2002

TR-2002012: The Aggregation and Cancellation Techniques As a Practical Tool for Faster Matrix Multiplication

Igor Kaporin

THE AGGREGATION AND CANCELLATION TECHNIQUES AS A PRACTICAL TOOL FOR FASTER MATRIX MULTIPLICATION

Igor Kaporin

Exchange Visitor Program P-1-3705 in Computer Science
The Graduate Center - CUNY, New York, NY 10016

(on leave from Center of Supercomputer and Massively Parallel Applications
at Computational Center of the Russian Academy of Sciences, Moscow)

Correspondence to: I. Kaporin, Computing Center of the
Russian Academy of Sciences, Vavilova 40, Moscow 117967, RUSSIA
e-mail: kaporin@sci.kun.nl, kaporin@ccas.ru

Abstract

The main purpose of this paper is to present a fast matrix multiplication algorithm taken from [10] in a refined compact "analytical" form and to demonstrate that it can be implemented as quite efficient computer code. Our improved presentation enables us to simplify substantially the analysis of the computational complexity and numerical stability of the algorithm as well as its computer implementation. The algorithm multiplies two $N \times N$ matrices using $O(N^{2.7760})$ arithmetic operations. In the case where $N = 18 \cdot 48^k$, for a positive integer k, the total number of flops required by the algorithm is $4.893N^{2.7760} - 16.165N^2$, which is quite competitive with a similar estimate for the Winograd algorithm, $3.732N^{2.8074} - 5N^2$ flops, $N = 8 \cdot 2^k$, the latter being current record bound among all known practical algorithms. Moreover, we present a pseudo code of the algorithm which demonstrates its very moderate working memory requirements, much smaller than that of the best available implementations of Strassen and Winograd algorithms. We also reexamine an algorithm from [11] with operation count $3.682N^{2.8109} - 7.303N^2$, $N = 8 \cdot 12^k$, which performs well even for medium matrix sizes, e.g., $N < 2000$. For matrices of medium-large size (say, $2000 \leq N < 10000$) we consider one-level algorithms and compare them with the (multilevel) Strassen and Winograd algorithms. The results of numerical tests clearly indicate that our accelerated matrix multiplication routines implementing two or three disjoint product-based algorithm are comparable in computational time with an implementation of Winograd algorithm and clearly outperform it with respect to working space and (especially) numerical stability. The tests were performed for the matrices of the order of up to 7000, both in double and single precision.

KEYWORDS: fast matrix multiplication, Strassen algorithm, Winograd algorithm, Pan's aggergation/cancellation method, numerical stability

1 Introduction

Matrix multiplication is one of the most basic computational tasks arising in numerical computing. Software implementing this operation (among other basic linear algebra modules) is always included into general-purpose scientific packages, or invoked by them, see, e.g., [19], [20]. The most widely known is the LAPACK library, which includes, e.g., such routines as DGEMM and SGEMM (multiplication of general rectangular matrices in double and single precision, respectively).

Matrix multiplication is also a basic operation for many important nonnumerical computational problems such as:

(a) Transitive closure and all-pair-shortest-distance problems in graphs [1],[2];

(b) Parsing algorithms for context-free grammars (as is known, context-free language recognition over an input sequence of length n can be reduced to multiplication of $n \times n$ matrices) [7], [6];

(c) Pattern recognition tasks (classification and finding similar objects), arising, e.g., in factor analysis of texts or in image retrieval, see [5] and references therein;

(d) Computational molecular biology (processing gene expression profiles, which is reduced to the problem of identification of Boolean networks) [3], [4].

In some of the above problems, the matrices are Boolean rather than filled with floating-point numbers; however, most of the results on fast matrix multiplication still hold true. Moreover, the numerical stability problem disappears in Boolean settings.

As a part of intensive development of software for fundamental computational kernels during the last three decades, a considerable effort was directed towards efficient implementation of fast Matrix Multiplication (MM) algorithms [27], [18], [22], [23], [24]. However, only Strassen algorithm (1969) [13] and rather similar Winograd algorithm (1974), see, e.g.,[12], [21], have been implemented. The latter is often referred to as Strassen-Winograd's, and hereafter we use the abbreviation SW. The main deficiencies of the SW based implementations are:

(i) considerably weaker numerical stability than that of the classical $O(n^3)$ procedure (thus, the Strassen-type algorithms are rather useless in single precision floating-point computations, cf. [18]);

(ii) the need for a rather large volume of work memory;

(iii) essential inconsistency between the algorithmic tunings providing the minimization of the total operation count and the tunings aimed at the maximization of Mflops performance on modern RISC computers;

(iv) considerable loss of efficiency for inputs being rectangular matrices of arbitrary sizes.

There are also some complications with efficient parallel implementation, but these issues are not treated here.

However, *there exist better matrix multiplication algorithms* which outperform the SW ones in every above mentioned respect. The basis for the construction of such algorithms was set in [14], [15], [10], where the so-called aggregation-cancellation techniques were proposed for calculating two or three disjoint matrix products. Later on, in [11], [17] a great practical potential hidden in such designs was revealed, in particular the gain in floating-point accuracy, but also their rather regular structure and very moderate working memory requirements, typically smaller than that of the available SW implementations.

Our refined algorithm multiplies two $N \times N$ matrices by using $O(N^{2.7760})$ flops (floating point arithmetic operations). In the case where $N = 18 \cdot 48^k$, for a positive integer k, the total number of flops required by the algorithm is $4.894N^{2.7760} - 16.165N^2$ which is quite competitive with the estimate $T_{SW} = 3.732N^{2.8074} - 5N^2$ flops, $N = 8 \cdot 2^k$, for the SW algorithm. The latter was a current record bound among all known *practical* algorithms. (We do not count the theoretically fast algorithms [8], [9] that support even much smaller exponents (2.375... for square matrix multiplications) but are not competitive even with classical algorithm unless N is immensely large.)

Our numerical tests indicate that the fast matrix multiplication routine implementing our algorithm based on two and three disjoint products is comparable to an implementation of the SW algorithm with respect to time, but takes considerably less working storage and possesses much better numerical stability (almost as good as for some implementations of the standard MM algorithm). The tests were performed for the matrices of the order of up to 7000, both in double and single precision.

The paper is organized as follows. In Section 2, we restate and refine some results from [11]; one of the main results is the $n \times 2n$ by $2n \times n$ MM algorithm requiring $n^3 + 3n^2 - n$ bilinear multiplications. This also serves as an elementary introduction into our subject. In Section 3 we present a refined compact version of the fast Disjoint Triple MM algorithm taken from [10] as well as the related $n \times 3n$ by $3n \times n$ matrix multiplication algorithm using $n^3 + 12n^2 + 24n$ bilinear multiplications derived similarly to [11]. Throughout the paper, we give pseudo-codes for the key algorithms. The analysis of the computational complexity and discussion on numerical stability and computer implementation of the algorithm are the subjects of Section 4. There we outline one-level procedures derived from the above rectangular MM algorithms, in particular, their adjustment to odd-sized and rectangular inputs. In section 5, the results of numerical tests are given.

2 Two disjoint product based algorithms

Let us devise fast MM algorithms [16], [11] by relying on *aggregation* technique, specifically, on the so-called 2-procedure; hereafter we refer to them as PK2-algorithms.

2.1 A recursive procedure for two disjoint MM

To compute two generally disjoint matrix products

$$Z = XY, \qquad W = UV,$$

where all U, V, W, X, Y, Z are $n \times n$ block matrices with the blocks properly dimensioned, consider n^3 aggregates

$$m_{ijk} = (x_{ik} + u_{kj})(y_{kj} + v_{ji}).$$

Summation over k or over j gives us z_{ij} or w_{ki}, respectively, up to some additive correction terms which involve only $3n^2$ multiplications:

$$z_{ij} = -c_j - (x_i + u_j)v_{ji} + \sum_{k=1}^{n} m_{ijk}, \qquad w_{ki} = -r_k - x_{ik}(y_k + v_i) + \sum_{j=1}^{n} m_{ijk},$$

where

$$c_j = \sum_{k=1}^{n} u_{kj}y_{kj}, \quad r_k = \sum_{j=1}^{n} u_{kj}y_{kj}; \qquad x_i = \sum_{k=1}^{n} x_{ik}, \quad u_j = \sum_{k=1}^{n} u_{kj}, \quad y_k = \sum_{j=1}^{n} y_{kj}, \quad v_i = \sum_{j=1}^{n} u_{ji}.$$

Hence, the number of multiplications is only

$$\mu(n) = n^3 + 3n^2,$$

(compared to $2n^3$ for the double application of the standard algorithm).

The number of additions and subtractions must be accounted separately for each typical size of matrix blocks involved. In what follows, the three matrix pairs X, U, Y, V, and Z, W are composed

of $l \times l/t$, $l/t \times l$, and $l \times l$ blocks, respectively, where $t = 2$ for 2-procedure (section 2) and $t = 3$ for 3-procedure (section 3). One can see that the number of additions and subtractions is

$$\alpha_1(n) = 2n^3 + 6n^2 - 4n$$

for the input-type blocks (i.e., related to the input matrices X, Y, U, V), and

$$\alpha_2(n) = 2n^3 + 4n^2 - 2n$$

for the output-type blocks (i.e., related to the output matrices Z, W).

Since the number $n^3 + 3n^2$ is always even, a recursive algorithm groups smaller MM problems into pairs, and for each pair, the same procedure applies. For $N = n^k l$ with some l fixed, one has

$$b(N) = \frac{\mu(n)}{2} b(N/n) = \ldots = \left(\frac{n^3 + 3n^2}{2} \right)^k b(l),$$

where $b(N)$ is the number of multiplications for two $N \times N$ disjoint matrix products. Thus, the total number of operations can be estimated as $O(N^{\omega(n)})$, where

$$\omega(n) = \log \left(\frac{n^3 + 3n^2}{2} \right) / \log n,$$

in particular, $\omega(13) = 2 + \log_{13} 8 < 2.81071$. This exponent ω slightly exceeds $\omega = \log_2 7 < 2.80736$ in the Strassen-type algorithms, but the fast MM algorithm above is much more appealing from the practical viewpoint, especially for floating-point calculations, cf. [11].

2.2 The algorithm for $n \times 2n$ by $2n \times n$ product

For computation of a single matrix product, one may save more operations.

Consider the product H of $n \times 2n$ block matrix F by $2n \times n$ block matrix G:

$$H = FG,$$

The standard algorithm "by definition" $h_{ij} = \sum_k f_{ik} g_{kj}$ uses $2n^3$ block multiplications and $2n^3 - n^2$ (output-type) block additions.

The original problem is reduced to two disjoint products by the column splitting of F and row splitting of G into two equal blocks each, that is,

$$F = \left[\begin{array}{cc} X & U \end{array} \right], \qquad G = \left[\begin{array}{c} Y \\ V \end{array} \right],$$

where X, U and Y, V have the block sizes $n \times n$. Equations

$$Z = XY, \qquad W = UV; \qquad H = Z + W,$$

reduce the problem to a pair of disjoint matrix multiplications and an $n \times n$ matrix addition. Analysis of the expression for $z_{ii} + w_{ii}$ shows, however, that we may remove the aggregates m_{iii} from the summation by spreading their terms among the diagonal corrections for h_{ii}.

Skipping some rather easy linear algebra, let us present a pseudo-code for this algorithm (for a simpler version with a larger number of multiplications, see [11]):

1. $F1_i = 0, F2_i = 0,\ \ G1_i = 0, G2_i = 0,\ \ H1_i = 0, H2_i = 0,\ \ k = 1, \ldots n;$
2. **do** $i = 1, n$
   do $j = 1, n$
   $P := F_{i,n+j} \cdot G_{k,j}$
   if $(i = j)$ **then**
   $H_{i,i} = P$
   else
   $H1_j := H1_j - P$
   $H2_i := H2_i - P$
   $F1_i := F1_i - F_{i,j}$
   $F2_j := F2_j - F_{i,n+j}$
   $G1_i := G1_i - G_{i,j}$
   $G2_j := G2_j - G_{n+i,j}$
   end if
   end do
   end do
3. **do** $i = 1, n$
   $F0_i := F1_i + F2_i + F_{i,n+i}$
   $F1_i := F1_i - F_{i,i}$
   $F2_j := F2_j - F_{i,n+i}$
   $G0_i := G1_i + G2_i + G_{i,i}$
   $G1_i := G1_i - G_{i,i}$
   $G2_j := G2_j - F_{n+i,i}$
   $P = H_{i,i}$
   $H_{i,i} = H1_i + H2_i$
   $H1_j := H1_j - P$
   $H2_i := H2_i - P$
   end do
4. **do** $i = 1, n$
   do $j = 1, n$
   $P := F_{i,n+j} \cdot G_{k,j}$
   if $(i = j)$ **then**
   $H_{i,i} := H_{i,i} + F0_i \cdot G_{n+i,i} + F_{i,i} \cdot G0_i$
   else
   $S1 := F1_i + F2_j$
   $S2 := G1_i + G2_j$
   $H_{i,i} := H1_j + H2_i + S1 \cdot G_{n+j,i} + F_{j,i} \cdot S2$
   end if
   end do
   end do

5. **do** $i = 1, n$
  do $j = 1, n$
    do $k = 1, n$
      if $(i \neq j \neq k)$ **then**
      $S1 := F_{i,k} + F_{k,n+j}$
      $S2 := G_{k,j} + G_{n+j,i}$
      $P := S1 \cdot S2$
      $H_{i,j} := H_{i,j} + P$
      $H_{k,i} := H_{k,i} + P$
      end if
    end do
  end do
end do

Here $P, S1, S2$ are temporary variables and the symbol ":=" denotes in-place updating. The symbols $F0_i, H_{i,j}, \ldots$ indicate some storage areas rather than algebraic terms. The working memory is exactly defined by the matrix blocks $F0_i, F1_i, F2_i, F0_i, F1_i, F2_i, H1_i, H2_i,\ \ i = 1, \ldots n$. For n of the order of tens, this typically comprise only small fraction of the total volume of input and output data.

The operations count for the above algorithm is as follows. The number of multiplications is

$$\tilde{\mu}(n) = n^3 + 3n^2 - n,$$

and the number of block additions and subtractions is

$$\tilde{\alpha}_1(n) = 2n^3 + 6n^2 - 4n$$

for the input-type blocks and

$$\tilde{\alpha}_2(n) = 2n^3 + 5n^2 - 4n$$

for the output-type blocks.

2.3 The recursive algorithm for square matrices

Multiplying a pair of $N \times N$ matrices F and G with numerical entries, assume, for simplicity, that $N = n^k l$, where n and l are even, and $k \geq 1$, so $M = N/n$ is also even. Represent F as an $n \times 2n$ block matrix with $N/n \times N/(2n)$ blocks, G as an $2n \times n$ block matrix with $N/(2n) \times N/n$ blocks, and H as an $n \times n$ block matrix with $N/n \times N/n$ blocks.

Then the algorithm of the preceding section can be readily applied using

$$T_{PK2}(N) = \tilde{\alpha}_1(n)\frac{N^2}{2n^2} + \tilde{\alpha}_2(n)\frac{N^2}{n^2} + \frac{\tilde{\mu}(n)}{2}T_2(N/n),$$

arithmetic operations, where $T_2(M)$ operations are required for the computation of a pair of $M \times M/2$ by $M/2 \times M$ matrix products. The latter problem can be solved either by a standard algorithm ($T_2(M) = 2M^3 - 2M^2$), which gives rise to the so-called one-level algorithm [11], or by the application of the (generally, recursive) algorithm of Section 2.1.

This one-level algorithm is hereafter referred to as PK21. For this algorithm, one readily obtain that

$$T_{PK21}(N) = \left(1 + \frac{3}{n} - \frac{1}{n^2}\right) N^3 + \left(2n + 5 - \frac{5}{n}\right) N^2,$$

which has minimum near $n = O(N^{1/2})$. However, the actual constant within this "O" should be adjusted when running the corresponding PK21 code on a specific computer (see Section 5).

If one decides to use recursive calls, Step 5 in the above pseudo-code should be unrolled twice:

do $i = 1, n/2$
  do $j = 1, n$
    do $k = 1, n$
      if $(i \neq j \neq k)$ **then**
      $S1 := F_{i,k} + F_{k,n+j}$
      $S2 := G_{k,j} + G_{n+j,i}$
      $T1 := F_{n+1-i,n+1-k} + F_{n+1-k,2n+1-j}$
      $T2 := G_{n+1-k,n+1-j} + G_{2n+1-j,n+1-i}$
      $P := S1 \cdot S2, \;\; Q := T1 \cdot T2$
      $H_{i,j} := H_{i,j} + P$
      $H_{k,i} := H_{k,i} + P$
      $H_{n+1-i,n+1-j} := H_{n+1-i,n+1-j} + Q$
      $H_{n+1-k,n+1-i} := H_{n+1-k,n+1-i} + Q$
      end if
    end do
  end do
end do

For the recursive algorithm we have

$$T_2(M) = \alpha_1(n)\frac{M^2}{2n^2} + \alpha_2(n)\frac{M^2}{n^2} + \frac{\mu(n)}{2}T_2(M/n),$$

where $M = n^{k-1}l$ and $T_2(l) = 2l^3 - 2l^2$. We need the following simple technical result.

Lemma *(FMM Recursion). Let $T(l)$ be given, $M = n^m l$, and*

$$T(M) = \beta T(M/n) + \gamma n^{-2} M^2$$

for some constants $\beta > n^2$ and γ. Then

$$T(M) = \left(T(l) + \sigma l^2\right)\beta^m - \sigma M^2,$$

where $\sigma = \gamma/(\beta - n^2)$.

Corollary. *Under the assumptions of the FMM Recursion Lemma, it holds that*

$$T(M) = \left(T(l)l^{-\omega} + \sigma l^{2-\omega}\right) M^\omega - \sigma M^2,$$

where

$$\omega = \frac{\log \beta}{\log n}.$$

In our case,

$$\beta = \frac{\mu(n)}{2} = (n^3 + 3n^2)/2, \qquad \gamma = \frac{\alpha_1(n)}{2} + \alpha_2(n) = 3n^3 + 7n^2 - 4n,$$

and, consequently,

$$\omega = \frac{\log((n^3 + 3n^2)/2)}{\log n}, \qquad \sigma = \frac{6n^2 + 14n - 8}{n^2 + n}.$$

Table 1: PK2 exponents $\omega(n)$

n	$\frac{n^3+3n^2}{2}$	$\log \frac{n^3+3n^2}{2} / \log n$
8	352	2.819810
10	650	2.812913
12	1080	2.810856
14	1666	2.810920
16	2432	2.811981
18	3402	2.813520
20	4600	2.815275
22	6050	2.817112
24	7776	2.818957
26	9802	2.820770

Applying the lemma and using $M = N/n,\ \ m = k-1,\ \ T_2(l) = 2l^3 - 2l^2$, we obtain

$$T_2(N/n) = \left(2l^3 + \frac{4n^2+12n-8}{n^2+n} l^2\right) \left(\frac{n^3+3n^2}{2}\right)^{k-1} - \frac{6n^2+14n-8}{n^2+n}(N/n)^2.$$

Insert this into the formula for T_{PK2} and after some simplifications, obtain

$$T_{PK2}(N) = \frac{n^2+3n-1}{n^2+3n} \left(2l^{3-\omega} + 4\frac{n^2+3n-2}{n^2+n} l^{2-\omega}\right) N^\omega - \frac{7n^3+12n^2-13n+4}{n^3+n^2} N^2.$$

For $n = 12$ we obtain $\omega \le 2.81086$ and

$$T_{PK2}(N) = \frac{179}{180} \left(2l^{3-\omega} + \frac{178}{45} l^{2-\omega}\right) N^\omega - \frac{1709}{234} N^2.$$

Table 1 shows $\omega(n)$ for the nearby even n. (Although the smallest value is $\omega(13)$, odd n are less convenient for coding.) Finally, choosing $l = 8$, so $N = 8 \cdot 12^k$, we obtain

$$T(N) \le 3.683 N^{2.81086} - 7.303 N^2.$$

This estimate should be compared with similar bounds $T_S(N) = 3.895 N^{2.80736} - 6N^2,\ \ N = 10 \cdot 2^k$, for the Strassen algorithm [13] and $T_{SW}(N) = 3.732 N^{2.80736} - 5N^2,\ \ N = 8 \cdot 2^k$, for a similar algorithm by Winograd. As one can see, for our algorithm the power of N in the leading term is slightly larger, but the reduction in the multiplicative constant is essential to make the algorithm competitive. Indeed, $T_{PK2}(N) \le T_S(N)$ if $N \le 9 \cdot 10^6$ and $T_{PK2}(N) \le 1.02 T_{SW}(N)$ if $N \le 14500$.

Remark. In [11], a somewhat underestimated operation count was mistakenly given for a similar MM algorithm; this was a consequence of mixing together the input-type and output-type additions/subtractions (which have different complexity when the blocks are non-square).

3 Three disjoint product based algorithms

Our next construction of fast MM algorithms relies on aggregation/cancellation techniques and on two-level block matrix structure; the aggregates involve quadruple rather than double indexing of matrix entries. This enables us to develop the so-called 3-Procedure (for computing Three Disjoint MM's), and we refer to the resulting methods for single matrix product as the "PK3 algorithms".

In our exposition, we follow the notations of [10], Section 5. Our basic problem is the calculation of three disjoint $n \times n$ matrix products

$$C^0 = A^0 B^0, \qquad W^0 = U^0 V^0, \qquad Z^0 = X^0 Y^0, \tag{1}$$

and, for simplicity, we let n be even,

$$n = 2m - 2. \tag{2}$$

We first describe preprocessing of the input matrices similar to that in [10].

3.1 Reduction to the case of zero row and column sums

We assume, for simplicity, that the entries of $A^0, \ldots, Y^0$ are real numbers. (In general, these matrices can be composed of rectangular submatrices, and then our formulae (3)-(8) would still apply.) Write

$$A^0 = \begin{bmatrix} A^0_{11} & A^0_{12} \\ A^0_{21} & A^0_{22} \end{bmatrix}, \qquad B^0 = \begin{bmatrix} B^0_{11} & B^0_{12} \\ B^0_{21} & B^0_{22} \end{bmatrix},$$

and similarly for U^0, V^0, X^0, Y^0, where each of the four submatrices has the size $(m-1) \times (m-1)$, cf. (2).

Let I be the $(m-1) \times (m-1)$ identity matrix and let

$$u_0^T = [1 \ldots 1] \qquad \text{and} \qquad u^T = [u_0^T \ \ 1]$$

denote the $(m-1)$- and m-vectors composed of all ones, respectively. Define the matrices

$$L = \begin{bmatrix} I \\ -u_0^T \end{bmatrix}, \qquad R = \begin{bmatrix} I - \frac{1}{m} u_0 u_0^T & -\frac{1}{m} u_0 \end{bmatrix}$$

of sizes $m \times (m-1)$ and $(m-1) \times m$, respectively. Noting that

$$\begin{aligned} u^T L &= 0, \\ Ru &= 0, \\ RL &= I, \end{aligned}$$

consider the transformations

$$A_{11} = L A^0_{11} R, \qquad B_{11} = L B^0_{11} L^T$$

of the blocks A^0_{11} and B^0_{11}. Then clearly,

$$u^T A_{11} = 0, \qquad A_{11} u = 0,$$

$$u^T B_{11} = 0, \qquad B_{11} u = 0,$$

$$A_{11} B_{11} = (L A^0_{11} R)(L B^0_{11} L^T) = L (A^0_{11} B^0_{11}) L^T = \begin{bmatrix} A^0_{11} B^0_{11} & * \\ * & * \end{bmatrix}.$$

Now, replace each of the four $(m-1) \times (m-1)$ blocks A^0_{ij} in A^0 and B^0_{ij} in B^0 by the transformed $m \times m$ blocks A_{ij} and B_{ij} with zero row and column sums and arrive at the matrices

$$A = \begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} A^0 \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} = \begin{bmatrix} L A^0_{11} R & L A^0_{12} R \\ L A^0_{21} R & L A^0_{22} R \end{bmatrix},$$

$$B = \begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} B^0 \begin{bmatrix} L^T & 0 \\ 0 & L^T \end{bmatrix} = \begin{bmatrix} LB^0_{11}L^T & LB^0_{12}L^T \\ LB^0_{21}L^T & LB^0_{22}L^T \end{bmatrix}.$$

The product

$$A^0B^0 = C^0 = \begin{bmatrix} C^0_{11} & C^0_{12} \\ C^0_{21} & C^0_{22} \end{bmatrix}$$

is recovered from the $(m-1)\times(m-1)$ leading submatrices of the $m \times m$ blocks $C_{11}, C_{12}, C_{21}, C_{22}$ in the product

$$C = AB = \begin{bmatrix} L & 0 \\ 0 & L \end{bmatrix} A^0B^0 \begin{bmatrix} L^T & 0 \\ 0 & L^T \end{bmatrix} = \begin{bmatrix} C^0_{11} & * & C^0_{12} & * \\ * & * & * & * \\ C^0_{21} & * & C^0_{22} & * \\ * & * & * & * \end{bmatrix}.$$

To conclude this section, let us specify the transformation $H = LGR$ of an $(m-1)\times(m-1)$-submatrix G of a left multiplier (e.g., $G = A^0_{11}$ into $H = A_{11}$):

$$H_{im} = -\frac{1}{m}\sum_{j=1}^{m-1} G_{ij}, \quad i = 1, \ldots, m-1; \tag{3}$$

$$H_{ij} = G_{ij} + H_{im}, \quad i = 1, \ldots, m-1, \;\; j = 1, \ldots, m-1; \tag{4}$$

$$H_{mj} = -\sum_{i=1}^{m-1} H_{ij}, \quad j = 1, \ldots, m-1. \tag{5}$$

For the right multipliers, the transformation of an $(m-1)\times(m-1)$-submatrix G (e.g., $G = B^0_{11}$ into $H = B_{11}$) given by $H = LGL^T$ is even simpler:

$$H_{im} = -\sum_{j=1}^{m-1} G_{ij}, \quad i = 1, \ldots, m-1; \tag{6}$$

$$H_{ij} = G_{ij}, \quad i = 1, \ldots, m-1, \;\; j = 1, \ldots, m-1; \tag{7}$$

$$H_{mj} = -\sum_{i=1}^{m-1} H_{ij}, \quad j = 1, \ldots, m-1. \tag{8}$$

Due to (5) and (8), we avoid computing the matrices $A_{pm+m,qm+m}$, $B_{pm+m,qm+m}$, $\ldots$, $Y_{pm+m,qm+m}$, $p = 0, 1$, $q = 0, 1$, which are not used in our algorithm (as one can see in the next section).

Remark. The above preprocessing algorithm is different from that of [10] (Section 5) where the same transformation is made for both left and right multiplicands (e.g., for A^0 and B^0, respectively), followed by a postprocessing stage. In our case, there is no numerical postprocessing, and the operation count corresponding to (3)-(8) is, therefore, only about 5/8 times that involved in the preprocessing in [10].

To obtain our next algorithm for three disjoint matrix products, we removed some redundant operations in the algorithm in Section 5 of [10], change some signs in the aggregates, and reordered rows and columns in the transformed matrices $A, B, \ldots, Y$.

3.2 A compact form of the aggregation-cancellation algorithm

Suppose all six input matrices $A^0, \ldots, Y^0$ are preprocessed as in the preceding section. Then the following three disjoint products,

$$C = AB, \qquad W = UV, \qquad Z = XY,$$

are actually computed, where each matrix has size $(n+2)\times(n+2)$ for $n+2 = 2m$. For the transformed matrices we have the following "zero-sum" relationships:

$$\sum_{i=1}^{m} A_{pm+i,qm+j} = 0, \quad 1 \le j \le m; \qquad \sum_{j=1}^{m} A_{pm+i,qm+j} = 0, \quad 1 \le i \le m; \qquad p = 0,1, \quad q = 0,1;$$

$$\sum_{j=1}^{m} B_{qm+j,rm+k} = 0, \quad 1 \le k \le m; \qquad \sum_{k=1}^{m} B_{qm+j,rm+k} = 0, \quad 1 \le j \le m; \qquad q = 0,1, \quad r = 0,1;$$

$$\sum_{j=1}^{m} U_{rm+j,pm+k} = 0, \quad 1 \le k \le m; \qquad \sum_{k=1}^{m} U_{rm+j,pm+k} = 0, \quad 1 \le j \le m; \qquad r = 0,1, \quad p = 0,1;$$

$$\sum_{k=1}^{m} V_{pm+k,qm+i} = 0, \quad 1 \le i \le m; \qquad \sum_{i=1}^{m} V_{pm+k,qm+i} = 0, \quad 1 \le k \le m; \qquad p = 0,1, \quad q = 0,1;$$

$$\sum_{k=1}^{m} X_{qm+k,rm+i} = 0, \quad 1 \le i \le m; \qquad \sum_{i=1}^{m} X_{qm+k,rm+i} = 0, \quad 1 \le k \le m; \qquad q = 0,1, \quad r = 0,1;$$

$$\sum_{i=1}^{m} Y_{rm+i,pm+j} = 0, \quad 1 \le j \le m; \qquad \sum_{j=1}^{m} Y_{rm+i,pm+j} = 0, \quad 1 \le i \le m; \qquad r = 0,1, \quad p = 0,1.$$

To devise our algorithm, consider the $8m^3 = (n+2)^3$ products (the so called aggregates, cf. [10])

$$\begin{aligned} M_{ijk}^{pqr} = ((-1)^r A_{pm+i,qm+j} \quad &+(-1)^q U_{rm+j,pm+k} \quad +(-1)^p X_{qm+k,rm+i}) \\ (B_{qm+j,rm+k} \quad &+ \quad V_{pm+k,qm+i} \quad + \quad Y_{rm+i,pm+j}), \end{aligned} \tag{9}$$

$$1 \le i \le m, \quad 1 \le j \le m, \quad 1 \le k \le m, \qquad p = 0,1, \quad q = 0,1, \quad r = 0,1.$$

Each of these products equals the sum of the following nine terms:

$$\begin{aligned} M_{ijk}^{pqr} \;=\;& (-1)^r A_{pm+i,qm+j} B_{qm+j,rm+k} + (-1)^r A_{pm+i,qm+j} V_{pm+k,qm+i} + (-1)^r A_{pm+i,qm+j} Y_{rm+i,pm+j} \\ +\;& (-1)^q U_{rm+j,pm+k} B_{qm+j,rm+k} + (-1)^q U_{rm+j,pm+k} V_{pm+k,qm+i} + (-1)^q U_{rm+j,pm+k} Y_{rm+i,pm+j} \\ +\;& (-1)^p X_{qm+k,rm+i} B_{qm+j,rm+k} + (-1)^p X_{qm+k,rm+i} V_{pm+k,qm+i} + (-1)^p X_{qm+k,rm+i} Y_{rm+i,pm+j}. \end{aligned}$$

Sum these quantities over q, j, over p, k, and over r, i, note that the sums of the type

$$\sum_{q,j} (-1)^q U_{rm+j,pm+k} Y_{rm+i,pm+j}, \quad \sum_{p,k} (-1)^p X_{qm+k,rm+i} B_{qm+j,rm+k}, \quad \sum_{r,i} (-1)^r A_{pm+i,qm+j} V_{pm+k,qm+i}$$

are equal to zero (due to the so called cancellation effect, cf. [10]), and take into account the zero sum properties of the input matrices. This produces the following expressions for $(AB)_{pm+i,rm+k}$, $(UV)_{rm+j,qm+i}$, and $(XY)_{qm+k,pm+j}$, respectively, which define the desired algorithm:

$$\begin{aligned} (AB)_{pm+i,rm+k} \;=\;& (-1)^r \sum_{q,j} M_{ijk}^{pqr} - (-1)^{p+r} m \sum_{q} X_{qm+k,rm+i} V_{pm+k,qm+i} \\ -\;& \sum_{q,j} A_{pm+i,qm+j} Y_{rm+i,pm+j} - \sum_{q,j} (-1)^{q+r} U_{rm+j,pm+k} B_{qm+j,rm+k}, \end{aligned} \tag{10}$$

$$1 \le i \le m-1, \qquad 1 \le k \le m-1; \qquad p = 0,1, \quad r = 0,1;$$

$$
\begin{aligned}
(UV)_{rm+j,qm+i} &= (-1)^q \sum_{p,k} M_{ijk}^{pqr} - (-1)^{r+q} m \sum_p A_{pm+i,qm+j} Y_{rm+i,pm+j} \\
&- \sum_{p,k} U_{rm+j,pm+k} B_{qm+j,rm+k} - \sum_{p,k} (-1)^{p+q} X_{qm+k,rm+i} V_{pm+k,qm+i}, \\
1 \le j \le m-1, &\qquad 1 \le i \le m-1; \qquad r = 0,1, \quad q = 0,1;
\end{aligned} \tag{11}
$$

$$
\begin{aligned}
(XY)_{qm+k,pm+j} &= (-1)^p \sum_{r,i} M_{ijk}^{pqr} - (-1)^{q+p} m \sum_r U_{rm+j,pm+k} B_{qm+j,rm+k} \\
&- \sum_{r,i} X_{qm+k,rm+i} V_{pm+k,qm+i} - \sum_{r,i} (-1)^{r+p} A_{pm+i,qm+j} Y_{rm+i,pm+j}, \\
1 \le k \le m-1, &\qquad 1 \le j \le m-1; \qquad q = 0,1, \quad p = 0,1.
\end{aligned} \tag{12}
$$

In the next section, we estimate arithmetic complexity of this algorithm.

Remark. The above algorithm can be easily generalized to the case where the sizes of the input matrices are $n_1 \times n_2$, $n_2 \times n_3$, $n_2 \times n_3$, $n_3 \times n_1$, $n_3 \times n_1$, and $n_1 \times n_2$ for A^0, B^0, U^0, V^0, X^0, and Y^0, respectively, as in [10].

Remark. For each fixed triple i, j, k, the eight products (9) obtained with different p, q, r correspond exactly to the eight products $P_1, \ldots, P_8$ introduced at p.572 of [10] as follows:

$$
P_1 = M^{000}, \qquad P_2 = M^{010}, \qquad P_3 = M^{100}, \qquad P_4 = M^{001},
$$

$$
P_5 = -M^{111}, \quad P_6 = -M^{101}, \quad P_7 = -M^{011}, \quad P_8 = -M^{110}.
$$

3.3 Asymptotics for bilinear multiplicative cost

The algorithm can be summarized as follows.

- Split the matrices properly and apply transformation (3)-(8) to each of the 24 blocks $A_{11}^0, \ldots, Y_{22}^0$; then perform all the matrix additions involved in (9);
- Perform the (bilinear) matrix multiplications involved in (9)-(12) (in general, either a recursive call, or the trivial algorithm, or another algorithm can be applied here);
- Perform all additions involved in (10)-(12) (as follows from Section 3, for the resulting products C, W, and Z, the bordering rows and columns introduced at the preprocessing stage need not be calculated).

This rather rough sketch makes it possible to estimate the number of bilinear multiplications involved. To estimate the number of linear operations (additions, subtractions, and multiplications by scalars) and the working memory usage, we have to reorder the computations properly, see subsections 3.4-3.6.

First, note that for all p, q, r there is no actual need to calculate the products

$$
M_{imm}^{pqr}, \quad M_{mim}^{pqr}, \quad M_{mmi}^{pqr}, \qquad i = 1, \ldots, m,
$$

Table 2: PK3 exponents $\omega(n)$

n	$\frac{n^3+12n^2+24n}{3}$	$\log \frac{n^3+12n^2+24n}{3} / \log n$
12	1248	2.869040
18	3384	2.811685
24	7104	2.790517
30	12840	2.781468
36	21024	2.777555
42	32088	2.776125
48	46464	2.775995
54	64584	2.776577
60	86880	2.777559
66	113784	2.778763
72	145728	2.780085
78	183144	2.781464
84	226464	2.782860
90	276120	2.784249
96	332544	2.785617

and

$$A_{pm+m,qm+m}Y_{rm+m,pm+m}, \quad U_{rm+m,pm+m}B_{qm+m,rm+m}, \quad X_{qm+m,rm+m}V_{pm+m,qm+m}$$

since these quantities are never used in (10)-(12).

The remaining products M_{ijk}^{pqr} and the correction terms of the type $A_{pm+i,qm+j}Y_{rm+i,pm+j}$ are computed by using $8(m^3-3m+2)$ and $24(m^2-1)$ multiplications, respectively. Add these quantities and recall $2m = n+2$ to yield the following expression for the total number of bilinear multiplications:

$$\begin{aligned} \mu(n) &= 8m^3 + 24m^2 - 24m - 8 \\ &= n^3 + 12n^2 + 24n. \end{aligned}$$

This number is divisible by 3 whenever

$$n = 6k, \quad k = 1, 2, \ldots$$

(recall that we already assumed that n is even). Hence, the MM's of smaller size in this construction can be regrouped again in triples. Assuming that $N = n^k l, \ k \geq 1, \ l \geq 1$, one readily obtains the following recurrence relation

$$b(N) = \frac{\mu(n)}{3} b\left(\frac{N}{n}\right) = \left(\frac{n^3}{3} + 4n^2 + 8n\right) b\left(\frac{N}{n}\right) = \ldots = \left(\frac{n^3}{3} + 4n^2 + 8n\right)^k b(l),$$

where $b(N)$ is the number of bilinear multiplications in the resulting recursive algorithm for three disjoint products of $N \times N$ matrices. For $n = 48$, fixed l, and $k \to \infty$, we obtain an algorithm with asymptotic complexity

$$T(N) = O(N^{2.7760}).$$

In general, the "base n" algorithm has the asymptotic complexity $O(N^{\omega(n)})$, where $\omega(n) = \log_n(\mu(n)/3))$ (cf. Section 2.1); some exponents $\omega(n)$ are shown in Table 2.

The above asymptotics hold for all N since the limitation $N = n^k l$ can be relaxed using simple bordering of the original matrices by zeroes [13]. Such techniques are also of practical use, see Section 4 below, where the case of rectangular matrices is considered.

3.4 Implementation details for 3-procedure

Next we study the computational scheme for Three Disjoint MM's in some detail to estimate the number of linear operations involved and the working memory used.

Let us introduce the following more compact notation using 4-dimensional indexing:

$$A_{pm+i,qm+j} = A_{ij}^{pq}, \ \ldots, \ Z_{qm+k,pm+j} = Z_{kj}^{qp}.$$

The main part of the algorithm described by (9)-(12) can be implemented as shown by the following pseudo-code:

1. **do** $p = 0, 1;\ q = 0, 1:$
   do $i = 1, \ldots, m:$
   $C_{mi}^{pq} := 0,\ C_{im}^{pq} := 0,\ W_{mi}^{pq} := 0,\ W_{im}^{pq} := 0,\ Z_{mi}^{pq} := 0,\ Z_{im}^{pq} := 0$
   end do
   end do
2. **do** $p = 0, 1;\ q = 0, 1;\ r = 0, 1:$
   do $i = 1, \ldots, m;\ j = 1, \ldots, m:$
   if ($i < m$ **or** $j < m$ **)** $C_{mm}^{00} := A_{ij}^{pq} Y_{ij}^{rp}$
   if ($i < m$ **and** $j < m$ **) then**
   if ($p = 0$ **) then**
   $W_{ji}^{rq} := C_{mm}^{00}$
   else
   $W_{ji}^{rq} := -(-1)^{q+r} m (W_{ji}^{rq} + C_{mm}^{00})$
   end if
   end if
   if ($i < m$ **)** $C_{im}^{pr} := C_{im}^{pr} + C_{mm}^{00}$
   if ($j < m$ **)** $C_{mj}^{qp} := C_{mj}^{qp} + (-1)^{p+r} C_{mm}^{00}$
   end do
   do $j = 1, \ldots, m;\ k = 1, \ldots, m:$
   if ($j < m$ **or** $k < m$ **)** $C_{mm}^{00} := U_{jk}^{rp} B_{jk}^{qr}$
   if ($j < m$ **and** $k < m$ **) then**
   if ($r = 0$ **) then**
   $Z_{kj}^{qp} := C_{mm}^{00}$
   else
   $Z_{kj}^{qp} := -(-1)^{p+q} m (Z_{kj}^{qp} + C_{mm}^{00})$
   end if
   end if
   if ($j < m$ **)** $W_{jm}^{rq} := W_{jm}^{rq} + C_{mm}^{00}$
   if ($k < m$ **)** $W_{mk}^{pr} := W_{mk}^{pr} + (-1)^{q+r} C_{mm}^{00}$
   end do
   do $k = 1, \ldots, m;\ i = 1, \ldots, m:$
   if ($i < m$ **or** $k < m$ **)** $C_{mm}^{00} := X_{ki}^{qr} V_{ki}^{pq}$
   if ($i < m$ **and** $k < m$ **) then**
   if ($q = 0$ **) then**
   $C_{ik}^{pr} := C_{mm}^{00}$

else

$C_{ik}^{pr} := -(-1)^{q+r} m (C_{ik}^{pr} + C_{mm}^{00})$

end if

end if

if ($k < m$ **)** $Z_{km}^{qp} := Z_{km}^{qp} + C_{mm}^{00}$

if ($i < m$ **)** $Z_{mi}^{rq} := Z_{mi}^{rq} + (-1)^{p+q} C_{mm}^{00}$

end do

end do

3. **do** $p = 0, 1;\ \ q = 0, 1;\ \ r = 0, 1:$

do $i = 1, \ldots, m;\ \ j = 1, \ldots, m;\ \ k = 1, \ldots, m:$

if (($i < m$ **and** $j < m$) **or** ($j < m$ **and** $k < m$) **or** ($k < m$ **and** $i < m$) **) then**

$C_{mm}^{01} := (-1)^r A_{ij}^{pq} + (-1)^q U_{jk}^{rp} + (-1)^p X_{ki}^{qr}$

$C_{mm}^{10} := B_{jk}^{qr} + V_{ki}^{pq} + Y_{ij}^{rp}$

$C_{mm}^{00} := C_{mm}^{01} C_{mm}^{10}$

end if

if ($i < m$ **and** $k < m$ **)** $C_{ik}^{pr} := C_{ik}^{pr} + (-1)^r C_{mm}^{00}$

if ($i < m$ **and** $j < m$ **)** $W_{ji}^{rq} := W_{ji}^{rq} + (-1)^q C_{mm}^{00}$

if ($j < m$ **and** $k < m$ **)** $Z_{kj}^{qp} := Z_{kj}^{qp} + (-1)^p C_{mm}^{00}$

end do

end do

4. **do** $p = 0, 1;\ \ r = 0, 1:$

do $i = 1, \ldots, m-1;\ \ k = 1, \ldots, m-1:$

$C_{ik}^{pr} := C_{ik}^{pr} - C_{im}^{pr} - W_{mk}^{pr}$

end do

end do

do $q = 0, 1;\ \ r = 0, 1:$

do $j = 1, \ldots, m-1;\ \ i = 1, \ldots, m-1:$

$W_{ji}^{rq} := W_{ji}^{rq} - W_{jm}^{rq} - Z_{mi}^{rq}$

end do

end do

do $p = 0, 1;\ \ q = 0, 1:$

do $k = 1, \ldots, m-1;\ \ j = 1, \ldots, m-1:$

$Z_{kj}^{qp} := Z_{kj}^{qp} - Z_{km}^{qp} - C_{mj}^{qp}$

end do

end do

We use the bordering rows of the resulting matrices C^{pr}, W^{rq}, Z^{qp} as temporary variables for the accumulation of appropriate sums. The symbol ':=' denotes in-place updating, so our symbols $C_{ik}^{pr}, W_{ji}^{rq}, Z_{kj}^{qp}$ indicate certain storage areas rather than algebraic terms. Obviously, the required memory does not exceed the amount of bordering introduced for all the input and output matrices. We choose $n = 2m - 2$ of the order of tens, so this typically comprises only a moderate fraction (not larger than $(4n + 4)/(n + 2)^2$) of the total input data volume.

We have not commented above on the grouping of matrix products into triples as implied by the recursion. However, for matrix sizes not larger than 10000, the one-level scheme appears to be most efficient, at least for many modern RISC computers (see sections 4 and 5). In this case, no grouping by triples is required, whereas grouping of pairs should be done if the 2-procedure instead of the standard MM is used at the inner level; the latter choice seems to be good for very large matrix sizes.

3.5 The number of scalar multiplications and additions: exact operation count

To show that the algorithm is practically competitive, e.g., with the ones presented in [13], [10], [11], we should estimate the actual number of linear operations.

The number of "linear operations" (i.e., matrix additions, subtractions, and multiplications by scalars m^{-1} or m required for performing (3) or computing the correction terms in (8)-(10), respectively) can be estimated as follows.

- Steps (3)-(8) take $12(5m^2 - 13m + 8)$ operations applied to input-type blocks;
- Step (9) involves $32(m^3 - 3m + 2)$ operations applied to input-type blocks;
- Steps (10)-(12) can be performed in $24(m^3 + 2m^2 - 6m + 3)$ operations applied to output-type blocks.

Substituting $2m = n + 2$, one obtains the estimates

$$\begin{aligned} \alpha_1(n) &= 4n^3 + 39n^2 - 18n \qquad \text{and} \\ \alpha_2(n) &= 3n^3 + 30n^2 + 12n \end{aligned}$$

for linear operations performed on the input-type and the output-type blocks, respectively. In Section 3.7, the above formulas are used as the basis for the operation count for a regular level of recursion in the above algorithm.

3.6 An algorithm for a single matrix product

The above procedure can be applied to multiply a single pair of $N \times N$ matrices with scalar coefficients quite similarly to the approach of Section 3 (cf.[11]).

Consider the product $H = FG$ of two square $N \times N$ matrices. Let N be an integer multiple of 3. Split the columns of F and the rows of G into three equal blocks each, that is,

$$F = \left[\begin{array}{ccc} A & X & U \end{array} \right], \qquad G = \left[\begin{array}{c} B \\ Y \\ V \end{array} \right],$$

where A, X, U and B, Y, V have the sizes $N \times N/3$ and $N/3 \times N$, respectively. Then, by computing

$$C = AB, \qquad Z = XY, \qquad W = UV,$$

one obtains the required product as

$$H = C + Z + W,$$

and the problem is thus reduced to a triple of disjoint matrix multiplications, followed by a pair of $N \times N$ matrix additions.

We keep working memory as small as in section 3.4, by accumulating all three products simultaneously in the course of calculations. Indeed, as one can see from the pseudo-code below, after adding the bordering block rows and columns to the input and output matrices, all the subsequent computations can be made in-place again.

Write as above

$$F_{pm+i,qm+j} = F_{ij}^{pq}, \;\; G_{qm+j,rm+k} = G_{ij}^{pq}, \;\; H_{pm+i,rm+k} = H_{kj}^{qp},$$

and summarize the main part of the algorithm (performed after completing the bordering) as follows:

1. **do** $p = 0, 1;\ \ q = 0, 1:$

do $i = 1, \ldots, m;\ \ j = 1, \ldots, m:$

$H^{pq}_{ij} := 0$

end do

end do

2. **do** $p = 0, 1;\ \ q = 0, 1;\ \ r = 0, 1:$

do $i = 1, \ldots, m;\ \ j = 1, \ldots, m:$

if ($i < m$ **or** $j < m$) $\ H^{00}_{mm} := F^{pq}_{ij} G^{rp}_{2m+i,j}$

if ($i < m$ **and** $j < m$) $H^{rq}_{ji} := H^{rq}_{ji} - (-1)^{q+r} H^{00}_{mm}$

if ($i < m$) $\ H^{pr}_{im} := H^{pr}_{im} + H^{00}_{mm}$

if ($j < m$) $\ H^{qp}_{mj} := H^{qp}_{mj} + (-1)^{p+r} H^{00}_{mm}$

end do

do $j = 1, \ldots, m;\ \ k = 1, \ldots, m:$

if ($j < m$ **or** $k < m$) $\ H^{00}_{mm} := F^{rp}_{j,m+k} G^{qr}_{jk}$

if ($j < m$ **and** $k < m$) $H^{qp}_{kj} := H^{qp}_{kj} - (-1)^{p+q} H^{00}_{mm}$

if ($j < m$) $\ H^{rq}_{jm} := H^{rq}_{jm} + H^{00}_{mm}$

if ($k < m$) $\ H^{pr}_{mk} := H^{pr}_{mk} + (-1)^{q+r} H^{00}_{mm}$

end do

do $k = 1, \ldots, m;\ \ i = 1, \ldots, m:$

if ($i < m$ **or** $k < m$) $\ H^{00}_{mm} := F^{qr}_{k,2m+i} G^{pq}_{m+k,i}$

if ($i < m$ **and** $k < m$) $H^{pr}_{ik} := H^{pr}_{ik} - (-1)^{q+r} H^{00}_{mm}$

if ($k < m$) $\ H^{qp}_{km} := H^{qp}_{km} + H^{00}_{mm}$

if ($i < m$) $\ H^{rq}_{mi} := H^{rq}_{mi} + (-1)^{p+q} H^{00}_{mm}$

end do

end do

3. **do** $p = 0, 1;\ \ q = 0, 1:$

do $i = 1, \ldots, m-1;\ \ j = 1, \ldots, m-1:$

$H^{pq}_{ij} := m H^{pq}_{ij}$

end do

end do

4. **do** $p = 0, 1;\ \ q = 0, 1;\ \ r = 0, 1:$

do $i = 1, \ldots, m;\ \ j = 1, \ldots, m;\ \ k = 1, \ldots, m:$

if ($(i < m$ **and** $j < m)$ **or** $(j < m$ **and** $k < m)$ **or** $(k < m$ **and** $i < m)$) **then**

$H^{01}_{mm} := (-1)^r F^{pq}_{ij} + (-1)^q F^{rp}_{j,m+k} + (-1)^p F^{qr}_{k,2m+i}$

$H^{10}_{mm} := G^{qr}_{jk} + G^{pq}_{m+k,i} + G^{rp}_{2m+i,j}$

$H^{00}_{mm} := H^{01}_{mm} H^{10}_{mm}$

end if

if ($i < m$ **and** $k < m$) $\ H^{pr}_{ik} := H^{pr}_{ik} + (-1)^r H^{00}_{mm}$

if ($i < m$ **and** $j < m$) $\ H^{rq}_{ji} := H^{rq}_{ji} + (-1)^q H^{00}_{mm}$

if ($j < m$ **and** $k < m$) $\ H^{qp}_{kj} := H^{qp}_{kj} + (-1)^p H^{00}_{mm}$

end do

end do

5. **do** $p = 0, 1;\ \ q = 0, 1:$
do $i = 1, \ldots, m-1;\ \ j = 1, \ldots, m-1:$
$H_{ij}^{pq} := H_{ij}^{pq} - H_{im}^{pq} - H_{mj}^{pq}$
end do
end do

Fortunately, the algorithm for a single MM appears to be even more compact than the generic Three Disjoint Product procedure. Here we use $4m^2$ redundant additions due to the simplistic initialization at Step 1 above, but we save many scalar multiplications by m, performing them just once at Step 3.

The latter algorithm actually presents a procedure for multiplying $n \times 3n$ matrix F by $3n \times n$ matrix G and requires $\mu(n) = n^2 + 12n + 24$ bilinear multiplications (the same as above) and

$$\begin{aligned} \tilde{\alpha}_1(n) &= 4n^3 + 39n^2 - 18n \qquad \text{and} \\ \tilde{\alpha}_2(n) &= 3n^3 + 27n^2 + 9n \end{aligned}$$

linear operations performed on input-type and output-type blocks, respectively. The above formulas are used in the next section to estimate the complexity for the starting level of recursion in the PK3 algorithm.

Note that in the above algorithm, the preprocessing stage of Section 3 is made separately for every $n \times n$ block, a triple of which composes F or G.

3.7 Recursive algorithm and its best-case performance

Let the (block) sizes of all these matrices be $n \times n$. This corresponds to the assumption that $N = nl$, where $n = 6k$ (as was assumed earlier) and l is an integer multiple of 3, so each matrix A, X, U and B, Y, V is partitioned as a square $n \times n$ block matrix composed of $l \times l/3$ and $l/3 \times l$ submatrices, respectively ($l = N/n$).

Hence, the above recursion scheme readily applies. Noting that the recursive 3-Procedure and the corresponding PK3 method for square matrix multiplication differ only in their initialization stage, one can formally write

$$T_{PK3}(N) = -\frac{3n+3}{n}N^2 + T_3(N). \tag{13}$$

The input-type and output-type linear operations take $(N/n)^2/3$ and $(N/n)^2$ flops, respectively, so the 3-Procedure (i.e., Three Disjoint products of $N \times N/3$ by $N/3 \times N$ matrices) uses

$$T_3(N) = \left(\frac{\alpha_1(n)}{3} + \alpha_2(n)\right) N^2/n^2 + \frac{\mu(n)}{3} T_3(N/n)$$

$$= \frac{13n^3 + 129n^2 + 18n}{3} N^2/n^2 + \frac{n^3 + 12n^2 + 24n}{3} T_3(N/n)$$

flops, and

$$T_3(l) = 2l^3 - 3l^2, \qquad l << N.$$

Applying now the FMM recursion Lemma, one obtains

$$T_3(N) = \left(2l^{3-\omega} + \frac{10n^2 + 102n - 54}{n^2 + 9n + 24} l^{2-\omega}\right) N^\omega - \frac{13n^2 + 129n + 18}{n^2 + 9n + 24} N^2,$$

and therefore,

$$T_{PK3}(N) = \left(2l^{3-\omega} + \frac{10n^2 + 102n - 54}{n^2 + 9n + 24} l^{2-\omega}\right) N^\omega - \frac{16n^3 + 159n^2 + 117n + 72}{n^3 + 9n^2 + 24n} N^2,$$

where

$$\omega = \frac{\log\left(\frac{n^3}{3} + 4n^2 + 8n\right)}{\log n},$$

To minimize ω, set $n = 48$ to obtain

$$T_{PK3}(N) = \left(2l^{3-\omega} + \frac{4647}{460}l^{2-\omega}\right) N^{\omega} - \frac{29743}{1840}N^2.$$

With optimum $l = 18$, this yields

$$T_{PK3}(N) \leq 4.894N^{2.7760} - 16.165N^2.$$

With respect to the total operations count, the above PK3 algorithm is quite competitive with Strassen's, for which

$$T_S(N) \simeq 3.895N^{2.80736} - 6N^2,$$

and even with Winograd's one, which has

$$T_{SW}(N) \simeq 3.732N^{2.80736} - 5N^2.$$

For instance, $T_{PK3}(N) \leq 1.01T_S(N)$ for all $N \geq 701$ and $T_{PK3}(N) \leq 1.05T_{SW}(N)$ for all $N \geq 803$. (Of course, for considerably greater N, one has "$\ll$" relation due to smaller PK3 exponent.)

3.8 Estimating numerical stability of 3-Procedure

As we show in Section 5, the presented matrix multiplication algorithm (similar to the one in [11]) demonstrates very good numerical stability due to the structural advantage given by the "long base" recursions. This is an essential property of the algorithms based on the schemes in [14], [15], [10], whereas the Strassen type algorithms use "base two" recursions and therefore are much less numerically stable. The techniques for the estimation of stability of MM algorithms can be found in [19], [20], [21]. The general approach to theoretical estimation of the error growth factor for the floating-point implementation of such algorithms can be found in [12], where the whole class of Strassen-like algorithms was analyzed.

Using the standard techniques [12], [19], [21] for estimating the numerical error growth for the 3-Procedure, one can obtain the following result (quite similar to that presented for the 2-Procedure in [11]). If we denote by τ the machine tolerance (usually near 10^{-15} and 10^{-7} in double and single precision, respectively) and use the matrix norm

$$\|S\| = \max_{i,j} |(S)_{i,j}|,$$

then the error in the floating point implementation of the 3-Procedure applied to a triple of $N \times N/3$ by $N/3 \times N$ products $C = AB, W = UV, Z = XY$ with $N = n^{k-1}l,\ \ l \leq n$ satisfies the bound

$$\|fl([C|W|Z]) - [C|W|Z]\| \leq c_0 N \exp\left(2\log n + \frac{\log(10 + \frac{20}{n})}{\log n}\log N\right)\tau\|[A|U|X]\|\|[B|V|Y]\| + O(\tau^2).$$

Here and hereafter, $[C|W|Z]$ denotes the $N \times 3N$ matrix having 1×3 block structure, etc. The sketch of the proof is as follows. (We are trying to be as close as possible to the analysis of Strassen's

algorithm in [19],[21].) The floating point model of scalar additions/subtractions and multiplications is

$$\begin{aligned} fl(a \pm b) &= a(1+\alpha) \pm b(1+\beta), \\ fl(ab) &= ab(1+\gamma), \end{aligned}$$

where $|\alpha|, |\beta|, |\gamma| \le \tau$. Together with the simple estimate

$$\|fl(S+T) - (S+T)\| \le 2\tau \|[S|T]\|$$

we use its general form

$$\|fl\left(\sum_{i=1}^{q} S_i\right) - \left(\sum_{i=1}^{q} S_i\right)\| \le \frac{q^2+q-2}{2}\tau \|[S_1|\ldots|S_q]\| + O(\tau^2)$$

and the (easily obtained) error bound for the standard algorithm applied to the product ST of a $p \times q$ matrix S by a $q \times r$ matrix T:

$$\|fl(ST) - ST\| \le \frac{q^2+3q-2}{2}\tau \|S\|\|T\| + O(\tau^2).$$

The latter, taken with $p = r = l, q = l/3$, yields

$$\|fl([AB|UV|XY]) - [AB|UV|XY]\| \le \Omega(l)\tau \|[A|U|X]\|\|[B|V|Y]\| + O(\tau^2),$$

where $\Omega(l) = (l^2 + 9l - 18)/18$, which can be used as the induction basis.

The inductive hypothesis of the same form (and with $\Omega(l)$ replaced by $\Omega(N)$) is then proved for one recursive step of the algorithm (as specified in Section 4.5 above) with

$$\Omega(N) = (10n + 20)\Omega(N/n) + n^2 N$$

in a way quite similar to that used in [19],[21]. In view of $l \le n$, solving this recurrence readily yields

$$\Omega(N) \le c_0 N n^2 \left(10 + \frac{20}{n}\right)^k$$

with some rather small $c_0 \approx 0.01$. Hence, one gets

$$\|fl([AB|UV|XY]) - [AB|UV|XY]\| \le c_0 N n^2 \left(10 + \frac{20}{n}\right)^k \tau \|[A|U|X]\|\|[B|V|Y]\| + O(\tau^2)$$

Assuming here, for simplicity, $l = n$, $n = N^{1/k}$, one easily obtains the required error estimate.

The values $k = 2, 3, ..., \log_{48} N$ correspond to the one-level, two-level, and asymptotically fastest (multilevel) versions of our algorithm, which have the error growth estimates $O(N^2), O(N^{5/3})$, and $O(N^{1.605})$ respectively. This should be compared to a rather disappointing estimates $O(N^{3.585})$ and $O(N^{4.170})$ for the Strassen and Winograd algorithms, respectively (valid for fixed size of the innermost matrix multiplications, see [19], [12]). The numerical tests given below clearly confirm this theoretical comparison of stability between the Strassen-type algorithms and the new ones.

Remark. From the formula given above one can see that the upper bound for the numerical error attains its minimum when $n = \exp\left(O(\sqrt{\log N})\right)$. Hence, the requirement for the innermost matrix multiplication sizes to be not small (see the next section) conforms well with the numerical stability of the PK3 algorithm.

4 One-level algorithms for medium-size matrices

As follows from consideration regarding the performance of modern RISC computers, it appears that when the matrix size is not too large, say $n < 10000$, it makes sense to perform only *one step* of the recursion, and then switch to the standard algorithm. Otherwise, a large number of small subproblems of matrix addition/subtraction and multiplication arises, and they cannot be processed at high Mflops rates.

Hence, to multiply two not too large $N \times N$ matrices, $N = nl$, it is enough to apply the procedure of the preceding section for $m = (n+2)/2$ with all block multiplications being $l \times l/3$ by $l/3 \times l$ ones and performed, e.g., by a properly tuned *standard* MM routine, e.g. DMR code [25], [26].

The one-level PK21 algorithm was already outlined in Section 2.3. We now consider the PK31 algorithm, where the triple disjoint product procedure is applied once, and then switch is made to the standard MM. If the nearly optimum (from the viewpoint of section 5.1) values of $n \approx 50$ are used, then $40 \le l \le 200$, which is rather advantageous for attaining a sufficiently high performance for MM of sizes 2000 to 10000 on RISC computers. Next we present an analysis showing the optimum n which minimizes the total operation count for the two-level method.

As follows from the discussion presented in Section 6.1,

$$\begin{aligned} T_{PK31}(N) &= \frac{13n^2+120n+9}{3n}N^2 + (n^3+12n^2+24n)\left(\frac{N}{n}\right)^2\left(2\frac{N}{3n}-1\right) \\ &= \left(\frac{2}{3}+\frac{8}{n}+\frac{16}{n^2}\right)N^3 + \left(\frac{10}{3}n+28-\frac{31}{n}\right)N^2 \end{aligned}$$

for $N = nl$. The minimizer of the latter expression is

$$n^* \approx \sqrt{2.4N},$$

and the corresponding operation count is given by

$$T^*_{PK31}(N) \approx \frac{2}{3}N^3 + 10.33N^{5/2} + O(N^2).$$

This minimum is attained at

$$l^* \approx \sqrt{\frac{5}{12}N},$$

which satisfies $32 \le l \le 62$ for $2000 < N < 10000$. Such bounds on l seem rather satisfactory for attaining good Mflops performance. Note also that T as the function in l is very flat to the right of l^*, so using somewhat larger l would only slightly increase the operation count while may considerably improve the Mflops rate for the standard MM routine at the inner level. Also, using larger values of l is necessary to adjust the algorithm to odd-sized and rectangular input matrices by padding them with zeros, see the next section.

The latter operation count should be compared with related to that in Section 2.3,

$$T^*_{PK2}(N) = N^3 + 4.45N^{5/2} + O(N^2).$$

The latter bound is clearly inferior for sufficiently large values of N. However, the advantage of PK21 algorithm is that it tends to have larger optimum cut-off level $l^* \approx \sqrt{\frac{2}{3}N}$, and therefore, may deliver better Mflops performance.

It should be noted that for realistic cut-off sizes l and limited values of N, say, $500 < N < 18000$, these simple procedures appear to be quite competitive even in operation count with the Strassen-type algorithms. This is demonstrated in the next section, where the operation count of the above mentioned methods is estimated for an arbitrary value of N.

4.1 The comparison of performance for Odd-Sized matrices

Consider the case where N is an **arbitrary** number. Both algorithms of the preceding section can be employed using the bordering technique. For Winograd's algorithm we find some N_+ such that $N \leq N_+ = 2^k l$, for which the operation count $T_{SW}(N_+)$ is minimum. Similarly, for the one-level algorithm we use N_+ such that $N \leq N_+ = nl$ (with n even and l an integer multiple of 3) for which $T_{PK31}(N_+)$ is minimum. To this end, the estimated total number of operations is

$$T_{SW}(2^k l) = (2l^3 + 4l^2)7^k - 5l^2 4^k$$

and

$$T_{PK31}(nl) = ((2n^2 + 24n + 48)l + 10n^2 + 84n - 93)nl^2/3.$$

Then the original matrices were augmented with $N_+ - N$ zero rows and columns, and the above described procedures applied. The results shown in Fig.1 (where we give the ratio $T(N)/N^2$ versus N) confirm our best expectations. Indeed, for *all* medium-large matrices ($1500 < N < 18000$), the one-level PK31 algorithm requires clearly smaller number of operations, provided that the cut-off size satisfies $l \geq 72$.

A similar comparison can be done between the SW algorithm and PK21 (the one-level 2-Procedure, see Section 2.3 above), for which

$$T_{PK21}(nl) = ((n^2 + 3n - 1)l + 2n^2 + 5n - 5)nl^2$$

under the same restriction $l \geq 72$. In this case, one can observe that PK21 has (on average) a better operation count for all matrix sizes in the range $500 \leq N \leq 2300$. Note that for $N > 18000$ one can switch to 2-level algorithms, e.g. PK22, see [11], or 2-level designs for 3-Procedure.

Recall that imposing a lower bound on the cut-off size l (say, near 72, or even more, as in our numerical experiments) is necessary for attaining a satisfactory Mflops rate on RISC computers.

4.2 Adjustment of fast algorithms for rectangular MM

We mainly follow the bordering techniques outlined in [22], [24]. Assume that we are multiplying $N \times K$ matrix A by $K \times M$ matrix B. The design can rely on using either Strassen-type algorithm for $n \times n$ by $n \times n$ MM with $n = 2^k$, $k \geq 1$, or a 2-Procedure related algorithm for $n \times 2n$ by $2n \times n$ MM with $n \geq 4$, or a 3-Procedure related algorithm for $n \times 3n$ by $3n \times n$ MM with $n = 2k$, $k \geq 4$. For simplicity, let us consider the case when the $n \times 2n$ by $2n \times n$ algorithm of Section 2.2. is used.

Assuming that n is considerably smaller than $\min(N, K, M)$, represent the matrix sizes as

$$N = nl_N - r_N, \quad 0 \leq r_N < n,$$

$$K = 2nl_K - r_K, \quad 0 \leq r_K < 2n,$$

$$M = nl_M - r_M, \quad 0 \leq r_M < n.$$

Then we set

$$N_+ = nl_N, \qquad K_+ = 2nl_K, \qquad M_+ = nl_M,$$

and note that

$$l_N < \frac{N}{n} + 1, \qquad l_K < \frac{K}{2n} + 1, \qquad l_M < \frac{M}{n} + 1.$$

Next we augment the matrix A by $N_+ - N$ null rows and $K_+ - K$ null columns to obtain $N_+ \times K_+$ matrix A_+, and augment the matrix B by $K_+ - K$ null rows and $M_+ - M$ null columns to obtain

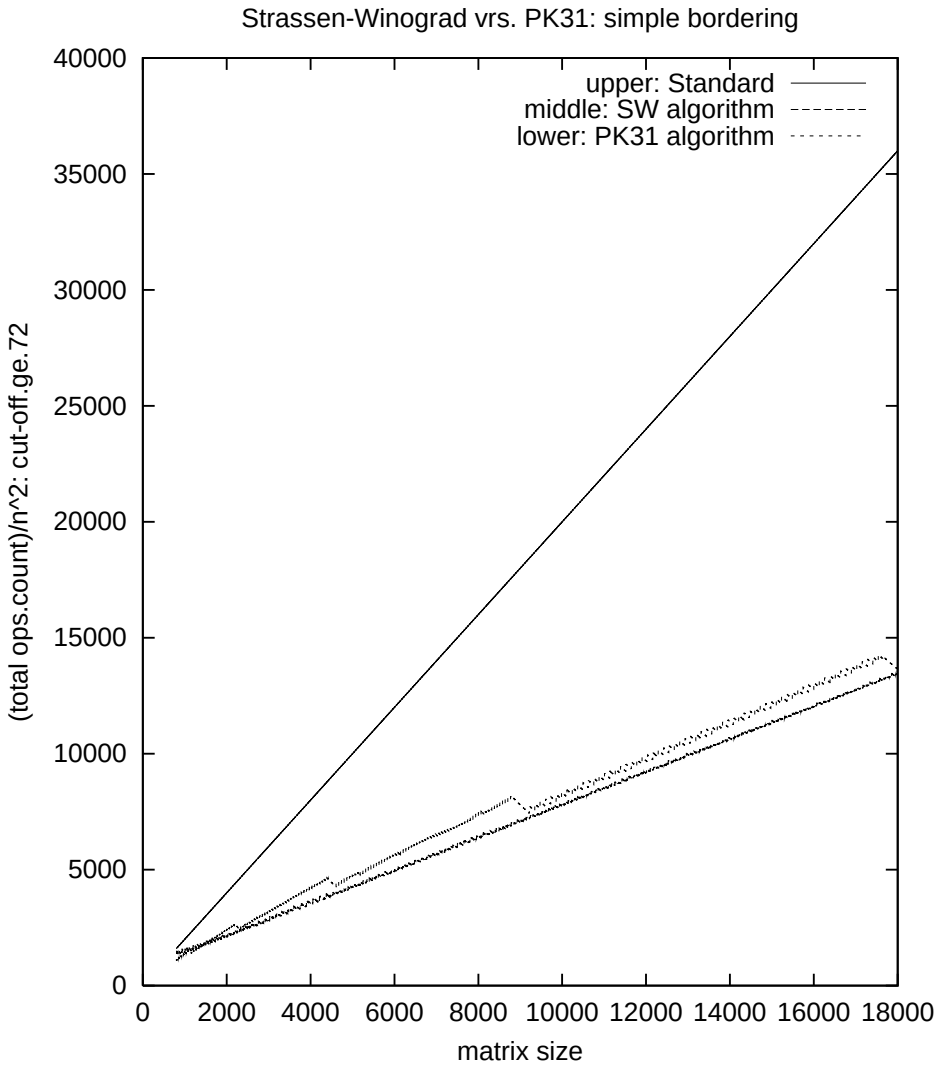

Figure 1: Standard, Strassen-Winograd, and PK31 operation counts (cut-off ≥ 72)

$K_+ \times M_+$ matrix B_+. Finally, we multiply these matrices using the algorithm of Section 2.2 to obtain $C_+ = A_+ B_+$, and return the first N rows and M columns of C_+ as the required product C.

Since all the blocks of so constructed matrix A_+ and B_+ are of the size $l_N \times l_K$ and $l_K \times l_M$, respectively, we have the following estimate (cf. Section 2.2):

$$\begin{aligned}
T_{PK21}(N,K,M;n) &= \frac{\tilde{\alpha}_1(n)}{2} l_N l_K + \frac{\tilde{\alpha}_1(n)}{2} l_K l_M + \tilde{\alpha}_1(n) l_N l_M + \tilde{\mu}(n) l_N l_M (2l_K - 1) \\
< \; & (n^3 + 3n^2 - 2n)\left(\frac{N}{n}+1\right)\left(\frac{K}{2n}+1\right) + (n^3 + 3n^2 - 2n)\left(\frac{K}{2n}+1\right)\left(\frac{M}{n}+1\right) \\
+ \; & (2n^3 + 5n^2 - 4n)\left(\frac{N}{n}+1\right)\left(\frac{M}{n}+1\right) + (n^3 + 3n^2 - n)\left(\frac{N}{n}+1\right)\left(\frac{M}{n}+1\right)\left(\frac{K}{n}+1\right) \\
= \; & \left(1 + \frac{3}{n} - \frac{1}{n^2}\right) NKM + \frac{3n^2+9n-4}{2n}(NK+KM) + \frac{3n^2+8n-5}{n} NM \\
+ \; & (4n^2 + 11n - 7)(N+M) + (2n^2 + 6n - 3)K + 5n^3 + 14n^2 - 9n \\
= \; & NKM\left(1 + \frac{3}{n} + 3n\left(\frac{1}{2M} + \frac{1}{2N} + \frac{1}{K}\right)\right) + O(NK + KM + NM).
\end{aligned}$$

Hence, if $\min(N,K,M)$ is large and n is chosen as

$$n_* \approx \sqrt{\frac{2NKM}{NK + KM + 2NM}},$$

then the operation count is almost by twice smaller than that of the standard algorithm, $2NKM - NM$:

$$T_{PK21}(N,K,M;n_*) = NKM\left(1 + 3\sqrt{\frac{2}{2M} + \frac{2}{2N} + \frac{4}{K}}\right) + O(NK + KM + NM).$$

If $N \approx K \approx M$, then we still have $n = O(N^{1/2})$ and therefore the cut-off levels are again $O(N^{1/2})$, but with somewhat larger constant, which even gives us some additional advantage of improving Mflops performance on RISC computers, see the next section.

5 Numerical results

For numerical tests we used a server installed at GC CUNY with two Pentium III XEON 733MHz processors, 1GB ECC RAM, and 50GB RAID 5 storage. The operating system is RedHat Linux 7.2; tests were run using single processor. The object code was compiled using "g77 -O3 -funroll-loops *.f" command line.

Another set of test runs was performed on a single processor of a multiprocessor high-performance SUN workstation under UNIX. In this case the codes were compiled from the command line "f77 -O4 -native -dalign -fsimple=1 *.f".

We used the matrix-matrix multiplication Fortran routine DMR [25] as the lowest-level procedure for fast matrix multiplication, as well as the benchmark code which implements the standard $O(N^3)$ algorithm. DMR is a public-domain code optimized for the IBM RS6000 architecture and based on the use of blocked and unrolled matrix-matrix multiplication. The source code of DMR can be downloaded from the Internet address "htpp://www.netlib.org/blas/dmr". We also give a comparison with the "plain" MM routine DGEMM downloaded from the same NETLIB/BLAS website.

The test problem $C = AB$ was chosen with

$$A = I + uv^T, \qquad B = I - \frac{1}{1 + v^T u} uv^T, \qquad C = I,$$

where the vectors u and v were specified by

$$u_i = \frac{1}{N+1-i}, \qquad v_i = \sqrt{i}, \qquad i = 1, \ldots, N.$$

Therefore, the computational error was measured as

$$Err = \max_{i,j} |(fl(C))_{i,j} - \delta_{i,j}|,$$

where $fl(C)$ denotes the product computed in the double precision floating point arithmetics and $\delta_{i,j}$ stands for the Kronecker's delta.

In Tables 3-5 we display (for several matrix sizes $N = 2^n 3^m$) the total operation count, CPU time in seconds, performance in megaflops, floating point error as defined above, and the memory volume in float words per N^2. The SW method and our one-level 2-Procedure and 3-Procedure based methods with the cut-off level $l = 2^p 3^q$ are denoted here SW(l), PK21(l), $l \approx 2.5\sqrt{N}$, and PK31(l), $l \approx 2\sqrt{N}$, respectively. We have not actually run SGEMM with $N = 6912$ on SUN workstation; extrapolated data are given instead.

The results show that the new algorithms are quite competitive with the SW algorithm with respect to the total operation count and, at the same time, provide a dramatic improvement in the precision of the floating point result.

An unexpected observation is that the running time of our SW routine decreases as the total operation count increases. This effect definitely suggests that local data processing (within CPU/registers/cache) is many times faster than main core memory addressing. Therefore, the elapsed time depends on the number of main memory references rather than on the arithmetic operation count (cf.[25]). This also applies to the PKt1(l) codes, $t = 2, 3$, where numerous $l \times l/t$, $l/t \times l$, and $l \times l$ matrix additions take relatively large fraction of time (running at ≈50 Mflops) as compared to $\approx 3t-2$ times fewer number of $l \times l/t$ by $l/t \times l$ matrix multiplications (running at ≥350 Mflops).

In Figs. 2, 3 and 4, some data from these tables are visualized to show the computing time versus matrix size in log-log scale. The floating-point error is presented in a similar way in Figs. 5, 6, and 7. The cut-off size was chosen $l = 72$ for Strassen-Winograd algorithm, except for the case of $N = 6912$, where $l = 54$.

In Fig.8, the ratio of computing times $T_{\rm DMR}(N)/T_{\rm PK1}(N)$ for Pentium III is shown for *all* $N = 1000, 1001, \ldots, 4000$. It is seen that this ratio approaches its limiting value 1/2 as the matrix size increases. Here we used the simple bordering approach described in Section 4.1.

It can be seen that, despite somewhat lower Mflops rates, the PK2 and, especially, PK3 methods make it possible to perform matrix multiplication up to 1.7 times faster even when compared to the one of the fastest available Fortran codes, the DMR routine. Strassen-Winograd algorithm appears to be competitive with PK21 and PK31 with respect to running time (mainly because of better Mflops performance due to a smaller percentage of matrix addition calls), but its numerical accuracy level is by several orders of magnitude worse.

6 Conclusions

In this paper, a new class of practically applicable fast matrix multiplication algorithms was described which is quite competitive with the Strassen and Winograd methods with respect to the total arithmetic costs. At the same time, new algorithms are considerably more numerically stable, take much less working storage, and have clear and flexible structure that make them rather appealing for the implementation on computers with memory hierarchy and/or parallel processing.

Table 3: MM on Pentium III server: double precision

Size	Method	Total Ops.	Time,s.	Mflops	Err	Rel.Mem.
1152	DGEMM	0.306+10	29.83	102.5	6.77-14	0.0000
1152	DMR	0.306+10	8.57	356.6	8.81-15	0.0031
1152	SW(18)	0.152+10	8.28	183.6	8.30-11	0.7531
1152	SW(36)	0.165+10	7.58	217.7	1.93-11	0.7531
1152	SW(72)	0.184+10	7.66	240.2	4.41-12	0.7531
1152	SW(144)	0.207+10	7.27	284.7	8.82-13	0.7531
1152	SW(288)	0.235+10	7.15	328.7	1.81-13	0.7531
1152	PK21(72)	0.186+10	7.21	258.6	2.27-13	0.2747
1152	PK31(72)	0.198+10	8.99	220.8	8.44-14	0.8260
2304	DGEMM	0.245+11	238.03	102.7	1.72-13	0.0000
2304	DMR	0.245+11	70.12	349.4	1.76-14	0.0008
2304	SW(18)	0.106+11	56.68	187.0	6.91-10	0.7508
2304	SW(36)	0.116+11	51.25	226.3	1.40-10	0.7508
2304	SW(72)	0.129+11	51.26	251.6	3.58-11	0.7508
2304	SW(144)	0.145+11	50.08	289.5	7.78-12	0.7508
2304	SW(288)	0.165+11	49.66	332.3	1.41-12	0.7508
2304	PK21(144)	0.147+11	54.16	271.8	3.27-13	0.2562
2304	PK31(96)	0.131+11	58.46	224.6	1.96-13	0.5281
4608	DGEMM	0.196+12	1911.80	102.3	3.58-13	0.0000
4608	DMR	0.196+12	553.79	353.9	4.60-14	0.0002
4608	SW(18)	0.746+11	386.24	193.1	4.47-09	0.7502
4608	SW(36)	0.810+11	357.49	226.6	1.10-09	0.7502
4608	SW(72)	0.902+11	361.32	249.6	2.25-10	0.7502
4608	SW(144)	0.102+12	334.53	304.9	5.17-11	0.7502
4608	SW(288)	0.115+12	364.77	315.3	1.05-11	0.7502
4608	PK21(144)	0.108+12	408.33	264.5	1.08-12	0.1265
4608	PK31(144)	0.941+11	363.38	259.0	4.72-13	0.3885

Table 4: MM on Pentium III server: single precision

Size	Method	Total Ops.	Time,s.	Mflops	Err	Rel.Mem.
1152	SGEMM	0.306+10	27.12	112.7	4.80-05	0.0000
1152	DMR	0.306+10	7.46	409.7	4.83-06	0.0031
1152	SW(18)	0.152+10	6.26	242.3	4.69-02	0.7531
1152	SW(36)	0.165+10	5.77	285.8	8.66-03	0.7531
1152	SW(72)	0.184+10	5.79	317.0	2.68-03	0.7531
1152	SW(144)	0.207+10	5.89	351.5	4.89-04	0.7531
1152	SW(288)	0.235+10	6.31	372.5	1.01-04	0.7531
1152	PK21(72)	0.186+10	5.97	312.2	1.26-04	0.2747
1152	PK31(72)	0.198+10	7.55	262.9	4.29-05	0.8260
2304	SGEMM	0.245+11	221.42	110.5	7.76-05	0.0000
2304	DMR	0.245+11	59.81	408.9	1.06-05	0.0008
2304	SW(18)	0.106+11	42.83	250.1	3.45-01	0.7508
2304	SW(36)	0.116+11	39.25	294.5	6.50-02	0.7508
2304	SW(72)	0.129+11	39.44	326.3	1.68-02	0.7508
2304	SW(144)	0.145+11	40.50	358.3	3.90-03	0.7508
2304	SW(288)	0.165+11	43.36	379.9	7.67-04	0.7508
2304	PK21(144)	0.147+11	45.15	326.0	1.45-04	0.2562
2304	PK31(96)	0.131+11	46.45	282.8	1.40-04	0.5281
4608	SGEMM	0.196+12	1741.96	112.3	1.83-04	0.0000
4608	DMR	0.196+12	472.87	413.8	2.87-05	0.0002
4608	SW(18)	0.746+11	306.16	244.6	2.72+00	0.7502
4608	SW(36)	0.810+11	281.43	287.4	5.64-01	0.7502
4608	SW(72)	0.902+11	283.74	317.7	1.26-01	0.7502
4608	SW(144)	0.102+12	292.15	348.0	2.46-02	0.7502
4608	SW(288)	0.115+12	306.75	376.2	5.15-03	0.7502
4608	PK21(144)	0.108+12	336.52	322.4	4.73-04	0.1265
4608	PK31(144)	0.941+11	308.71	304.9	2.51-04	0.3885
6912	SGEMM	0.660+12	5873.83	112.4	1.71-04	0.0000
6912	DMR	0.660+12	1607.36	410.9	8.57-05	0.0001
6912	SW(27)	0.231+12	915.67	252.5	3.05+00	0.7501
6912	SW(54)	0.268+12	844.40	317.4	6.23-01	0.7501
6912	SW(108)	0.302+12	877.34	343.8	1.42-01	0.7501
6912	SW(216)	0.342+12	933.15	366.1	2.77-02	0.7501
6912	PK21(192)	0.361+12	1075.20	336.1	8.70-04	0.1118
6912	PK31(144)	0.286+12	950.61	300.9	4.73-04	0.2560

Table 5: MM on SUN workstation: single precision

Size	Method	Total Ops.	Time,s.	Mflops	Err	Rel.Mem.
2304	SGEMM	0.245+11	455.08	53.7	7.68-05	0.0000
2304	DMR	0.245+11	112.01	218.3	4.77-05	0.0008
2304	SW(36)	0.116+11	86.71	133.3	3.61-01	0.7508
2304	SW(72)	0.129+11	79.11	162.7	1.41-01	0.7508
2304	SW(144)	0.145+11	75.71	191.7	1.97-02	0.7508
2304	SW(216)	0.163+11	77.49	212.6	3.45-03	0.7508
2304	PK21(144)	0.147+11	76.45	192.5	2.14-04	0.2562
2304	PK31(96)	0.131+11	98.58	133.3	1.41-04	0.5281
4608	SGEMM	0.196+12	3647.83	53.6	1.72-04	0.0000
4608	DMR	0.196+12	987.23	198.2	3.81-05	0.0002
4608	SW(36)	0.810+11	639.30	126.5	2.90+00	0.7502
4608	SW(72)	0.902+11	575.50	156.6	1.08+00	0.7502
4608	SW(144)	0.102+12	557.50	182.4	1.49-01	0.7502
4608	SW(216)	0.115+12	548.28	210.5	3.61-02	0.7502
4608	PK21(144)	0.108+12	570.62	190.1	6.63-04	0.1265
4608	PK31(144)	0.941+11	597.62	157.5	2.52-04	0.3886
6912	SGEMM	0.660+12	12332.00	53.5	3.85-04	0.0000
6912	DMR	0.660+12	3143.21	210.1	6.87-05	0.0001
6912	SW(27)	0.231+12	2280.88	101.4	1.87+01	0.7501
6912	SW(54)	0.268+12	1774.05	151.1	6.47+00	0.7501
6912	SW(108)	0.302+12	1646.10	183.2	1.40+00	0.7501
6912	SW(216)	0.342+12	1720.38	198.6	2.46-01	0.7501
6912	PK21(192)	0.361+12	1757.01	205.7	1.11-03	0.1118
6912	PK31(144)	0.286+12	1814.96	157.6	4.73-04	0.2560

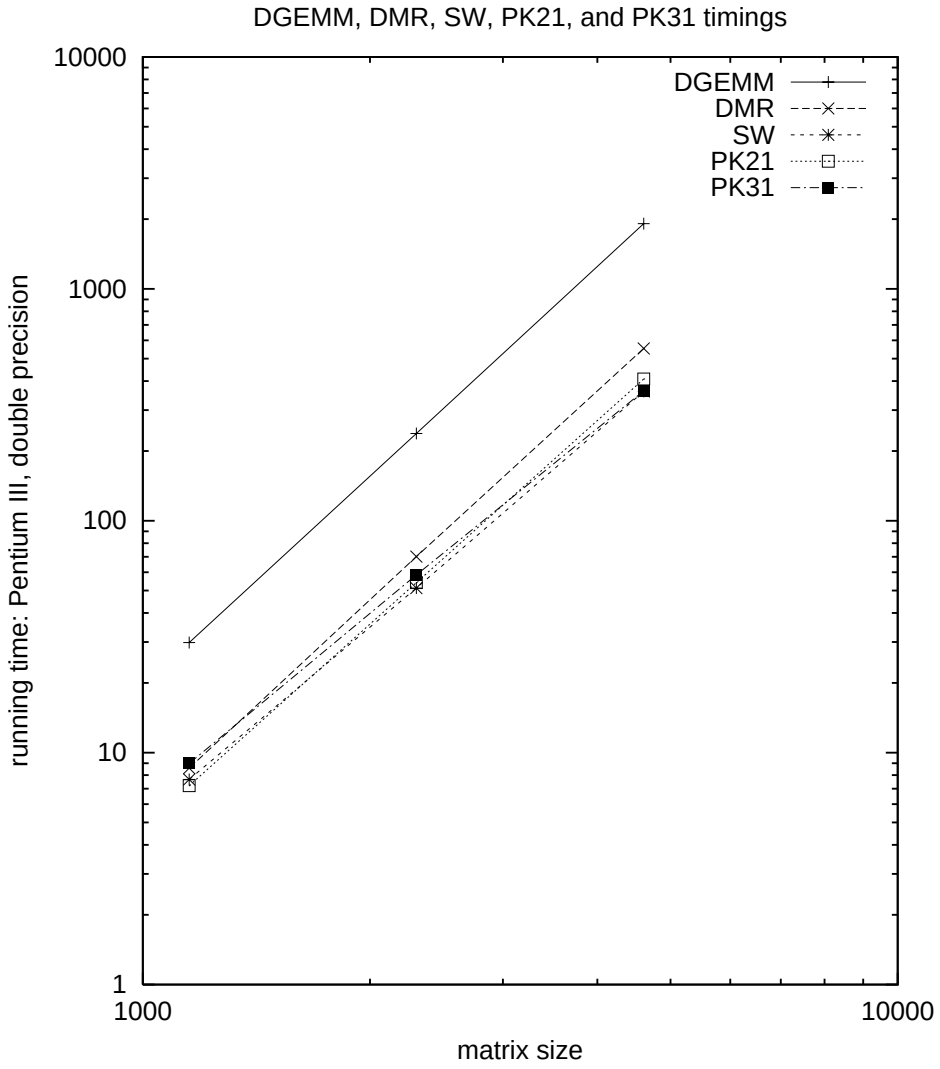

Figure 2: Running time of MM algorithms on Pentium III server: double precision

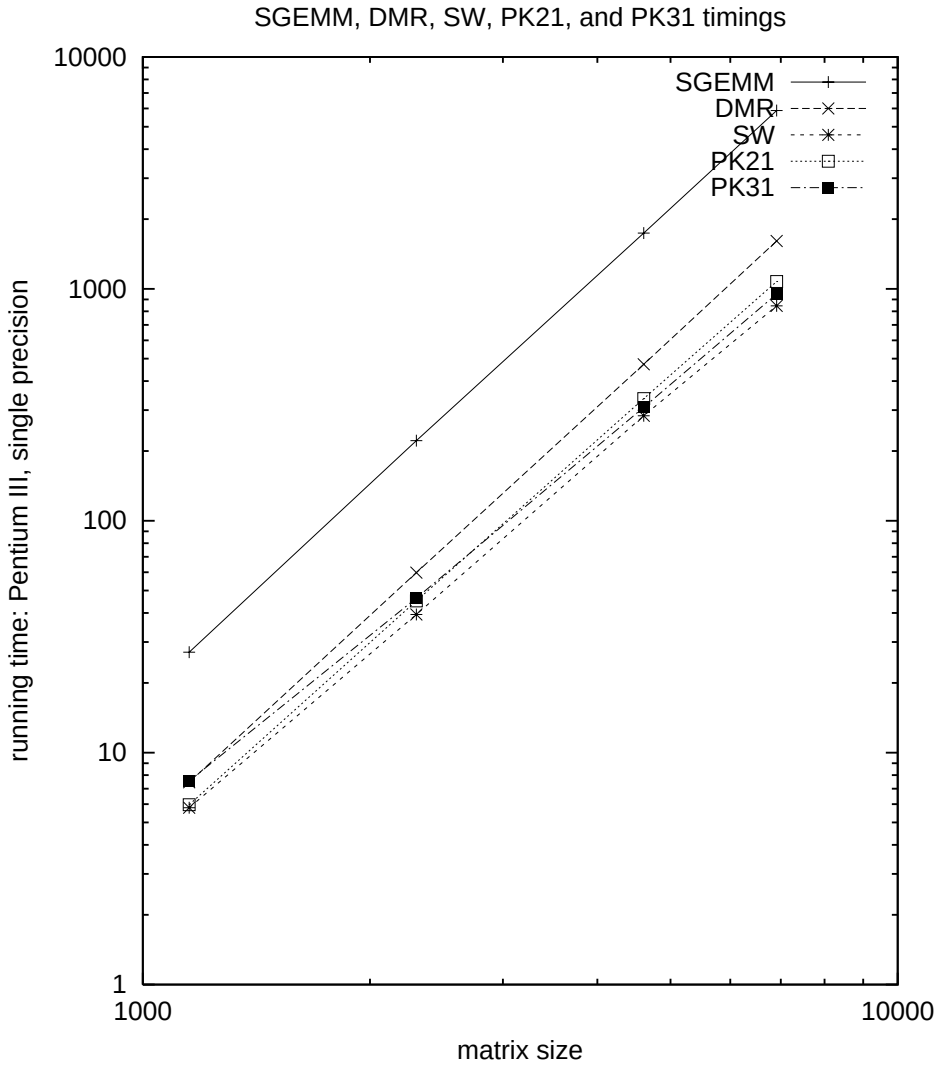

Figure 3: Running time of MM algorithms on Pentium III server: single precision

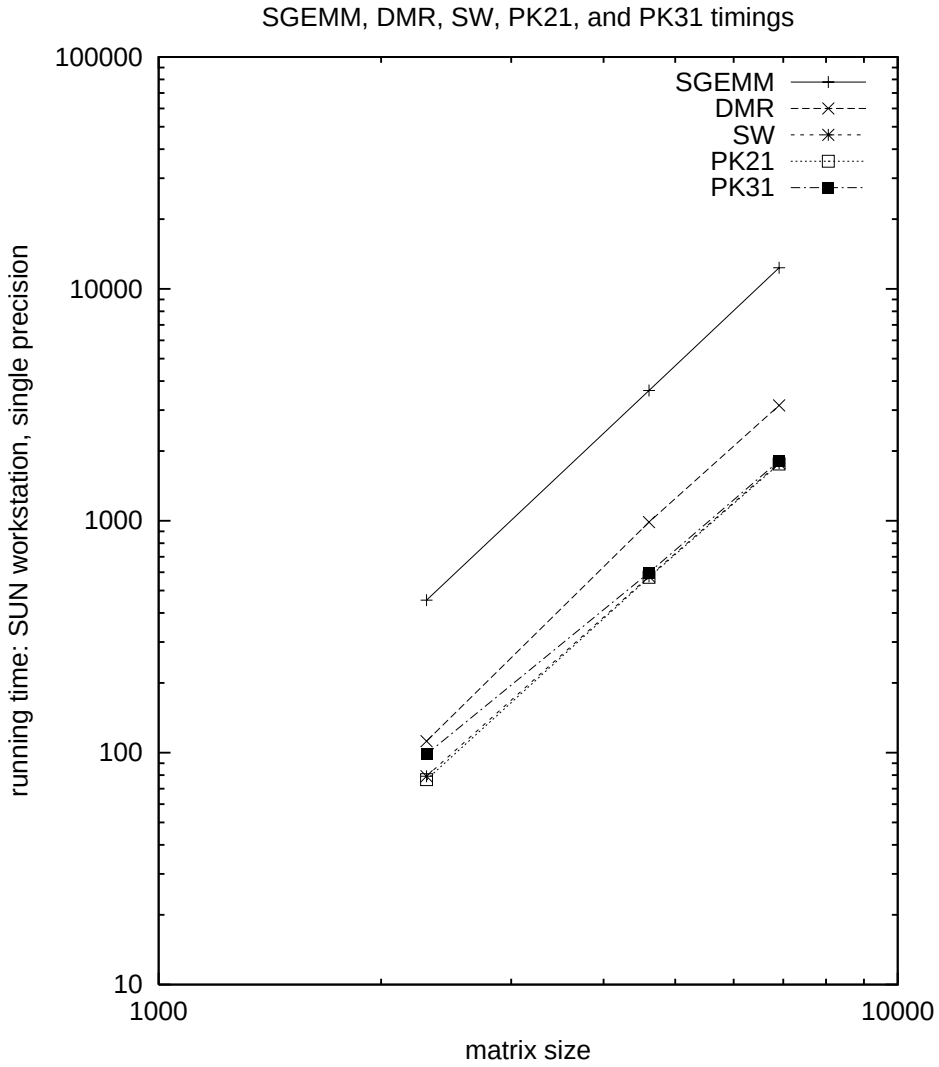

Figure 4: Running time of MM algorithms on SUN workstation: single precision

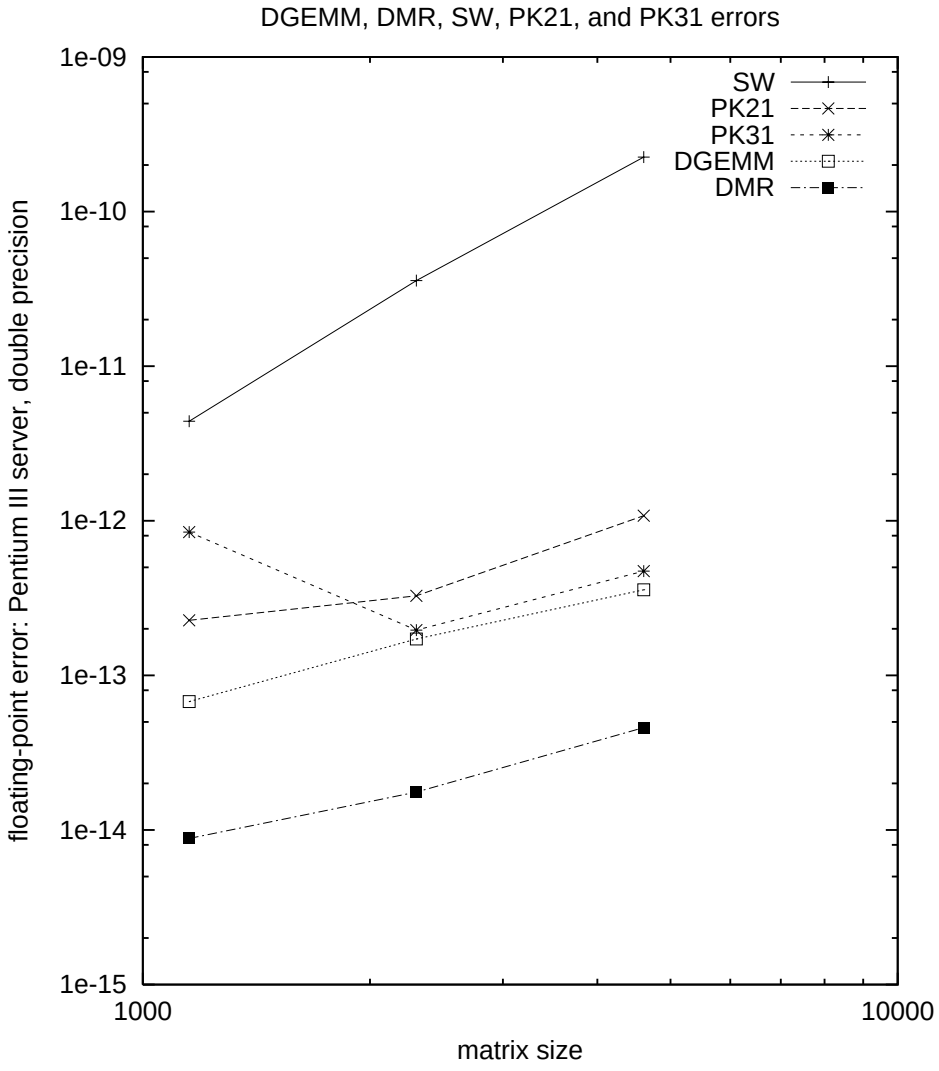

Figure 5: Floating-point error of MM algorithms on Pentium III server: double precision

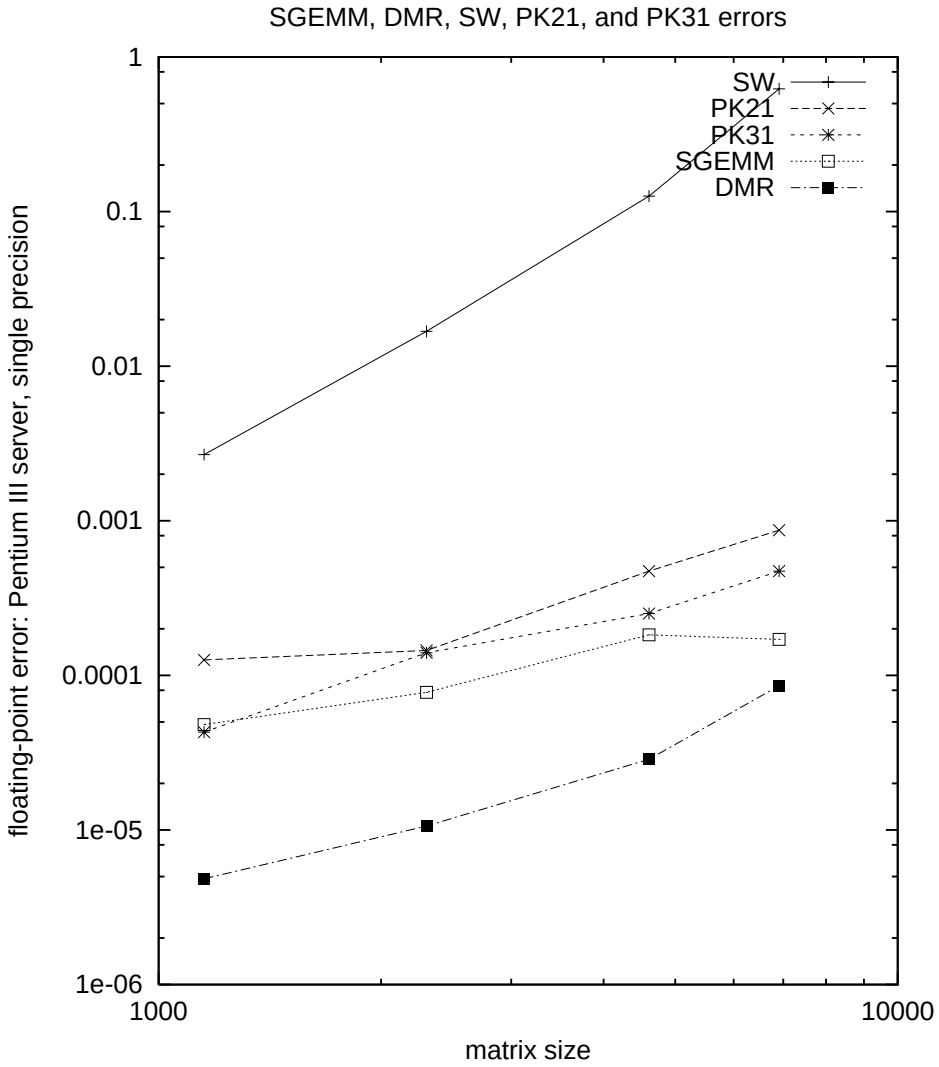

Figure 6: Floating-point error of MM algorithms on Pentium III server: single precision

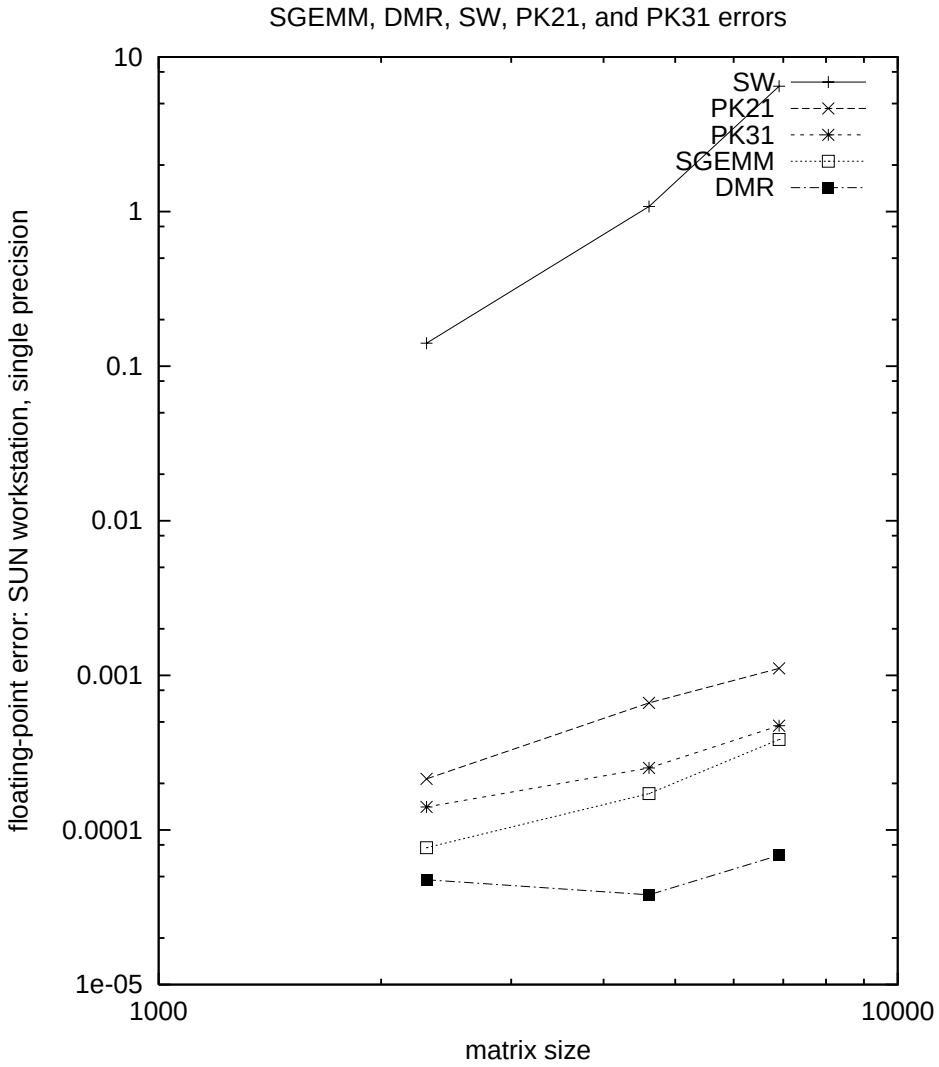

Figure 7: Floating-point error of MM algorithms on SUN workstation: single precision

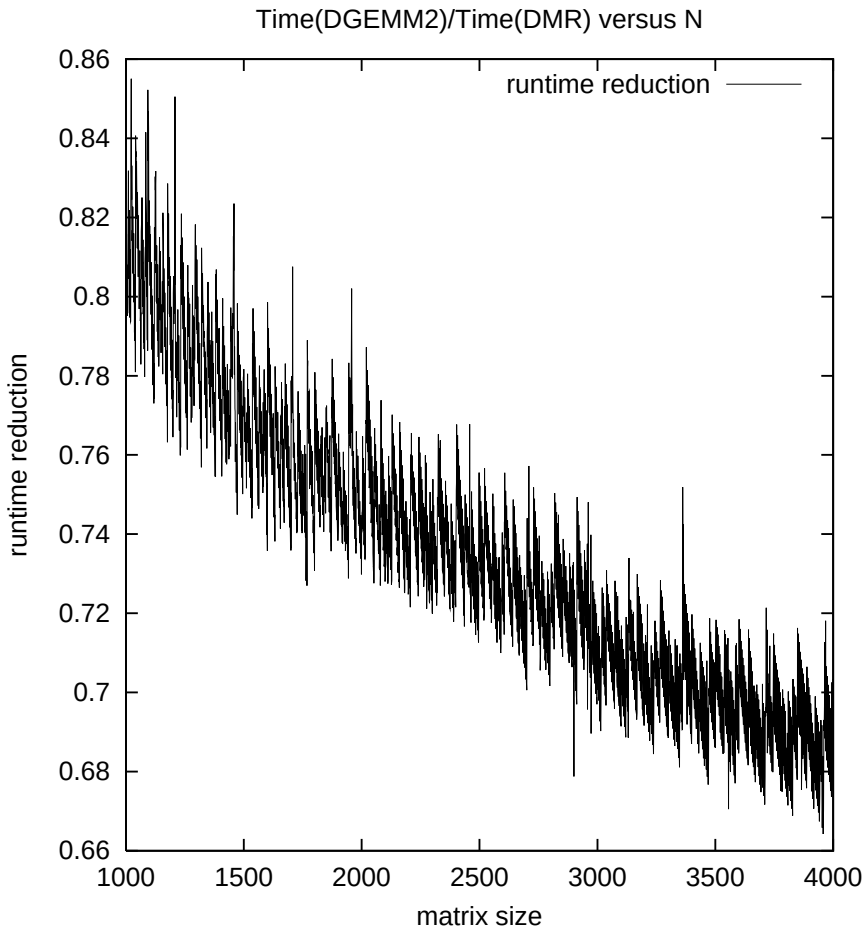

Figure 8: Running time reduction with PK1 compared to DMR on Pentim III: double precision

Acknowledgements

This work has been supported by the NSF grant CCR-9732206. The author would like to acknowledge helpful comments of Prof. V.Y.Pan on the subject of the paper and the assistance of A.Koukinova in performing experiments shown in Fig.8.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Massachusets (1974).

[2] U. Zwick, All pairs shortest paths in weighted directed graphs - exact and inexact algorithms, Proc. 39th Annual Symp. on Foundations of Computer Science (FOCS'98), 310–319, IEEE Computer Soc. Press, Los Alamos, California, 1998.

[3] I. Akutsu, S. Miyano, and S. Kuhara, Algorithms for identifying Boolean networks and related biological networks based on matrix multiplication and fingerprint function, J. Comput. Biol., **7** (2000) 331–343

[4] T. Biedl, B. Brejova, E. D. Demaine, A. M. Hamel, and T. Vinar, Optimal arrangement of leaves in the tree representing hierarchical clustering of gene expressin data, Technical Report 2001-14, Dept. of Comput. Science, University of Waterloo, 12 pp.

[5] E. Cohen and D. Lewis, Approximating Matrix Multiplication for Pattern Recognition Tasks, J. of Algorithms, (special issue of selected papers from SODA'97), **30** (1999) 211-252.

[6] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, Massachusets (1979).

[7] L. G. Valiant, General context free recognition in less than cubic time, J. Computer and System Sciences, **10** (1975) 308–315

[8] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progressions, J. Symbolic Comput., **9**, no.3 (1990) 251–280

[9] X. Huang and V. Y. Pan, Fast rectangular matrix multiplication and applications, J. of Complexity, **14** (1998) 257–299

[10] J. Laderman, V. Y. Pan, and X.-H. Sha, On practical algorithms for accelerated matrix multiplication, Linear Algebra Appls. 162-164 (1992) 557–588

[11] I. Kaporin, Practical algorithm for faster matrix multiplication, Numer. Linear Algebra Appls., **6** (1999) 687–700

[12] D. Bini and G. Lotti, Stability of fast algorithms for matrix multiplication, Numer. Math., **36** (1980) 63–72

[13] V. Strassen, Gaussian elimination is not optimal, Numer. Math., **13** (1969) 354–356

[14] V. Y. Pan, Computation schemes for a product of matrices and for the inverse matrix (Russian), Uspekhi Mat. Nauk., **27** no.5 (1972) 249–250

[15] V. Y. Pan, New combinations of methods for the acceleration of matrix multiplications, Computers and Mathematics with Appls., **7** (1981) 73–125

[16] V. Y. Pan, How can we speed up matrix multiplication, SIAM Review, **26** no.3 (1984) 393–415

[17] I. Kaporin, A compact form of the aggregation-cancellation algorithm for three disjoint matrix products (Paper in progress, CCRAS Moscow – CG CUNY New York, 2001) 11 pp.

[18] C. C. Douglas, M. Heroux, G. Slishman, and R. M. Smith, GEMMW: A portable Level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm, J. Comput. Physics, **110** (1994) 1–10

[19] N. Higham, Exploiting fast matrix multiplication within the Level 3 BLAS, ACM Trans. Math. Soft., **16**, no.4 (1990) 352–368

[20] J. W. Demmel and N. J. Higham, Stability of Block Algorithms with Fast Level-3 BLAS, ACM Trans. Math. Soft., **18**, no.3 (1992) 274–291

[21] N. Higham, Accuracy and Stability of Numerical Algorithms, SIAM Publications, Philadelphia (1996)

[22] S. Huss-Lederman, E. M. Jacobson, J. R. Johnson, A. Tsao, and T. Turnbull, A portable implementation of Strassen's algorithm (DGEFMM User's Guide), Comp. Sci. Dept., Univ. of Wisconsin-Madison. Madison, WI, November 12, 1996.

[23] V. P. Pauca, X. Sun, S. Chatterjee, and A. R. Lebeck, Architecture-efficient Strassen's matrix multiplication: a case study of divide-and-conquer algorithms, Proc. ILAS Symp., June 1997, *or* Tech. Report CS-1998-06, Dept. Comput. Sci. Duke University, Durham, May 1998, 16 pp.

[24] M. Thottlehodi, S. Chatterjee, and A. R. Lebeck, Tuning Strassen's matrix multiplication for Memory Efficiency, Proc. Supercomputing'98, Nov.1998.

[25] Jack J. Dongarra, Peter Mayes, and Giuseppe Radicati di Brozolo, The IBM RISC System/6000 and Linear Algebra Operations, University of Tennessee Computer Science Tech Report: CS-90-122 (1990)

[26] Basic Linear Algebra Subprograms, http://www.netlib.org/blas/dmr

[27] D. H. Bailey, Extra speed matrix multiplication on the Cray-2, SIAM J. Sci. Stat. Comput. **9** no.3 (1988) 603–607