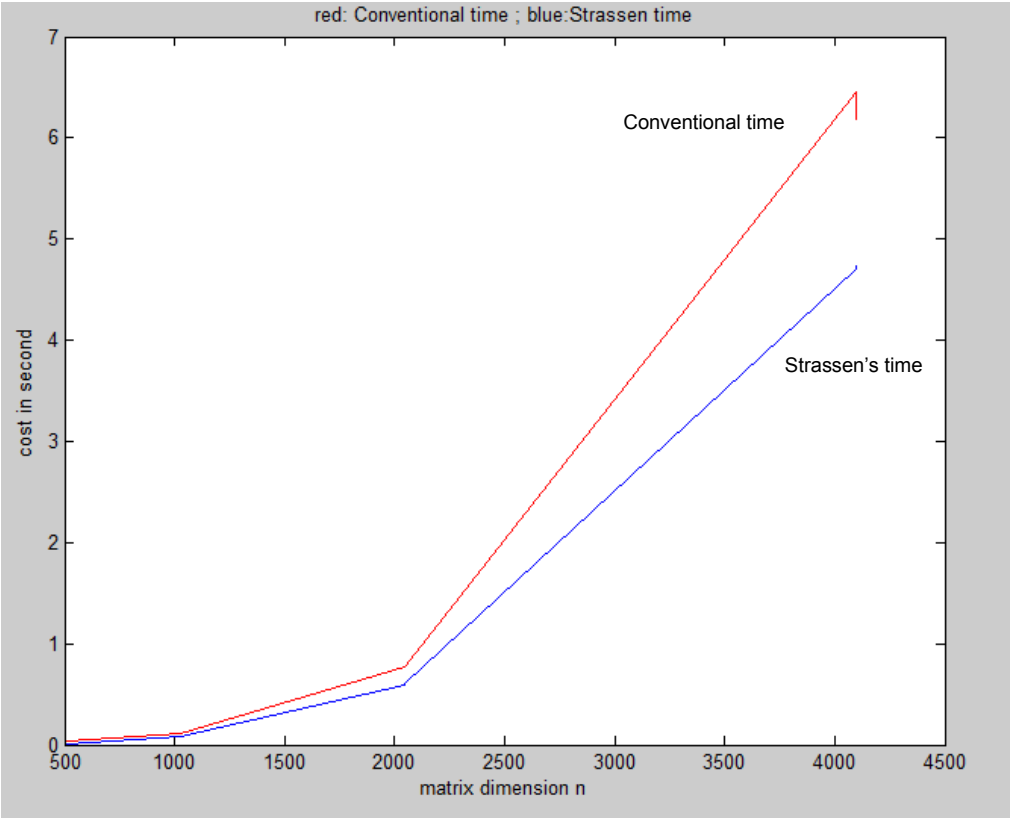

Fig. 4.3. Graph of experiment with three recursions

Table 4.3. Experiment with 3 recursions

SD	PD	Values of 3 levels of recursion (divide and conquer)			CT (in seconds)	ST (in seconds)
512	512	256	128	64	0.0315	0.0137
1024	1024	512	256	128	0.1048	0.0793
1536	2048	1024	512	256	0.7753	0.5959
2048	2048	1024	512	256	0.7773	0.6107
2560	4096	2048	1024	512	6.4553	4.6971
3072	4096	2048	1024	512	6.2016	4.7111
3584	4096	2048	1024	512	6.1695	4.7505
4096	4096	2048	1024	512	6.1806	4.7355

Tables 4.3 shows that three levels of recursions were used for the cutoff points and that there are even more additional costs in the time to complete the corresponding matrix multiplications. The graph in Fig. 4.3 shows that when the cutoff point undergoes three recursions, Strassens algorithm performs much faster even with the matrix dimensions of 512 by 512.

5 Conclusion

As shown from the graph, Strassen's algorithm definitely performs better than the conventional algorithm for multiplying matrixes with large dimensions. If the division proceeds to the level of single matrix elements, then it means a large additional temporary storage is required therefore, reducing the potential advantage in performance. This overhead is generally limited by stopping the recursion early and performing the conventional matrix multiplication on sub matrices at the crossover point. From the above result evaluation, if a level of Strassen algorithm is efficient, it should be applied. Therefore as the matrix size increases, it is important to increase the number of recursions it undergoes to reduce the time complexity and enhance performance.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Popor S. Algorithm of the week: Strassen's matrix multiplication; 2011. Retrieved December, 29, 2013, from the architects. Available: www.architects.dzone.com

[2] Malik DM. Design and analysis of algorithm. Lahore: COMSATS Insititute of Technology; 2011.

[3] Harvard. CS E-124–Spring. Programming Assignment 2; 2014. Retrieved August, 20, 2014. Available: www.fes.harvard.edu/ ... CS/prog2ext.pdf

[4] Junjie L, Sanjay R, Sartaj S. Strassen's matrix multiplication on GPUs; 2012. Retrieved October, 15 2014. Available: www.cise.ufl.edu/.../strassen.pdf

[5] Ezugwu E. Absalom, Abdullahi M, Sani B, Junaidu B. Sahalu. Application of Strassen's algorithm in Rhotrix row-column multiplication; 2011. Available: http://www.ncs.org.ng Retrieved on Nov, 2014

[6] Hedtke I. Strassen's matrix multiplication algorithm for matrices of arbitrary order. Lahore: COMSATS Insititute of Technology; 2011.

[7] David B, King L, Horst S. Using Strassen's algorithm to accelerate the solution of linear systems. The Journal of Supercomputing. 1990;4:357-371.

[8] Matlab Central. The matrix computational toolbox – file exchange; 2014. Retrieved November, 18, 2013. Available: http://www.mathworks.com/matlabcentral

Appendix

Appendix 1.0. Sample codes implemented in math lab

```
n1 = input ('Enter the smallest dimension n1 of the matrices A and B ');
n2 = input ('Enter the largest dimension n2 of the matrices A and B ');
step =  input ('Enter the step dimensions of the matrices A and B ');
a = input ('Enter left range of interval ');
b = input ('Enter right range of interval ');
%build levels
l = zeros (4,1);
l (1,1) = input('Enter Cutoff level for 512 x 512 matrix ');
l (2,1) = input('Enter Cutoff level for 1024 x 1024 matrix ');
l (3,1) = input('Enter Cutoff level for 1536 x 1536 matrix ');
l (4,1) = input('Enter Cutoff level for 2048 x 2048 matrix ');
l (5,1) = input('Enter Cutoff level for 2560 x 2560 matrix ');
l (6,1) = input('Enter Cutoff level for 3072 x 3072 matrix ');
l (7,1) = input('Enter Cutoff level for 3584 x 3584 matrix ');
l (8,1) = input('Enter Cutoff level for 4096 x 4096 matrix ');
%
j = 0;
size = round((n2-n1)/step+1);
matrix_size = zeros (size,1);
temps_con = zeros (size,1);
temps_str = zeros (size,1);
for n = n1:step:n2,
   j = j + 1;
   A = (b-a)*rand (n)+a;
   B = (b-a)*rand (n)+a;
   %Check if n is a power of 2
   s = nextpow2(n);
   power2s = pow2(s);
   if power2s == n
      matrix_size(j) = n;
      tic;
      C = A*B;
      temps_con(j) = toc;
      clear C;
      %mi = input ('Enter level to switch to conventional multiplication matrix ');
      mi = l (j,1);
      [S,tstras] = strassen (A, B, mi);
      temps_str(j) = tstras;
      clear S;
      clear A;
      clear B;
   else
      m = power2s;
      matrix_size(j) = m;
      AP = zeros(m);
      AP(1:n,1:n) = A;
      clear A;
      BP = zeros(m);
      BP (1:n,1:n) = B;
```

```
      clear B;
      tic;
      CP = AP * BP;
      temps_con(j) = toc;
      clear CP;
      %mi = input ('Enter level to switch to conventional multiplicationmatrix ');
      mi = l(j,1);
      [SP,tstras] = strassen (AP, BP, mi);
      temps_str(j) = tstras;
      clear SP;
      clear AP;
      clear BP;
   end;

end;
plot(matrix_size(1:j),temps_con(1:j),'r',matrix_size(1:j),temps_str(1:j),'b');
xlabel('matrix dimension n');
ylabel('cost in second');
title('red: Conventional time ; blue: Strassen time');
matrix_size(1:j),temps_con(1:j), temps_str(1:j),l
```

Listing 1: Main script codes

```
function [C, strastime] = strassen (A, B, nmin )
strastime = 0;
n = length(A);
if n <= nmin
   tic;
   C = A*B;
   strastime = toc;
else
   m = n/2; i = 1:m; j = m+1:n;
   [P1,tstra1] = strassen (A(i,i)+A(j,j), B(i,i)+B(j,j), nmin);
   [P2,tstra2] = strassen (A(j,i)+A(j,j), B(i,i), nmin);
   [P3,tstra3] = strassen (A(i,i), B(i,j)-B(j,j), nmin);
   [P4,tstra4] = strassen (A(j,j), B(j,i)-B(i,i), nmin);
   [P5,tstra5] = strassen (A(i,i)+A(i,j), B(j,j), nmin);
   [P6,tstra6] = strassen (A(j,i)-A(i,i), B(i,i)+B(i,j), nmin);
   [P7,tstra7] = strassen (A(i,j)-A(j,j), B(j,i)+B(j,j), nmin);
   C = [ P1+P4-P5+P7  P3+P5;  P2+P4  P1+P3-P2+P6 ];
strastime = strastime + tstra1 + tstra2 + tstra3 + tstra4 + tstra5 + tstra6 + tstra7;
end
```

Listing 2: Strassen function call codes

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/10079