A Hybrid Approach on Matrix Multiplication

Department of Computer Engineering, Hankuk Aviation University
Goyang City, Korea
{tolentinobel[O], kimmk, chae}@hau.ac.kr

KISS Korea Computer Congress 2006

Maribel Tolentino[O], Myungkyu Kim, Soo-Hoan Chae
Hankuk Aviation University

Abstract

Matrix multiplication is an important problem in linear algebra, its main significance for combinatorial algorithms is its equivalence to a variety of other problems, such as transitive closure and reduction, solving linear systems, and matrix inversion. Thus the development of high-performance matrix multiplication implies faster algorithms for all of these problems. In this paper, we present a quantitative comparison of the theoretical and empirical performance of key matrix multiplication algorithms and use our analysis to develop a faster algorithm. We propose a Hybrid approach on Winograd's and Strassen's algorithms that improves the performance and discuss the performance of the hybrid Winograd-Strassen algorithm. Since Strassen's algorithm is based on a 2 x 2 matrix multiplication it makes the implementation very slow for larger matrix because of its recursive nature. Though we cannot get the theoretical threshold value of Strassen's algorithm, so we determine the threshold to optimize the use of Strassen's algorithm in nodes through various experiments and provided a summary shown in a table and graphs.

1. Introduction

Matrix multiplication is an important problem in linear algebra, its main significance for combinatorial algorithms is its equivalence to a variety of other problems [8]. Matrix multiplication is also essential in applied fields, such as computer graphics and digital signal processing (DSP)[9]. DSP chips are found in devices such as mobile phones, multimedia computers, video recorders, hard disk drive controllers, digital cameras, and modems, as matrix operations are the processes by which DSP chips are able to digitize sounds or images so that they can be stored or transmitted electronically.

The fast matrix multiplication algorithm is still an open problem but the more common area of development is the implementation of existing algorithms rather than the design of new algorithms as shown in [6]. Matrix multiplication is the core of many scientific applications. Some algorithms substitute multiplications by additions and thus reduce the number of multiplications computed as discussed in [1][2][3][5].

Strassen and Winograd are such algorithms that are best suited for a practical implementation. In the case of multiplying two 2×2 matrices, Strassen's algorithm is an improvement over the naive algorithm. It uses only seven scalar multiplications as opposed to the usual eight. Even though it has been shown that Strassen's algorithm is optimal for two-by-two matrices[10], there have been asymptotic improvements to the algorithm for very large matrices.

Thus, the search for improvements over Strassen's algorithm for smaller matrices is still being conducted. Even Strassen's algorithm is not considered an efficient reduction as it requires the size of the multiplicand matrices to be large powers of two.

2. Related Study

This section presents the naive, Winograd's, and Strassen's algorithms along with discussions of the theoretical bounds for each algorithm. Then, we present a confirmation of the theoretical study on the running times of the algorithms, followed by the results of an empirical study. We present an improved Hybrid algorithm, which incorporates. Strassen's asymptotical advantage with Winograd's practical performance. Then, we present the optimum threshold value that can be used when using Strassen's algorithm on a single node through experiments and provided a summary shown in a table and graphs.

2.1. Naïve Algorithm

The naive algorithm is solely based on the familiar mathematical definition for the multiplication of two matrices as shown in Equation (1). To compute each entry in the final $n\times n$ matrix, we need exactly n multiplications and $n - 1$ additions. by counting the number of multiplications and additions in this code segment it is clear that this standard code the total number of multiplications is $n\times n^2 = n^3$, and the total number of additions is $(n - 1)\cdot n^2 = n^3 - n^2$. Thus, we classify the naive algorithm as an $O(n^3)$ algorithm, because its running time increases as the cube of the given parameter n, so doubling the size of n will increase the number of multiplications eight-fold. While the algorithm performs n^3 scalar multiplications, the input itself is on the order of n^2. So, the naive algorithm has an almost linear running time with the size of the input.

2.2. Winograd's Algorithm

Winograd's algorithm trades multiplications for additions as described in [1], much like Strassen's method. Nevertheless, it is asymptotically the same as the naive algorithm. Winograd's algorithm uses pairwise multiplication of couples of entries instead of multiplying individual numbers as in the naive algorithm, and then subtracts the accumulated error. Winograd's algorithm is defined as shown in Equations (2), (3), and (4) as demonstrated in [1]. Since A_i and B_j are pre-computed only once for each row and column, they require only n^2 scalar multiplications. The final summation does require $O(n^3)$ multiplications, but only half of those in the naive algorithm. Thus, the total number of scalar multiplications has been reduced to $\frac{1}{2} n^3 + n^2$. However, the number of additions has been increased by $\frac{1}{2} n^3$. Winograd's algorithm is theoretically faster than the naive algorithm, computers add faster than they multiply, while the total number of operations remains almost unchanged.

Naive Algorithm

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} * b_{k,j} \qquad \text{Eq. (1)}$$

Winograd Algorithm

$$A_i = \sum_{k=1}^{n/2} a_{i,2k-1} * a_{i,2k} \qquad \text{Eq. (2)}$$

$$B_j = \sum_{k=1}^{n/2} b_{2k-1,j} * b_{2k,j} \qquad \text{Eq. (3)}$$

$$c_{i,j} = \sum_{k=1}^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - A_i - B_j \qquad \text{Eq. (4)}$$

Strassen Algorithm

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} \qquad \text{Eq. (5)}$$

2.3. Strassen's Algorithm

Strassen used the Divide-and-Conquer Technique. The Idea behind this technique was apparently named for a military maneuver[7]. The division into several parts and solving each part is the divide, while the combining of the individual solutions into a global solution is the conquer phase. Strassen devised a clever method of dividing the given matrices into four sub-matrices and then recursively multiplying them to obtain the resultant matrix. First, the two matrices are divided into four quadrants, or submatrices, of size $n/2$ by $n/2$. Strassen's method uses only seven multiplications and eighteen additions for each recursive call in order to compute the final matrix product. It is relatively simple to show that the number of multiplications for an n-by-n matrix is $n^{2.81}$, and the number of additions is $6n^{2.81} - 6n^2$.

Operation	Naïve	Winograd	Strassen
Add/Sub	$n^3 - n^2$	$3/2\, n^3 + 2n^3 - 2n$	$6n^{2.81} - 6n^2$
Multiplication	n^3	$1/2\, n^3 + n^2$	$n^{2.81}$

Figure 1. Summary of the operations of the three classical algorithms.

3. Empirical Study

The theoretical results were consistent with the asymptotic models of the algorithms. The naive algorithm always had the lowest number of scalar additions with the largest number of scalar multiplications. Figure 2 and Figure 3 compare the three algorithms in terms of the number of additions and the number of multiplications as a function of the matrix parameter n. Figure 1 shows the comparison of the number of arithmetic operations done by the three matrix multiplication methods given n x n matrices. For large n, Strassen's algorithm does fewer multiplications and fewer additions and subtractions that either of the other methods as seen in Figure 2 and Figure 3. In practice, however, because of its recursive nature, implementing this algorithm requires a lot of bookkeeping that might be slow and is complicated [1]. Both naive and Winograd's algorithms were implemented very easily. The empirical tests concur with the theoretical derivations made earlier. Both algorithms appear to run as the cube of n, but Winograd's algorithm is faster by a constant because of the tradeoff between additions and multiplications. Figure 7 illustrates the comparative runtimes of the $O(n^3)$ algorithms.

Since, Strassen's algorithm is based on recursion, it presented problems. Despite its lower theoretical run-time complexity of $O(n^{2.807})$, it still took three times longer to execute than the $O(n^3)$ algorithms. This is a result of the large number of stack operations caused by the extensive recursion of Strassen's algorithm, which requires large addressing headers in order to contain all four sub-matrix pointers.

For the purpose of these performance comparisons, we conducted tests primarily on a personal desktop computer-Intel Pentium 4 CPU 2.40 GHz with 1.25 GB of RAM. Figure 5 presents the running times in seconds of the algorithms as a function of the matrix parameter n. The polynomial behavior of the algorithms is clearly shown here. We should also take note that these specific run-time approximations depend on the processor speed.

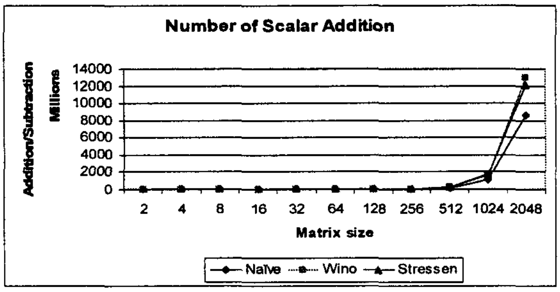

Figure 2. Theoretical Comparison of Scalar Addition

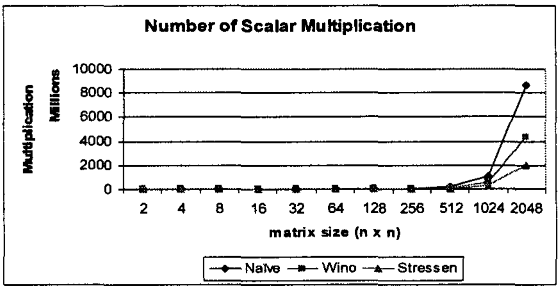

Figure 3. Theoretical Comparison of Scalar Multiplication.

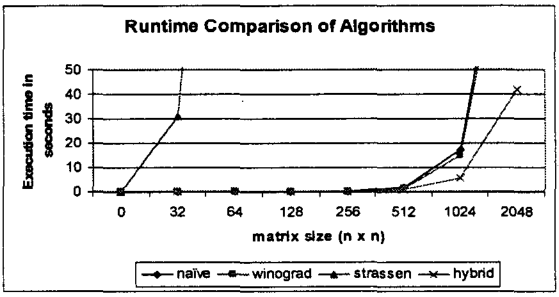

Figure 4. Runtime comparison of four matrix multiplication algorithms.

4. Hybrid Algorithm

Due to the large number of recursive calls, Strassen's algorithm performs more slowly than expected in the empirical tests mainly which theoretically should not take up too much time, but in fact account for most of the running time as seen on Figure 7. Since Strassen's algorithm is based on a 2 x 2 matrix multiplication, the bookkeeping after each recursion makes the algorithm very slow. In view to this problem, we proposed a Hybrid algorithm in the search for a more efficient matrix multiplication, which uses the Strassen's method until a predetermined cutoff size of the seven sub-matrices, after which Winograd's algorithm takes over. For matrices of smaller sizes, for instance when matrix size n is less than 16, has no significant difference in cost between multiplications and additions of submatrices. Instead, naïve algorithm is likely to outperform both Winograd's and Strassen's algorithm. The most intuitive hybrid algorithm to take advantage of this is a Strassen variant where at certain level the recursion is stopped and a simpler but faster matrix multiplication algorithm is called instead. In this paper, Winograd's algorithm is considered to be simpler algorithm. The main parameter to be investigated in this case is the breakpoint or threshold size. Other computationally intensive tasks such as sorting also use this kind of optimization. The theoretically faster quick-sort algorithm performs better when the recursion is stopped and the slower but simpler insertion sort is used to complete the sorting. Likewise, we reduce the number of recursive calls exponentially while still taking advantage of Strassen's reduced number of multiplications in this hybrid algorithm.

Figure 6 shows the pseudo code of the hybrid algorithm. The outline of the code is similar to Strassen's algorithm but the proposed hybrid switches to Winograd's algorithm at a given threshold. The critical level of recursion before switching to Winograd is related to the Hybrid algorithm's performance. It depends primarily on the starting value for *n*. It has been found that a larger *n* would yield a more significant speedup to the matrix multiplication. The empirical tests of the implementation show that the resultant Winograd-Strassen hybrid algorithm performs significantly better than both of its predecessors. Although theoretically, the Hybrid algorithm performs almost $O(n^3)$ multiplications which is reduced by the use of Strassen's algorithm, with a smaller scaling constant which is due to the use of Winograd's algorithm, it is empirically faster than the original Winograd's algorithm by small percentage.

```
void Hybrid ( matrix<ClassName> A[ra1..ra2,ca1..ca2],
              matrix<ClassName> B[rb1..rb2,cb1..cb2],
              matrix<ClassName> C[rc1..rc2,cc1..cc2]){
    integer n = ra2 - ra1 + 1;
    if(n==threshold){
              Compute C=A*B using the Winograd's
                          algorithm from Figure 4;
    }
    else{
      Partition A into submatrices A11, A12, A21, A22;
      Partition B into submatrices B11, B12, B21, B22;
      Partition C into submatrices C11, C12, C21, C22;
      Define square matrices M1, M2, M3, M4, M5, M6, M7;
      // Compute A*B using Strassen's algorithm
         from Figure 5
      Hybrid(A11 + A22, B11 + B22, M1)
      Hybrid(A21 + A22, B11, M2)
      Hybrid(A11, B11 - B22, M3)
      Hybrid(A22, B21 - B11, M4)
      Hybrid(A11 + A12, B22, M5)
      Hybrid(A21 - A11, B11 + B12, M6)
      Hybrid(A12 - A22, B21 + B22, M7)
      C11 = M1 + M4 - M5 + M7
      C12 = M3 + M5
      C21 = M2 + M4
      C22 = M1 + M3 - M2 + M6
    }
}
```

Figure 6. Pseudo code of the Hybrid Algorithm

5. Optimum Threshold

Divide-and-conquer algorithms are naturally implemented as recursive procedures. In that case, the partial sub-problems leading to the one currently being solved are implicitly stored in the procedure call stack. Due to the large number of recursive calls, Strassen's algorithm performs more slowly than expected. Technically speaking, C++(which was used for the implementation) arranges the memory spaces needed for each function call in a stack. The memory area for each new call is placed on the top of the stack, and then taken off again when the execution of the call is completed. At a certain level the recursion can be stopped and a simpler but faster algorithm is called. The main parameter to be investigated in this case is the breakpoint or threshold size. Other computationally intensive tasks such as sorting also use this kind of optimization.

In this study, we tried to identify the optimum threshold in using Strassen's algorithm based on our observation from the results. We restrict our attention here to square matrices whose dimension is a power of two. The determinations of the optimum threshold are presented through a table and graphs. Along with the experiment, we used two separate nodes for comparison purposes. System A is using Pentium 4 CPU 3.00 GHz with 512 Gb RAM while System B is using Xeon ™ CPU 3.00 GHz with 2.0 Gb RAM. Figure 7 and Figure 8 show that the nearer the points to zero second the more optimal it is when implementing Strassen's algorithm in matrix multiplication. As it is observed from the graphs, the algorithm is optimal when the threshold is 2^5 and 2^6. Figure 10 shows the detailed execution time consumed by the given matrix sizes. Though our experiment shows these results, the values might vary in a small percentage if implemented to a different kind of system. Our Hybrid algorithm can perform well if the optimum threshold value will be used in the given matrix sizes.

6. Conclusion

The result of the study leads to conclude that our Hybrid algorithm is more efficient compared to the three classical algorithms in terms of matrix multiplication. We found out that Strassen's algorithm can still be optimized in large matrix-matrix multiplication with a size of n x n by declaring a larger parameter for the threshold before switching to a simpler matrix multiplication algorithm. Though many studies are going on the search for faster algorithm on clusters of workstations, we can conclude that matrix multiplication on square matrices is improved by using our hybrid approach in a single node. Due to large demand of matrix multiplication in our daily life, it will be beneficial for us to use and improve the implemented hybrid Winograd-Strassen algorithm. In view of the fact that iterative solution on linear systems needs a lot of matrix multiplication, hybrid algorithm can be applied to it.

<Acknowledgement>
본 논문은 산업자원부 지정 경기도 지역협력연구센터인 한국항공대학교 인터넷정보검색연구센터의 지원에 의한 것임.

7. References

[1] Sara Baase and Allen Van Gelder (2000), Computer Algorithms Introduction to Design & Analysis, Addison Wesley Longman, Inc.
[2] William H. Press, et. al.(2002), Numerical Recipes in C++, Cambridge University Press.
[3] Sartaj Sahni (1998), Data Structures, Algorithms, and Applications in C++, WCB McGraw-Hill.
[4] Jeri R. Hanly and Elliot B. Koffman (2001), C Program Design for Engineers Second Edition, Addison Wesley Longman, Inc.
[5] Ellis Horowitz, et. al. (1997), Computer Algorithms/C++, Computer Science Press.
[6] Steven Huss-Ledeman, et al., "Implementation of Strassen's Algorithm for Matrix Multiplication." http://www.supercomp.org/sc96/proceedings/SC96PROC/JACOBSON/INDEX.HTM.
[7] Bruce P. Hillam, "Introduction to Algorithms: An Intuitive Object Oriented Approach", http://www.csupomona.edu/~bphillam/algotext/pdfpages/cho.pdf
[8] Richard L. Burden and J. Douglas Faires (2001), Numerical Analysis Seventh Edition, Brooks/Cole Thomson Learning Inc.
[9] D. Koenig, "Digital Signal Processing Fundamentals", http://www.sss-mag.com/pdf/sigdsp.pdf

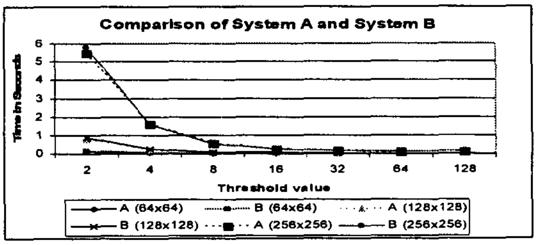

Figure 7. Comparison of System A and B using Strassen Algorithm in small sized matrix

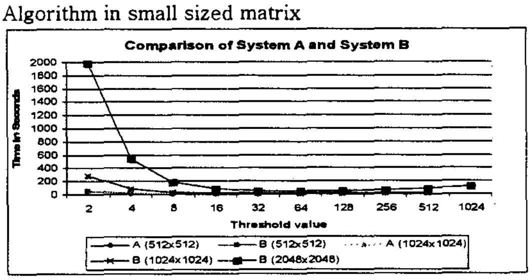

Figure 8. Comparison of System A and B using Strassen Algorithm in larger matrix size.