Comparative Study of Cache Utilization for Matrix Multiplication Algorithms

Vikash Kumar Singh#1, Hemant Makwana*2, Richa Gupta #3

1Vikash125@gmail.com
2hemanturl@yahoo.com
3richa_tg@rediffmail.com

Abstract— In this work, the performance of basic and strassen’s matrix multiplication algorithms are compared in terms of memory hierarchy utilization. The problem taken here is MATRIX MULTIPLICATION (Basic and Strassen’s). Strassen’s Matrix Multiplication Algorithm has time complexity of O(n^{2.807}) with respect to the Basic multiplication algorithm with time complexity of O(n^3). This slight reduction in time makes Strassen’s Algorithm seems to be faster but introduction of additional temporary storage makes Strassen’s Algorithm less efficient in space point of view. Access patterns of the two multiplication algorithms are generated and then cache replacement algorithms (namely LRU and FIFO) are applied to find the misses in cache. With the number of misses in hand and time taken to process 1 miss taken from Hou Fang’s research, we calculate the overall time consumed to process misses in case of both the matrix multiplication algorithms. It is found that the basic matrix multiplication is far better than the Strassen’s matrix multiplication algorithm because of higher memory usage. This analysis is important because memory plays more vital role in deciding the efficiency of an algorithm. Additional temporary storage causes increase in number of memory locations and number of memory accesses too in case of strassen’s algorithm.

Keywords:— Matrix Multiplication, Strassen’s Multiplication, LRU, FIFO, Access pattern.

I. INTRODUCTION

Matrix Multiplication is a basic algorithm which has been commonly used in areas like Graph theory, Probability theory and statistics, Electronics etc. The problem of matrix multiplication is a classical example taken to analyse the effectiveness of techniques used to improve the memory utilization. The core work is to analyse the basic matrix multiplication (row to column multiplication) and Strassen’s matrix multiplications. This work is done using access patterns of Matrix Multiplications. The basic Motivation of the present work is to compare the performances of memory system when both methods of matrix multiplication are executed.

Algorithms for Matrix Multiplication:

1. Basic Matrix Multiplication Algorithm:
Suppose we want to multiply two matrices of size N x N: for example A x B = C, then T (N) = O (N^3). The flow of Basic Matrix Multiplication algorithm is shown in Fig 1. Eight recursive matrix multiplication calls and four matrix additions are performed in this algorithm to calculate the multiplication result.

\[
\begin{array}{c|c|c|c} 
| C_{11} & C_{12} | & | A_{11} & A_{12} | & | B_{11} & B_{12} |
\end{array}
\]

\[
\begin{align*}
C_{11} &= a_{11}b_{11} + a_{12}b_{21} \\
C_{12} &= a_{11}b_{12} + a_{12}b_{22} \\
C_{21} &= a_{21}b_{11} + a_{22}b_{21} \\
C_{22} &= a_{21}b_{12} + a_{22}b_{22}.
\end{align*}
\]

Fig 1. Basic Matrix Multiplication.

2. Strassen’s Matrix Multiplication Algorithm:
Here, Divide-and-conquer is a general algorithm design paradigm:

- Divide: divide the input data S in two or more disjoint subsets S_1, S_2, ...
- Recur: solve the sub problems recursively
- Conquer: combine the solutions for S_1, S_2, ..., into a solution for S.

Suppose we have to perform multiplication on two matrices A and B, for example A x B = C, then T(N) = O(N^{2.807}). Fig. 2 shows the basic flow of Strassen’s Matrix Multiplication algorithm. It takes 7 multiplications w.r.t 8 in basic at a cost of 18 additions/subtractions.

\[
\begin{align*}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) & P_4 &= (A_{21} + A_{22}) * B_{11} \\
P_2 &= A_{11} * (B_{12} - B_{22}) & P_5 &= A_{21} * (B_{21} - B_{11}) \\
P_3 &= (A_{11} + A_{22}) * B_{22} & P_6 &= (A_{21} - A_{11}) * (B_{11} + B_{12}) \\
P_7 &= (A_{12} - A_{22}) * (B_{21} + B_{22}) & P_8 &= (A_{21} + A_{22}) * B_{11} \\
C_{11} &= P_1 + P_4 - P_5 + P_7 & C_{12} &= P_3 + P_5 \\
C_{21} &= P_2 + P_4 & C_{22} &= P_1 + P_3 - P_2 + P_6.
\end{align*}
\]

Fig 2. Strassen’s Matrix Multiplication.

From these two matrix multiplication algorithms this is clear that Strassen’s algorithm with time complexity O(n^{2.806}) is better than Basic with time complexity O(n^3) in Processing time point of view. Now, we have to see the performances of these two algorithms on cache memory in this research. With Strassen’s algorithm in hand, we find that the time complexity is reduced to a good extent. This seems to be a reason for joy but if we think another side of the coin i.e. memory hierarchy
analysis which is far more important to consider because any program’s execution is largely determined by the memory stall cycles rather than the processor’s time. Minimizing the average data access time is prime factor to be considered while designing high performance machines.

**Cache Replacement Algorithms**: There are several cache replacement algorithms suggested mainly to get maximum hit ratio. The most crucial challenge of a replacement strategy is to choose which data would be optimum to evict in order to replace it by the new one. A number of experiments are done and several replacement algorithms are proposed

a) **Least Recently Used (LRU)**: This algorithm swaps the least recently used page/word/block with the new page/word/block. This algorithm basically keeps track of all previous accesses and replaces that block that was used least recently.

b) **First In First Out (FIFO)**: This algorithm simply replaces the first entered block from cache. When a page needs to be replaced, the page at the front of the queue (the oldest page) is selected. While FIFO is cheap and intuitive, it performs poorly in practical application. Thus, it is rarely used in its original form.

c) **Optimal (OPT)**: This algorithm is theoretically best strategy to decide which existing page/word/block to be replaced by the pre-fetched one. This strategy can work only work when

d) **Clock (CLK)**: This includes the benefits of both LRU and FIFO algorithms.

e) **Random**: This is the simplest replacement algorithm. In this, A candidate item is randomly selected and is replaced by the pre-fetched item.

f) **Most Recently Used (MRU)**: This strategy is quite opposite to LRU. In this, it is assumed that the most recently used item will not be used again in recent future so should be discarded.

**II. LITERATURE SURVEY**

Volker Strassen[3] published the Strassen’s algorithm in 1969 which has time complexity of $O(N^{2.807})$ in comparison to Basic MM algorithm’s $O(N^{3})$. This method relies on recursively breaking down each of the matrices that are to be multiplied into 4 sub-matrices, and performing arithmetic operations on them. This algo reduces the number of multiplications from 8 to 7 with 18 additions. This reduction in time complexity of strassen’s algo results in the introduction of additional temporary storage which makes this less efficient in terms of memory. In 1990, Coppersmith-Winograd’s [4] variant of Strassen’s algorithm uses seven recursive matrix multiplication calls and 15 matrix additions/subtractions. Coppersmith-Winograd reduced the time complexity further to $O(N^{2.376})$ [4]. Various Cache page/word/block replacement algorithms have been introduced. These can be based either on recency or frequency. Some of the algorithms are LRU (Least recently used), FIFO (First In First Out), Optimal (OPT), LFU (Least Frequently Used), RDM (Random) and CLK (Clock based) etc. Hou Fang, Zhao Yue-Long and Hou Fang [5] in their analysis, calculated that for a given hard disk cache model the average time consumed when a cache miss occur is 12.67 ns. The performance of Basic Matrix Multiplication is further improved by designing a new algorithm by Richa Gupta [8] but there is no efficient improvement to enhance the performance of strassen’s algorithm.

**III. COMPARISON MODEL**

To analyse the behaviour of the two matrix multiplication algorithms in space point of view, the access patterns of both the matrix multiplication algorithms are generated. The access patterns themselves show the huge increment in the no. of memory locations as well as no. of memory accesses in the strassen’s algorithm. After getting the access patterns of both the algorithms, replacement algorithms (LRU and FIFO) are applied for different cache frame sizes. This work shows the comparative performance of cache systems in Basic and Strassen’s matrix multiplication algorithms. Performance analysis is done in C programming language.

To get the result, access patterns of matrix multiplication algorithms (both Basic and Strassen’s) are generated for matrix sizes 4*4 and 8*8. This access patterns are taken as input sequence for cache of varying sizes and no. of misses are calculated for different cache sizes by using replacement algorithms LRU (Least recently used) and FIFO (First In First Out).

We compare these two matrix multiplication algorithms based on the following criterion:

1. Number. Of memory units needed for the algorithm to be executed.
2. Number of total memory accesses during execution of programs.
3. Time consumed while processing after cache misses (No. of misses x time taken to process 1 miss).

**IV. PERFORMANCE ANALYSIS**

After getting the reference patterns of both Basic and Strassen’s Matrix Multiplication Algorithms for sizes 4 and 8, number of Memory locations as well as number of memory accesses are counted. Both these parameters are in a far more in number for strassen’s than those in Basic MM. Now, LRU and FIFO cache page/block/word replacement algorithms are applied on these access patterns with varying cache frame length. Number of misses are found corresponding to the both MM algorithms. Following table gives different values for number of memory units and no. of memory accesses for corresponding program.
Calculation of time consumed in processing misses:

Number of misses are calculated by implementing LRU and FIFO replacement algorithms on the access patterns of both Matrix Multiplication algorithms (Basic and Strassen’s ). The time consumed in processing the misses in cache is calculated using the analysis done by Hou Fang, Zhao Yue-Long [5]. According to their analysis, for the disk model given in following figure:

\[
\text{Average time spent in processing 1 miss} = 12.67 \times \text{no. of misses in the corresponding algorithm.}
\]

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)

e.g.: For Basic Matrix multiplication of matrix sizes 4*4, number of cache misses (when LRU is applied on cache frame size is 20) = 136.

So the times taken to process 136 misses = \(12.67 \times 136 \text{ ms. i.e. 1723 ms approx.}\)

Following graphs show the relative performances of the two matrix multiplication algorithms based on the time taken to process cache misses.

Fig 6: Time consumed in processing misses when replacement algorithm is LRU for Matrices of sizes 4.

Fig 7: Time consumed in processing misses when replacement algorithm is FIFO for Matrices of sizes 4.

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)

e.g.: For Basic Matrix multiplication of matrix sizes 4*4, number of cache misses (when LRU is applied on cache frame size is 20) = 136.

So the times taken to process 136 misses = \(12.67 \times 136 \text{ ms. i.e. 1723 ms approx.}\)

Following graphs show the relative performances of the two matrix multiplication algorithms based on the time taken to process cache misses.

Fig 6: Time consumed in processing misses when replacement algorithm is LRU for Matrices of sizes 4.

Fig 7: Time consumed in processing misses when replacement algorithm is FIFO for Matrices of sizes 4.

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)

Thus, overall time spent during processing misses = \(12.67 \times \text{no. of misses in the corresponding algorithm.}\)
CONCLUSION

With the current analysis, it is seen that although Strassen’s Matrix Multiplication algorithms is faster than the Basic algorithm but for the memory critical machines the latter algorithm is more efficient. The efficiency of the Basic MM algorithm is due to the less number of memory locations, memory accesses required. Number of misses when the same size of cache size with same replacement algorithm is applied is found from 3 to more than 5 times higher than in Strassen’s algorithm. This gap will further be increased as the sizes of matrices grow. The memory hierarchy speed plays more vital role in deciding any system’s overall performance because processor’s speed is being enhanced day by day to a large extent so there is a need of bridging the speed gap between memory system and processor system. Therefore the algorithms with less memory requirement must be encouraged. Furthermore, since Strassen’s algorithm is based on divide-and-conquer, an implementation must handle odd-size matrices, and reduce recursion overhead by terminating the recursion before it reaches individual matrix elements. These issues make it difficult to obtain efficient implementations of Strassen's algorithm.

REFERENCES