A Variation of Strassen's Matrix Multiplication Algorithms

Sarah M. Loos
Undergraduate, Computer Science, Indiana University, smloos@indiana.edu

ABSTRACT
A very simple recasting of this classic 7-multiplication recursion improves its time performance for rectangular matrices of order \( n \) when \( n \) is not a power of two. No time is lost in the rare cases when \( n = 2^k \). A similar relabeling applies, as well, to Winograd's 15-addition variant.

SUB-CUBIC MULTIPLICATION
Strassen's \((\mathcal{O}(n^\log_27))\) matrix-multiplication algorithm was published forty years ago, stunning algorithmists with a simple recurrence that improved the best asymptotic time for a very important and heretofore "obviously cubic" problem. Since then there have been debates about its practicality, based on numeric stability, extra space, and locality. Among those issues has been how to deal with the \( n \) between powers of two and with rectangular matrices.

We cleave a matrix of order \( n \) by partitioning the \( n \) indices into 2\( \left\lceil \log_2(1+ n) \right\rceil - 1 \) and \( n - 2\left\lceil \log_2(1+ n) \right\rceil - 1 \). In the simplest cases — square matrices of order \( n = n \) — it results in a northwest quadrant that is square with order a power of two, and southwest, northeast, and southeast matrices fitted around it. This is close to static padding with zeroes, except that bounds checking identifies entire blocks of zeroes.

Recursions on such a northwest quadrant, therefore, will surely generate a lot of work, whereas recursions on the southeast might generate very little. In the standard algorithm below, it is clear that the big northwest quadrants are used frequently relative to the others. With this cleaving, any sum with one of them as an addend, or any product of two northwest factors, is likely to require a full dose of scalar operations, and so they are identified below in red.

\[
\begin{align*}
M_1 &= (A_{uw} + A_{xw})(B_{uw} + B_{xz}), \\
M_2 &= (A_{uw} + A_{xw})B_{uw}, \\
M_3 &= A_{uw}(B_{ux} - B_{xz}), \\
M_4 &= A_{xw}(B_{ux} - B_{uw}), \\
M_5 &= (A_{uw} + A_{xw})B_{xw}, \\
M_6 &= (A_{uw} - A_{xw})(B_{uw} + B_{xz}), \\
M_7 &= (A_{uw} - A_{xw})B_{xw}, \\
C_{uw} &= M_1 + M_4 - M_5 + M_7, \\
C_{ux} &= M_1 + M_4, \\
C_{ux} &= M_1 + M_4, \\
C_{xw} &= M_1 - M_2 + M_3 + M_6.
\end{align*}
\]

THE SIMPLE VARIANT
A simple relabeling of the quadrants generates an equivalent algorithm without a northwestern tilt.

Let \( Z \) be the zero matrix; it is the additive identity and the multiplicative annihilator: \( A + Z = A \); \( AZ = ZA \), assuming compatible factors. These algorithms treat any quadrant that is out-of-bounds as \( Z \), and so this algebra prunes away many scalar operations.

Let \( I \) be the identity matrix, and consider the simple permutation \( P = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} \) that flips columns or rows, depending on whether it is multiplied on the right or left of a matrix. Then

\[
AB = (A_{uw} A_{xw})(B_{uw} B_{xz}) = APP^T B = (A_{uu} A_{ux})(B_{uw} B_{xz}).
\]

When the quadrants in Strassen's Algorithm are renamed according to the right most of \( P \), one gets those same algorithms. Although they take different intermediate values, the sums for the final quadrants of \( C \) remain the same as above.

\[
\begin{align*}
M_1 &= (A_{uu} + A_{ux})(B_{uw} + B_{xz}), \\
M_2 &= (A_{uu} + A_{ux})B_{uw}, \\
M_3 &= A_{uu}(B_{ux} - B_{xz}), \\
M_4 &= A_{ux}(B_{ux} - B_{uw}), \\
M_5 &= (A_{uu} + A_{ux})B_{xw}, \\
M_6 &= (A_{uu} - A_{ux})(B_{uw} + B_{xz}), \\
M_7 &= (A_{uu} - A_{ux})B_{xw}, \\
C_{uu} &= M_1 + M_4 - M_5 + M_7, \\
C_{ux} &= M_1 + M_4, \\
C_{ux} &= M_1 + M_4, \\
C_{xw} &= M_1 - M_2 + M_3 + M_6.
\end{align*}
\]

Because we expect only the northwest quadrants to be full, we relabel the quadrants of Strassen's algorithm, as well as Winograd's variant, to favor work on the others. The other three quadrants, to the south and east, can be far smaller — even empty — and so we use them more. As soon as they go out-of-bounds in recurrence, they are treated as zeroes and many operations are saved.

In spite of our simple and effective improvements to the \( \mathcal{O}(n^2) \) algorithm, the race is still won by the classic \( \mathcal{O}(n^3) \) algorithm, which makes better use of memory bandwidth.

Our modified algorithms perform no worse than the classic ones in all cases and their performance is much improved on rectangular matrices as size grows beyond any power of two.

CONCLUSION

In order to demonstrate our alternative version, we implemented the 15-addition Strassen-Winograd version using C++ and MTL4, which provides recursors as a generic-programming alternative to iterators. Tests were run on a uniprocessor AMD Opteron, clocked at 2.0 GHz, using Gnu g++ -O3, and Release 6595 of MTL4.

Similarly we implemented the original Strassen-Winograd algorithm and the classic \( \mathcal{O}(n^3) \) algorithm (using MTL4 recursors). All three algorithms run to \( 32 \times 32 \times 32 \) base cases that are implemented in tuned assembly-language. As a result, all tests are on sizes that are even multiples of 32. We also compared to the implementation of Strassen-Winograd by Douglas et al. available from NetLib.

The following plots present the results for rectangular matrix multiplication of size \( n \times 2n \times n \) where \( n \) is plotted on the horizontal axis. All plots are normalized by dividing by \( 5.2 \times n^3 \). All sizes between \( 96 \times 192 \) up to \( 4096 \times 8192 \) in steps of \( 32 \) or \( 64 \) were tested. Notably only our algorithm and the classic \( \mathcal{O}(n^3) \) algorithm exhibit smooth performance. The latter plot rises because its \( 2^n \) time is normalized only by \( n^{15} \), but it still remains fastest.

PRINCIPAL REFERENCES


Supported, in part, by NSF under an REU and grants numbered CCF-0750960, CCF-0541364, and EIA-0305568.

October 5, 2008