Strassen's Matrix Multiplication Relabeled

Sarah M. Loos
Computer Science Department
Indiana University

David S. Wise
Computer Science Department
Indiana University

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 $n = 2^p$. A similar relabeling applies, as well, to Winograd's 15-addition variant, for which experiments on $n \times 2n \times n$ products show an average 25% time improvement.

1. SUB-CUBIC MATRIX MULTIPLICATION

Strassen's $O(n^{2.807})$ 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 [Strassen 1969]. 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 plethora of n between powers of two and with rectangular matrices.

We cleave a matrix of order n by partitioning the n indices into $2^{\lceil \lg n \rceil - 1}$ and $n - 2^{\lceil \lg n \rceil - 1}$. For example, order 800 would be partitioned into 512 and 288. In the simplest cases—square matrices of order n—it results in a northwest quadrant that is square with its order a power of two, and southwest, northeast, and southeast matrices fitted around it. (Intercardinal directions are used to identify the four quadrants of a matrix.) The middle two are likely rectangular and the last can be quite small. This cleaving contrasts with the balanced partitions commonly used in the literature: static padding, dynamic padding, and dynamic peeling, all of which cut n roughly in half [Huss-Lederman et al. 1996]. It is closest to static padding with zeroes, except that bounds checking identifies large blocks of zeroes.

Version 4 of the Matrix Template Library makes it very easy to declare patterns of tiled matrices and to recur on quadrants as described above [Gottschling et al. 2007]. First, it allows tiling to be declared by specifying the masks for the Cartesian indices into a matrix. So, matrices can be specified in row- or column-major, in Morton order, or in hybrids thereof. Second, it provides *recursators*, the analog of iterators from generic programming, to decompose matrices into its quadrants.

Recursions on such a northwest quadrant, therefore, will surely generate a lot of work, whereas recursions on the southeast might generate very, very little. Taking a look at the standard algorithms, just below, one notices that those big northwest quadrants are used quite a lot, compared to the others. With this cleaving, any sum with one of them as an addend, or any product of two big factors is likely to require a full dose of scalar operations, and so they are identified below in red.

Strassen's $O(n^{2.807})$ Algorithm:

$$\begin{aligned}
\mathbf{M_1} &:= (\mathbf{A_{nw}} + A_{se})(\mathbf{B_{nw}} + B_{se}), \\
M_2 &:= (A_{sw} + A_{se})\mathbf{B_{nw}}, \\
M_3 &:= \mathbf{A_{nw}}(B_{ne} - B_{se}), \\
M_4 &:= A_{se}(B_{sw} - \mathbf{B_{nw}}), \\
M_5 &:= (\mathbf{A_{nw}} + A_{ne})B_{se}, \\
\mathbf{M_6} &:= (A_{sw} - \mathbf{A_{nw}})(\mathbf{B_{nw}} + B_{ne}), \\
M_7 &:= (A_{ne} - A_{se})(B_{sw} + B_{se});
\end{aligned}$$

$$\begin{aligned}
\mathbf{C_{nw}} &:= \mathbf{M_1} + M_4 - M_5 + M_7, \\
C_{ne} &:= M_3 + M_5, \\
C_{sw} &:= M_2 + M_4, \\
C_{se} &:= \mathbf{M_1} - M_2 + M_3 + \mathbf{M_6}.
\end{aligned}$$

Eight-Recursion $O(n^3)$ Algorithm:

$$\begin{aligned}
C_{nw} &= \mathbf{A_{nw}}\mathbf{B_{nw}} + A_{ne}B_{sw}, \\
C_{ne} &= \mathbf{A_{nw}}B_{ne} + A_{ne}B_{se}, \\
C_{sw} &= A_{sw}\mathbf{B_{nw}} + A_{se}B_{sw}, \\
C_{se} &= A_{sw}B_{ne} + A_{se}B_{se}.
\end{aligned}$$

Strassen's full, northwest quadrants are named 8 times, and his other three are called out 16 times. Six of the seven initial products M_i have eight factors that are full. The classic algorithm names each of the intercardinal directions 4 times: 4 calls of the northwest and 12 of the others.

Winograd's 15-addition improvement of Strassen's algorithm suffers the same tilt to the northwest [Winograd 1971; Douglas et al. 1994].

2. 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: $A+Z = A$; $AZ=Z=ZA$, 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.

Consider the simple permutation $P = \begin{pmatrix} Z & I \\ I & Z \end{pmatrix}$, that flips columns or rows, depending on whether it is multiplied on the right or left of a matrix. Then

$$AB = \begin{pmatrix} A_{nw} & A_{ne} \\ A_{sw} & A_{se} \end{pmatrix}\begin{pmatrix} B_{nw} & B_{ne} \\ B_{sw} & B_{se} \end{pmatrix} = APP^T B = \begin{pmatrix} A_{ne} & A_{nw} \\ A_{se} & A_{sw} \end{pmatrix}\begin{pmatrix} B_{sw} & B_{se} \\ B_{nw} & B_{ne} \end{pmatrix}.$$

When the quadrants in Strassen's Algorithm are renamed according to the rightmost product, almost the same algorithm results. Although the M_i take different intermediate values, the sums for the final quadrants of C remain the same as above.

Variation of Strassen's $O(n^{2.807})$ Algorithm:

$$\begin{aligned}
M_1 &:= (A_{ne} + A_{sw})(B_{sw} + B_{ne}), & \mathbf{C_{nw}} &:= M_1 + M_4 - M_5 + \mathbf{M_7},\\
M_2 &:= (A_{se} + A_{sw})B_{sw}, & C_{ne} &:= M_3 + M_5,\\
M_3 &:= A_{ne}(B_{se} - B_{ne}), & C_{sw} &:= M_2 + M_4,\\
M_4 &:= A_{sw}(\mathbf{B_{nw}} - B_{sw}), & C_{se} &:= M_1 - M_2 + M_3 + M_6.\\
M_5 &:= (A_{ne} + \mathbf{A_{nw}})B_{ne},\\
M_6 &:= (A_{se} - A_{ne})(B_{sw} + B_{se}),\\
\mathbf{M_7} &:= (\mathbf{A_{nw}} - A_{sw})(\mathbf{B_{nw}} + B_{ne});
\end{aligned}$$

Fewer red quadrant identifiers tell the story. The full, northwest quadrants are now named only 4 times, and the others are used 20 times. Now only three of the seven initial products are full. Those 20 blocks could be Z, avoiding much computation. So there is more chance for addends to be Z, each simplifying a sum, and for factors to be Z, so annihilating an entire multiplication. Winograd's version of Strassen's multiplication can be relabeled similarly, also removing its northwestern tilt.

3. EXPERIMENTAL RESULTS

In order to demonstrate our alternative version, we implemented both 15-addition Strassen-Winograd versions using C++ and MTL4, which provides *recursators* as a generic-programming alternative to iterators. It also provides a very convenient declaration for the pattern of allocation within dense matrices, *e.g.* in row-major or Morton order [Adams and Wise 2006a]. Our MTL4 implementations of both Strassen-Winograd algorithms, as well as the 8-recursion classic algorithm, use a hybrid of Morton order whose 32 × 32 tiles are laid out in Morton order but internally each tile is nearly row-major. "Nearly" because each is actually shark-toothed to suit the available 32 × 32 × 32 base case [Gottschling et al. 2007]. Double-precision floating-point products were run on a uniprocessor AMD Opteron, clocked at 2.0 Ghz, using Gnu `g++ -O3`, and Revision 6595 of MTL4. Each test was timed thrice, with the middle value plotted.

All three algorithms run to $128 \times 128 \times 128$ base cases that use the classic algorithm for two recursions and then the $32 \times 32 \times 32$ base case, locally implemented in hand-coded assembly-language. It uses the sharktooth layout to get particularly good performance from the Opteron's sixteen superscalar, SSE2 registers. As a result all tests are run on sizes that are even multiples of 32. We also compare the implementation of Strassen-Winograd by Douglas *et al.* available from NetLib [Douglas et al. 1994]. (They also discuss how they size their base cases.)

The figures present the resources required for rectangular matrix multiplication of size $n \times 2n \times n$ where n is plotted on the horizontal axis. All plots are normalized to be expressed in units of "resource/SWFLOP" for our Strassen-Winograd algorithms. If we were normalizing for the classic algorithm, we would have divided the resource measures—of time and the respective cache-miss counts—by $4n^3 - 2n^2$ which is its FLOP count, equivalent to two square $n \times n \times n$ products of $2n^3 - n^2$ FLOPs each. The quotient would plot the observed leading coefficient of the cubic formula for each resource [Adams and Wise 2006b]. When it is a horizontal line, it identifies the leading coefficient hidden by big-Oh notation for this $O(n^3)$ algorithm.

In this way we obtain plots of each of the resources that reveal details at small n and that can be aligned directly against plots of other resources. Two comments remain. First, rather than expressing time in units of seconds, we multiply by the 2Ghz clock speed to express time in units of processor cycles; this abstracts away the clock of a faster, identical processor. Second, we normalize it with the analytically computed SWFLOPs of our Strassen-Winograd algorithm. Its $128 \times 128 \times 128$ base case is done with the classic algorithm that uses $2 \cdot 128^3 - 128^2 = 2 \cdot 8^7 - 4^7$ FLOPs. For $n = 2^p$, $p \geq 7$, the number of FLOPs in the square $2n \times 2n \times 2n$ problem is

$$f(2n) = 15 \cdot n^2 + 7f(n) = 15 \cdot 4^p \sum_{i=0}^{p-7} \left(\frac{4}{7}\right)^i + 7^{p-6}(2 \cdot 8^7 - 4^7) = 5.1726(2n)^{lg7} - 5(2n)^2.$$

For the rectangular problem tested, both Strassen and Strassen-Winograd perform *three* top-level recursive multiplications and two additions, and so our $n \times 2n \times n$ problem ideally takes $15.518n^{lg7} + O(n^2)$ SWFLOPs. *All* resources are normalized by dividing by that leading term. Dividing all results by the same factor preserves the relative comparisons among all our results, as if there were no normalization.

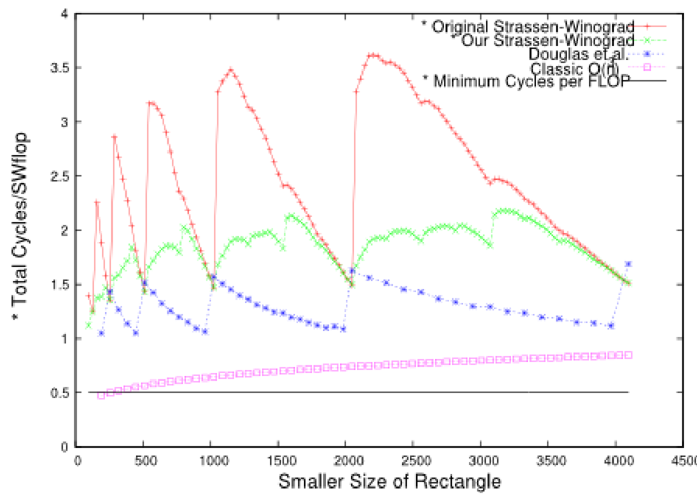

Fig. 1. Time expressed as cycles/SWFLOP.

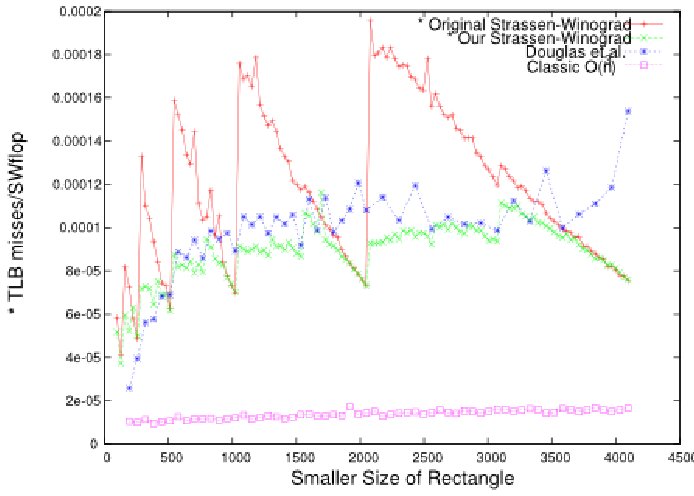

Fig. 2. TLB misses/SWFLOP.

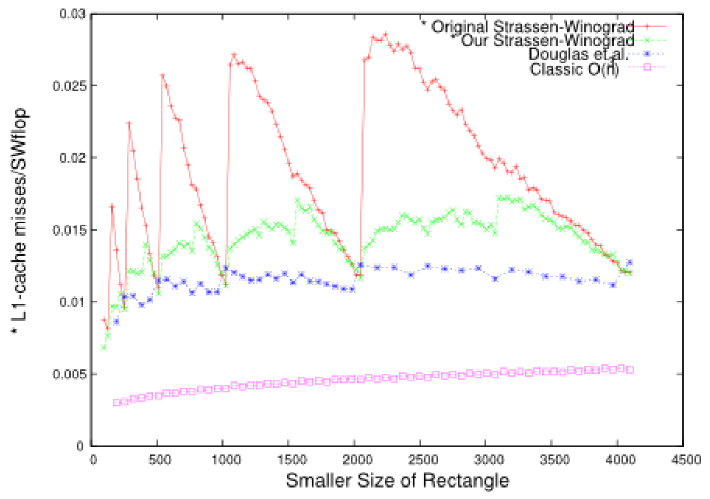

Fig. 3. L1 misses/SWFLOP.

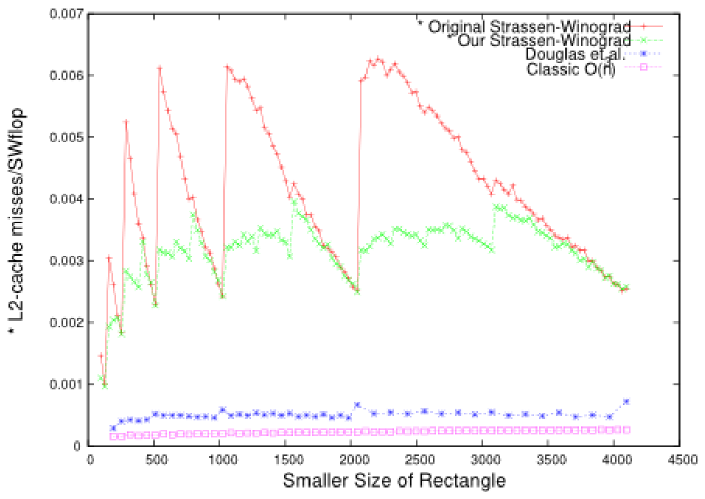

Fig. 4. L2 misses/SWFLOP.

The critical plots, in Figure 1, show how our simple relabeling of the Strassen-Winograd algorithm nicely smoothes the sharp spikes in the cycles-per-SWFLOP of the classic Strassen-Winograd multiplication. Integration of both plots yield average cycles/SWFLOP of 2.465 and 1.849, an improvement of 25%. Both these plots can be compared to the superscalar Opteron's optimum ½ cycle/FLOP plotted as a constant lower bound and not approached by any Strassen algorithm. That separation is likely due to their lack of locality, suggested in Figures 2-4. (The scale on the Y-axis is valid only for these three plots, as indicated by asterisks on each legend.)

Two more plots compare directly to the Strassen-Winograd tests, even though they do not fit the vertical scale. They are faster (lower) and also divided by $15.518n^{lg7}$, which preserves comparisons but is an improper divisor. For Douglas's *et al.* implementation, it is off only by an undetermined constant, so that plot should still be flat. The plot of the classic $O(n^3)$ algorithm in MTL4 shows it to be far faster, even though both the coefficient and the power of n in the divisor are incorrect, and so this plot smoothly rises.

The performance of Douglas *et al.* is faster than our MTL4 implementations, but it exhibits similar scalloped performance to the original MTL4 Strassen-Winograd algorithm. Its performance might also improve with the same relabeling.

Figures 2, 3, and 4 show normalized plots for translation lookaside buffer (TLB) misses, L1-cache misses, and L2-cache misses per FLOP. Because all plots are similarly normalized, each can be set against Figure 1's for a comparison of which caches most constrain the aggregate times. Figure 2 shows that the classic algorithm has dramatically fewer expensive TLB misses, perhaps accounting for its good time in spite of performing so many more FLOPs. With no need for temporary space it is much more local in memory. In that figure, our new algorithm's TLB miss rate is better than Douglas's *et al.* which does poorly here.

Figure 3 plots L1-cache misses that seem similar to Figure 1. So the L1 cache performance may equivalently constrain all the algorithms. It is remarkable, however, that the plots of the original Strassen-Winograd algorithm are so similar in all four figures—as if it consumes all resources apace. In contrast, Figure 4 is much different, showing that Douglas's *et al.* has the L2 misses of the highly local classic algorithm, suggesting that L2 misses do not constrain their performance.

4. CONCLUSION

Because we expect only the northwest quadrants to be full, we relabel the quadrants of Strassen's algorithm, as well as Winograd's variant of it, 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 the recurrence, they are treated as zeroes and many operations are saved.

In spite of our simple and effective improvements to the $O(n^{lg 7})$ algorithms, the race is still won by the classic $O(n^3)$ algorithm, which makes better use of memory bandwidth. Figures 2-4 show it to be highly local.

We did an extensive—yet incomplete—literature search for this relabeling. Because Strassen's tantalizing algorithm quickly found its way into textbooks, our search included both books and journals [Aho et al. 1974; Cohen and Roth 1976; Fischer and Probert 1979; Purdom and Brown 1984, for example]. We found no instance of the simple relabeling, nor of its dramatic smoothing of the performance of the Strassen and Strassen-Winograd multiplication algorithms.

Our modified algorithms perform no worse than the classic ones. Experiments with Strassen-Winograd demonstrate clear improvement where one expects, away from powers of two. The average improvement of 25% establishes the claim that this simple relabeling of Strassen's multiplication algorithms improves performance. So, after forty years, Strassen's multiplication should be relabeled as shown here.

5. ACKNOWLEDGEMENTS

This material is based on research supported, in part, by NSF under an REU and grants numbered CCF-0736560, CCF-0541364 and EIA-0202048.

6. REFERENCES

Adams, M. D. and Wise, D. S. 2006a. Fast additions on masked integers. *SIGPLAN Not. 41*, 5 (May), 39-45.

Adams, M. D. and Wise, D. S. 2006b. Seven at one stroke: Results from a cache-oblivious paradigm for scalable matrix algorithms. In *MSPC '06: Proc. 2006 Wkshp. Memory System Performance and Correctness*. ACM Press, New York, 41-50.

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA.

Cohen, J. and Roth, M. 1976. On the implementation of Strassen's fast multiplication algorithm. *Acta Inform*. 6, 4 (Aug.), 341-355.

Douglas, C. C., Heroux, M., Slishman, G., and Smith, R. M. 1994. GEMMW: a portable level-3 BLAS Winograd variant of Strassen's matrix- multiply algorithm. *J. Comput. Phys. 110*,1,1-10.

Fischer, P. C. and Probert, R. L. 1979. Storage reorganization techniques for matrix computation in a paging environment. *Commun. ACM 22*, 7 (July), 405-415.

Gottschling, P., Wise, D. S., and Adams, M. D. 2007. Representation-transparent matrix algorithms with scalable performance. In *Proc. 21st ACM Int. Conf. on Supercomputing*. ACM Press, New York, 116-125.

Huss-Lederman, S., Jacobson, E. M., Tsao, A., Turnbull, T., and Johnson, J. R. 1996. Implementation of Strassen's algorithm for matrix multiplication. In *Supercomputing '96: Proc. 1996 ACM/IEEE Conf. Supercomputing (CDROM)*. IEEE Computer Soc. Press, Washington, DC, 32.

Purdom, P. W. and Brown, C. A. 1984. *The Analysis of Algorithms*. Holt, Rinehart and Winston, New York.

Strassen, V. 1969. Gaussian elimination is not optimal. *Numer. Math. 13*, 4 (Aug.), 354-356.

Winograd, S. 1971. On the multiplication of 2 × 2 matrices. *Linear Algebra Appl. 4*, 4 (Oct.), 381-388.