Computer Science

Ultra-Fast Matrix Multiplication: An Empirical Analysis of Highly Optimized Vector Algorithms

The development of high-performance matrix multiplication algorithms is important in the areas of graph theory, three-dimensional graphics, and digital signal processing. We present a quantitative comparison of the theoretical and empirical performance of key matrix multiplication algorithms and use our analysis to develop a faster and more modern algorithm. Starting with the mathematical definitions of three classical algorithms (the naive algorithm, Winograd's algorithm and Strassen's algorithm), we derive their theoretical run-time complexity and then compare these theoretical values to each algorithm's real-world performance. We find that Strassen's algorithm, which is the fastest theoretically, suffers a severe performance loss due to its extensive use of recursion. Thus, we propose a Hybrid Winograd-Strassen algorithm that improves the performance of both advanced algorithms. Finally, we discuss the superior performance of algorithms when utilized on modern computational optimizations such as the AltiVec velocity engine, a highly optimized vector processing unit developed by Apple in conjunction with Motorola and IBM.

Boyko Kakaradov

Matrices play an important role in mathematics and computer science, but more importantly, they are ubiquitous in our daily lives as they are instrumental in the efficient manipulation and storage of digital data. Matrix multiplication is essential not only in graph theory but also in applied fields, such as computer graphics and digital signal processing (DSP). DSP chips are found in all cell phones and digital cameras, 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. Fast matrix multiplication is still an open problem, but implementation of existing algorithms [5] is a more common area of development than the design of new algorithms [6]. Strassen's algorithm is an improvement over the naive algorithm in the case of multiplying two 2×2 matrices, because 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 [6], 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.

The following two sections present the naive, Winograd's, and Strassen's algorithms along with discussions of the theoretical bounds for each algorithm. We then present a confirmation of the theoretical study on the running times of the algorithms, followed by the results of an empirical study. In the final two sections, we present an improved Hybrid algorithm, which incorporates Strassen's asymptotical advantage with Winograd's practical performance, and discuss the stunning performance of the AltiVec-optimized Strassen's algorithm.

Naive Algorithm

The naive algorithm is solely based on the familiar mathematical definition for the multiplication of two matrices, as shown in equation (1) on the following page. The lowercase letters on the left hand side of the equation indicate that the individual matrix components are combined in scalar multiplication. To compute each entry in the final $n \times n$ matrix, we need exactly n multiplications and $n - 1$ additions. And since each of the n^2 entries in the first matrix (A) is multiplied by exactly n entries from the second matrix (B), 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.

Advanced Algorithms

We present two famous algorithms for matrix multiplication which use a reduced number of multiplications and therefore run faster. Winograd's algorithm achieves a run-time complexity of $O(n^3)$; that is to say, as n increases, it approaches the run-time of the naive algorithm. On the other hand, Strassen's algorithm uses a peculiar property of sub-matrices to reduce the run-time complexity to approximately $O(n^{2.78})$, which means that for a large enough n, Strassen's algorithm should theoretically perform significantly faster than either of the two previous algorithms. Even though Winograd worked to improve Strassen's algorithm by reducing the number of sub-matrix additions from 18 to 15 [5], he also developed a matrix multiplication algorithm of his own, earlier than Strassen.

Winograd's Algorithm

As described in [3], Winograd's algorithm trades multiplications for additions, much like Strassen's method. However, it is asymptotically the same as the naive algorithm. Instead of multiplying individual numbers as in the naive algorithm, Winograd's algorithm uses pairwise multiplication of couples of entries and then subtracts the accumulated error. As demonstrated in [3], Winograd's algorithm is defined as shown in equations (2), (3), and (4) above.

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

$$C_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j} \quad (1)$$

Winograd's Algorithm

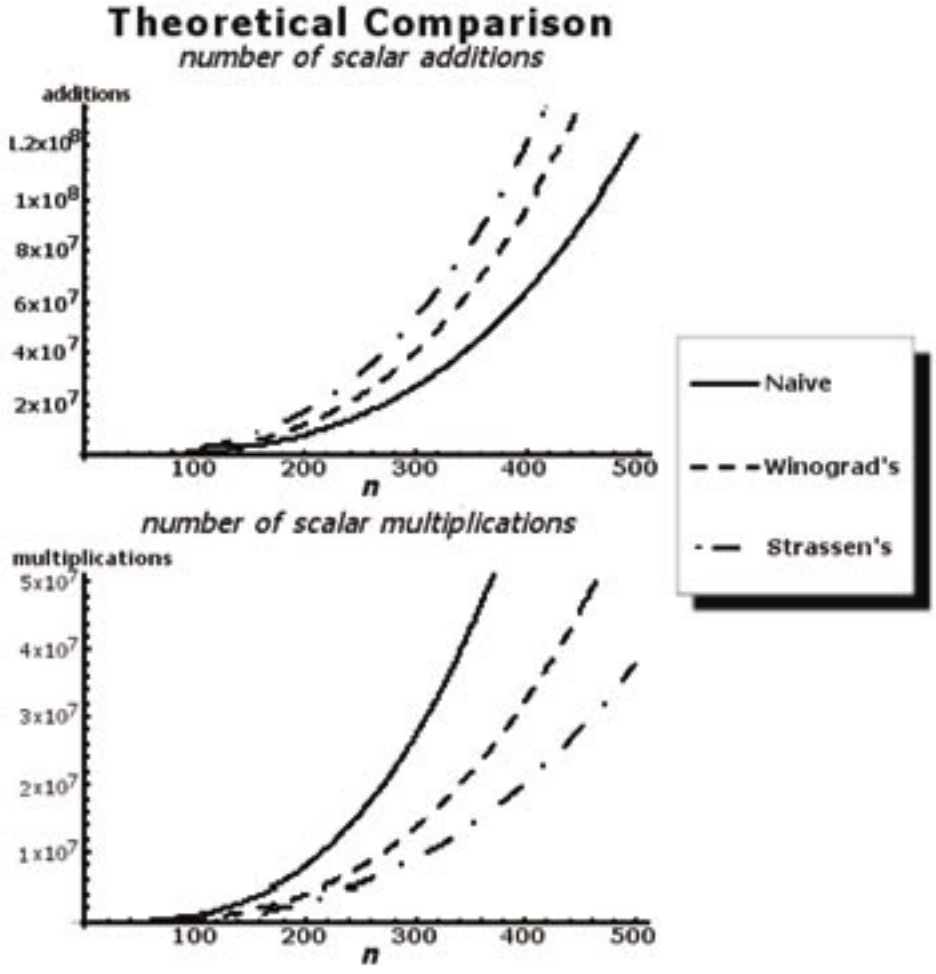
$$A_i = \sum_{k=1}^{n/2} a_{i,2k-1} \cdot a_{i,2k} \quad (2)$$

$$B_j = \sum_{k=1}^{n/2} b_{2k-1,j} \cdot b_{2k,j} \quad (3)$$

$$Ci,j = \sum^{n/2} (a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - A_i - B_j \quad (4)$$

Strassen's Algorithm

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} \quad (5)$$



naive algorithm, because computers add faster than they multiply (just as humans do), while the total number of operations remains almost unchanged.

Strassen's Algorithm

Strassen devised a clever method of dividing the given matrices into four sub-matrices and then recursively multiplying them to obtain the resultant matrix. This method is known as "divide-and-conquer" and adds not only elegance but also improved theoretical performance. First, the two matrices are divided into four quadrants, or submatrices, of size *n/2* by *n/2*. Now, via a series of intermediate variables, Strassen's method uses only seven multiplications and eighteen additions for each recursive call in order to compute the final matrix product.

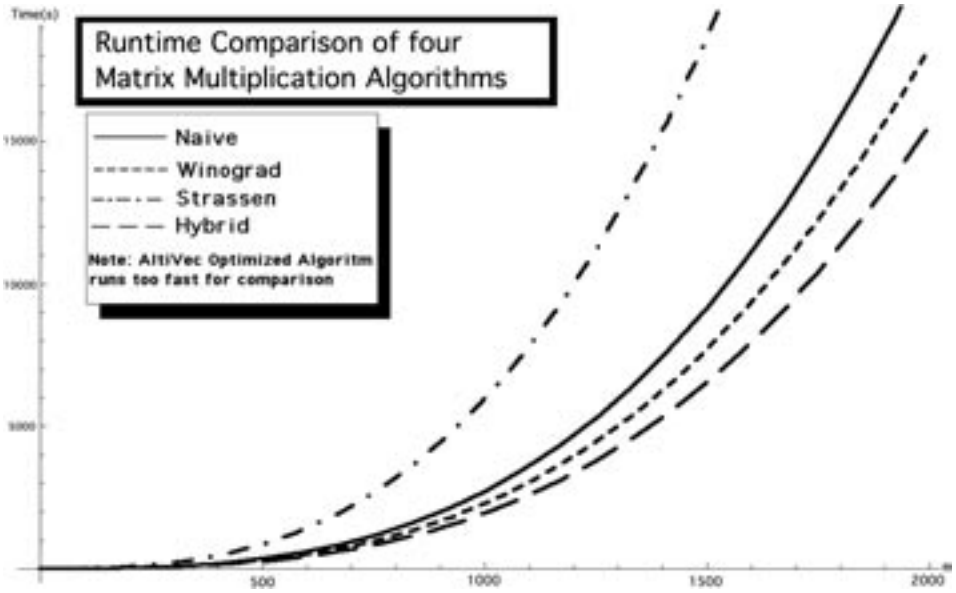
Since Strassen's algorithm is recursive, we can solve a set of recurrence relations for the number of scalar multiplications and additions. It is relatively straightforward to show that the number of multiplications for an *n*-by-*n* matrix is $n^{2.807}$, and the number of additions is $(4.5)\, n^2 \log_2 n$.

Unlike the Empirical Study (which is the next topic of this paper), 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. The two graphs on the previous page compare the three algorithms in terms of the number of additions and the number of multiplications as a function of the matrix parameter *n*.

Empirical Study

The naive and Winograd's algorithms were implemented very easily. The empirical tests agree 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. The graph below illustrates the comparative run-times of the $O(n^3)$ algorithms.

However, Strassen's algorithm presented problems because it is based on recursion. Initially, we manually copied the input matrices into the respective quadrants, a measure which was very time consuming. To circumvent this performance bottleneck, we needed to perform all operations on the input matrices themselves by keeping track of the partitioning. Despite optimizations of Strassen's algorithm, its running time decreased by only 25%, and 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. As we see in a later section, using optimized code with 128-bit headers significantly improves the actual running time of Strassen's algorithm.



For the purpose of these performance comparisons, we conducted tests primarily on a personal laptop computer—an Apple iBook G3 500MHz with 384MB of RAM. The figure below presents the running times (in seconds) of the algorithms as a function of the matrix parameter *n*. This figure clearly shows the polynomial behavior of the algorithms. Note also that these specific run-time approximations depend on the processor speed and so are specific only for the iBook computer. While they serve as a good graphical comparison, they also allowed us to predict the points (values of *n*) of intersection where Strassen's algorithm becomes faster than the $O(n^3)$ algorithms. Looking at the combined graph for all three algorithms, you can see that the intersection points will be for very large *n*. The value of *n* for which Strassen's algorithm (in this implementation) becomes better than the naive algorithm is $n = 60{,}750$. In fact, it would take the iBook computer 19.5 years to confirm this result. The intersection point for the Winograd algorithm is even more terrifying, as you can see from the graph, from which it becomes evident that Winograd's algorithm is best for reasonably-sized matrices (i.e. *n* between 2 and 60,000).

Hybrid Algorithm

Strassen's algorithm performs more slowly than expected in the empirical tests mainly due to the large number of recursive calls, which theoretically should not take up too much time, but in fact account for most of the running time as seen on the previous graph. In the search for a more efficient algorithm for matrix multiplication, we constructed a Hybrid algorithm, which uses the

Strassen's method until a predetermined cutoff size of the seven sub-matrices, after which Winograd's algorithm takes over. This kind of optimization is common to other computationally intensive tasks such as sorting. It has been found that the theoretically faster quick-sort algorithm performs better in real-world applications when the recursion is stopped and the slower, but simpler, insertion sort is used to complete the sorting. In a similar way, we reduce the number of recursive calls exponentially while still taking advantage of Strassen's reduced number of multiplications.

Empirical tests (the long-dashed line in the comparison figure above) show that the resultant Winograd-Strassen hybrid algorithm performs significantly better than both of its predecessors. While, theoretically, the Hybrid algorithm performs almost $O(n^3)$ multiplications (reduced by the use of Strassen's algorithm), with a smaller scaling constant (due to the use of Winograd's algorithm), it is empirically faster than the original Winograd's algorithm by about 15%. The critical level of recursion before switching to Winograd is related to the Hybrid algorithm's performance, and both depend primarily on the starting value for n, with a larger n yielding a more significant speedup.

AltiVec Optimization

In 1998, Apple Computer introduced a specialized vector computation unit called AltiVec, which handles multiple vector operations with increased performance, as in the SSE instruction set on Intel processors. In 2000, Apple's Advanced Computation Group [4] arrived at the revolutionary conclusion that AltiVec instructions drastically improve the running time for Strassen's algorithm. According to their results, a 500 MHz G4 processor could multiply two 2500×2500 matrices in about forty seconds.

Our own AltiVec experiment confirms the stunning performance of AltiVec code with an even more drastic result: a multiplication of two random 2500×2500 matrices finishes in less than a second on a 1.25 GHz G4 processor, presenting a sustained performance of 3.45 TFLOPS (Trillion Floating Point Operations Per Second). Note that the same operation takes 130 seconds to complete by a non-AltiVec optimized implementation.

The efficiency of Apple's AltiVec velocity engine results from the fact that it is able to process four multiplications simultaneously by multiplying four-dimensional vectors of numbers instead of individual numbers. This not only speeds up the pace of actual number crunching, but also allows the four sub-matrix references (pointers) to be contained in a single 128-bit header (because each is exactly 32 bits), which practically eliminates the slowdown observed earlier in the non-optimized Strassen's algorithm.

Conclusion

The empirical results of our study lead us to conclude that Strassen's algorithm is an efficient solution for highly-optimized processors such as the G4 with the AltiVec velocity engine by Apple, which most recently implemented fast matrix multiplication in a technical package called AG-BLAST, which significantly accelerates protein and DNA searches used in biomedical research and drug discovery. However, in everyday applications, such as embedded DSP processors in cell phones and digital cameras, where high-performance vector processors are not feasible, the computational bound for matrix multiplication could be effectively lowered by using our Hybrid algorithm.

References

1. Aho, Hopcroft, and Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley: New York, 1974.
2. Weiss, Mark. *Data Structures & Algorithm Analysis in C++*. Addison-Wesley: New York, 1998.
3. Manber, Udi. *Introduction to Algorithms: A Creative Approach.* pp 301- 304. Pearson Education: New Jersey, 1989.
4. Apple Computer, Inc. "Fast Matrix Algebra on Apple G4." http://developer.apple.com/hardware/ve/pdf/g4matrix.pdf.
5. Jacobson, et al. "Implementation of Strassen's Algorithm for Matrix Multiplication." http://www.supercomp.org/sc96/proceedings/SC96PROC/JACOBSON/INDEX.HTM.
6. McLoughlin, Aileen. "Using Computers to Search for Efficient Numerical Algorithms." http://www.compapp.dcu.ie/alumni/newsletter/scientist1.htm.

Boyko Kakaradov

Boyko Kakaradov is a freshman originally from Bulgaria, but who currently lives in Camden, Maine. While he is still undeclared, he is interested in artificial intelligence and nanobiology. He became involved in matrix multiplication through the extension of a high school algorithms class. He would like to dedicate this publication to his former Computer Science teacher, Duncan Innes, for his inspiration and academic support; Mr. Innes lived his lifelong dream of contributing to his students' academic and cultural enlightenment.