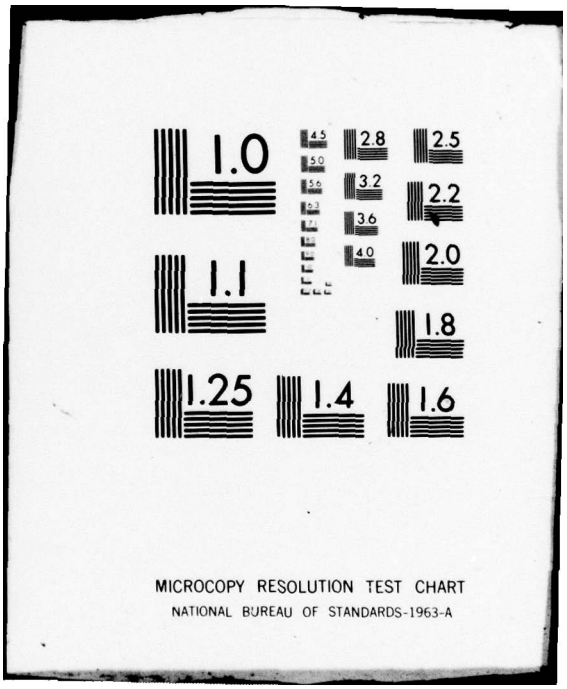
AD-A080 074 MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB F/G 12/1
FAST PARALLEL REALIZATION OF MATRIX MULTIPLICATION.(U)
NOV 79 E V KRISHNAMURTHY, R KLETTE AFOSR-77-3271
UNCLASSIFIED TR-834 AFOSR-TR-80-0079 NL

| OF |
ADA
080074

END
DATE
FILMED
2-80
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

LEVEL II

COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(9) Interim rept. (11) LEVEL II

(14) TR-834
(15) AFOSR-77-3271

(11) November 1979

(6) FAST PARALLEL REALIZATION OF MATRIX MULTIPLICATION.

(12) 31

(10) E. V. Krishnamurthy* and Reinhard Klette+
Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

(16) 2304 (17) A2

ABSTRACT

Fast parallel matrix multiplication algorithms in SIMD (Single-Instruction-Multiple data) and MIMD (Multiple-Instruction-Multiple-data) modes are described for implementation in a parallel-binary matrix processing system with facilities for bit-wise parallel Boolean operations and power-of-two shifts on Boolean matrices. A comparative study of these algorithms is made on the basis of their relative time-complexities, when conventional binary and prime-modulus arithmetic are used for forming the matrix product.

(18) AFOSR (19) TR-80-0079

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper.

*Permanent address: Indian Institute of Science, Bangalore 560012, India.
+Permanent address: Sektion Mathematik, Friedrich-Schiller-Universität, DDR-69 Jena, Universitätshochhaus 17.OG, German Democratic Republic.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 80-0079	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FAST PARALLEL REALIZATION OF MATRIX MULTIPLICATION		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) E. V. Krishnamurthy and Reinhard Klette		8. CONTRACT OR GRANT NUMBER(s) AFOSR 77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Maryland, Computer Vision Laboratory, Computer Science Center College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force OFFice of Scientific Research/NM Bolling AFB, Washington, DC 20332		12. REPORT DATE November 1979
		13. NUMBER OF PAGES 30
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel comutation
Matrix multiplication
SIMD Machines
MIMD Machines

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Fast parallel matrix multiplication algorithms in SIMD (Single -Instruction-Multiple-data) and MIMD (Multiple-Instruction-Multiple-data) modes are described for implementation in a parallel-binary matrix processing system with facilities for bit-wise parallel Boolean operation and power-of-two shifts on Boolean matrices. A comparative study of these algorithms is made on the basis of their relative time-complexities, when conventional binary and prime-modulus arithmetic are used for forming the matrix product.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

1. Introduction

The object of this paper is to describe fast parallel algorithms for multiplying matrices of arbitrary sizes having integer elements of arbitrary magnitude. These algorithms are based on the earlier work of Klette [3] but attempts are made to improve them by the use of modular arithmetic for computation. The use of modular arithmetic in effect improves the algorithms in [3] when the matrix product has elements which are very high precision numbers of the order of 2^e (or e bits). It is shown that if two matrices of size (mxm) are multiplied, and we choose r prime numbers k_i each of precision e_i bits, the time taken is as follows:

Single-instruction-multiple-data algorithm

$$O(re^2 + m(e \log m + \sum_{i=1}^{r} e_i^2)),$$

Multiple-instruction-multiple-data algorithm

$$O(\log r \log e + (\log em)(\max_i \log e_i)).$$ [1]

Since matrix multiplication is an important operation for realizing matrix transformations and inversions of matrices [7,12] it is believed that the present approach will have practical utility.

The organization of this paper is as follows: Section 2 describes the basic principle of the parallel matrix multiplication

[1] Logarithms are always taken to the base 2.

algorithms; Section 3 explains the design features of a parallel processor for bit-wise computation; the time complexity of the various possible schemes are discussed in Section 4; the use of fast multiplication schemes for further speedup is considered in Section 5; finally, the speedup that can result by the use of modular arithmetic is described in Section 6.

2. Principle and definitions

We denote the two matrices $A = (a_{ij})$ and $B = (b_{ij})$ of size (kxm) and (mxn) respectively. Our aim is to compute $A \cdot B$ of size kxn with elements

$$\sum_{p=1}^{m} a_{ip} b_{pj} \quad \text{for } i = 1,2,\ldots,k,\ j = 1,2,\ldots,n\ .$$

By 'parallel' we mean that the operations transform the operand matrix as if all the matrix elements were processed simultaneously. For the practical realization of such a scheme we can think of fiber optics [10] or CCD techniques [8].

We now describe the principal idea of our algorithm:

Let C[p] denote a (kxn) matrix in which all the columns are identical with the p-th column of the matrix A; let R[p] denote a (kxn) matrix in which all the rows are identical with the p-th row of the matrix B. For two matrices C and R of the same size kxn, let

$$C*R = (c_{ij} r_{ij}) \quad i = 1,2,\ldots,k,\ j = 1,2,\ldots,n$$

be the <u>modified product</u> of these two matrices C and R. It immediately follows that

<u>Lemma</u>: $A \cdot B = \sum_{p=1}^{m} C[p]*R[p]$.

We illustrate this lemma by an example in Figure 1.

This lemma can be used for parallel computation of $A \cdot B$, either in SIMD (single-instruction-multiple-data) mode or MIMD

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix}$$

$$A \cdot B = C[1]*R[1] + C[2]*R[2] + C[3]*R[3]$$

$$= \begin{bmatrix} a_{11}\cdot b_{11} & a_{11}\cdot b_{12} \\ a_{21}\cdot b_{11} & a_{21}\cdot b_{12} \\ a_{31}\cdot b_{11} & a_{31}\cdot b_{12} \\ a_{41}\cdot b_{11} & a_{41}\cdot b_{12} \end{bmatrix} + \begin{bmatrix} a_{12}\cdot b_{21} & a_{12}\cdot b_{22} \\ a_{22}\cdot b_{21} & a_{22}\cdot b_{22} \\ a_{32}\cdot b_{21} & a_{32}\cdot b_{22} \\ a_{42}\cdot b_{21} & a_{42}\cdot b_{22} \end{bmatrix} + \begin{bmatrix} a_{13}\cdot b_{31} & a_{13}\cdot b_{32} \\ a_{23}\cdot b_{31} & a_{23}\cdot b_{32} \\ a_{33}\cdot b_{31} & a_{33}\cdot b_{32} \\ a_{43}\cdot b_{31} & a_{43}\cdot b_{32} \end{bmatrix}$$

Figure 1. Illustration of Lemma 1

(multiple-instruction-multiple-data) mode, using Flynn's [1] notation. In writing these algorithms we use an Algol-like notation for brevity.

(1) SIMD mode.

Algorithm PS

```
    begin matrices A,B,C,R,A·B;
          integer m,p;
          initialize A·B identical 0;
          for p = 1 step 1 until m do
              C = C[p] using A, and R = R[p] using B;
              C = C*R;
              A·B = A·B+C
          od;
end
```

Remark: This algorithm involves m steps of computation, each with one construction of C[p] and R[p], one modified matrix product, and one matrix addition. If we regard these as primitive operations, we obtain A·B in SIMD mode in O(m) operations. This algorithm was described in Klette [3].

(2) MIMD mode.

For realization in this mode, it is assumed that m is an integral power of 2; where it is not we assume log m is replaced by its higher integral part in the algorithm described below.

Algorithm PM

```
begin matrices A,B,C[1],...,C[m],R[1],...,R[m];
    integer m,p,i;
    for p = 1 step 1 until m do in parallel
        compute C[p],R[p] using A, B, respectively;
        C[p] = C[p]*R[p];
    od;
    for i = 1 step 1 until log m do
        for p = 1 step 2^i until m-1 do in parallel
            C[p] = C[p] + C[p+2^(i-1)];
        od;
    od;
end
```

Remark: In this algorithm we first perform in one step the construction of $C[p]$ and $R[p]$ and one modified matrix product for $p = 1,2,\ldots,m$ in parallel. Then we compute in log m steps of parallel matrix addition the desired sum $\sum_{p=1}^{m} C[p] = A \cdot B$ in the matrix field $C[1]$. Assuming that these are primitive instructions for a certain MIMD computer, $A \cdot B$ can be computed in $O(\log m)$ steps.

In the next section we will explain how to devise a parallel processor for bit-wise computation in order to realize the operations used in the algorithms PS and PM.

3. Parallel processor for bit-wise computation

We assume that the elements of the matrices A and B are integers from the set $0,1,2,\ldots,2^e-1$ for $e \geq 1$. (Note that this does not restrict the choice of negative numbers as elements of A and B, as complement notation can be used.) Then each matrix A (or B) can be encoded in a bijective manner through a sequence of e Boolean matrices $A_{e-1}, A_{e-2}, \ldots, A_0$ of the same dimensions as A, where

$$A_{e-1}(i,j)\ A_{e-2}(i,j)\ \ldots\ A_0(i,j)$$

is the positional binary representation of the elements, viz., $A(i,j) = a_{ij}\ldots$ In other words, the matrices $A_{e-1}, A_{e-2}, \ldots, A_0$ represent the bit planes of the matrix A. For illustration, an example is provided below:

Let $A = \begin{bmatrix} 5 & 19 \\ 12 & 7 \end{bmatrix}$; then

$$A_4A_3A_2A_1A_0 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

It is clear that using such a positional-additive decomposition, the matrix operations (such as add, multiply) can be realized using Boolean operations and shifts on sequences of such Boolean matrices.

(1) Addition in SIMD mode.

Let $A_{e-1}, A_{e-2}, \ldots, A_0$ and $B_{e-1}, B_{e-2}, \ldots, B_0$ be binary matrix sequences of same dimensions. Then the following algorithm [3] performs the matrix addition A+B using Boolean

operations on the binary matrices (bit-wise parallel) as basic steps in SIMD mode.

Algorithm AS

begin binary matrices $A_0,\ldots,A_{e-1},B_0,\ldots,B_{e-1},C_0,\ldots,C_{e-1},C_e$, D, CARRY;

integer e,i;

$C_0 = A_0 \oplus B_0$; CARRY = $A_0 \wedge B_0$; i=1;

while i<e do

$D = A_i \oplus B_i$; C_i = D⊕CARRY;

CARRY = $(A_i \wedge B_i) \vee$(CARRY∧D);

i = i+1;

od;

C_e = CARRY;

end

Remark: The sequence of the outputs $C_e,C_{e-1},\ldots,C_0$ represents the result A+B. Using bitwise parallel Boolean operations on binary matrices this algorithm in SIMD mode has time complexity O(e).

Note that the basic idea of this matrix addition algorithm is similar to adding two e-bit numbers in sequential mode. Figure 2 illustrates this algorithm.

(2) Addition in MIMD mode.

In [9] a SIMD algorithm has been described for adding e-bit numbers in parallel in O(log e) steps, using parallel-bitwise Boolean operations and shifts in vector registers. Using this

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = A_1 A_0$$

$$B = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = B_1 B_0$$

$$C_0 = A_0 \oplus B_0 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad \text{CARRY} = A_0 \wedge B_0 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$$D = A_1 \oplus B_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad C_1 = D \oplus \text{CARRY} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

$$C_2 = (A_1 \wedge B_1) \vee (\text{CARRY} \wedge D) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$

$$A+B = \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = C_2 C_1 C_0$$

Figure 2. Illustration of Algorithm AS

SIMD algorithm we now give an algorithm in MIMD mode for adding binary matrices $A_{e-1},\ldots,A_0$ and $B_{e-1},\ldots,B_0$ in O(log e) time. This algorithm is most efficient when e is a power of 2; where it is not we assume log e is replaced by its higher integral part in the algorithm described below.

ALGORITHM AM

```
begin binary matrices A_0,...,A_{e-1},B_0,...,B_{e-1},C_0,...,C_{e-1},C_e,D_0,...,D_{e-1};
    integer e,i,j;
    for i = 0 step 1 until e-1 do in parallel
        C_i = A_i ∧ B_i; D_i = A_i ∨ B_i;
    od;
    for i = 0 step 1 until log e-1 do
        for j = 2^i step 1 until e-1 do in parallel
            C_j = C_j ∨ (C_{j-2^i} ∧ D_j);
            D_j = D_j ∧ D_{j-2^i};
        od;
    od;
    C_e = C_{e-1};
    for i = 1 step 1 until e-1 do in parallel
        C_i = C_{i-1} ⊕ A_i ⊕ B_i;
    od;
    C_0 = A_0 ⊕ B_0;
end
```

Remark: The sequence of the outputs $C_e, C_{e-1},\ldots,C_0$ represents the result A+B. This algorithm in MIMD mode has O(log e) time

complexity. In Figure 3 we illustrate this algorithm.

(3) Generation of C[p] and R[p] in SIMD mode.

The processes of generating C[p], p = 1,2,...,m and R[p], p = 1,2,...,m are important operations in our algorithm. Let $A = A_{e-1}, A_{e-2} \ldots A_0$ be the matrix of size mxm where m is a power of 2. The following SIMD algorithm computes $C_{e-1}, C_{e-2} \ldots C_0$ corresponding to C[1] (see [3]). The algorithm uses power-of-two shifts of Boolean matrices; such shifts are denoted by $\leftarrow 2^i$ (left), $\rightarrow 2^i$ (right), $\uparrow 2^i$ (up), $\downarrow 2^i$ (down) where the shift distance is 2^i. While performing such two-dimensional shifts, the vacated columns or rows are filled with zeros, and the shifted-out columns or rows are discarded.

Algorithm CS

```
begin binary matrices A_0,...,A_{e-1},C_0,...,C_{e-1},M;
      integer e,i,j;
      in M the leftmost column is identically 1, otherwise M is 0;
      i = 0;
      while i<e do
          C_i = A_i ∧ M; j=0;
          while j<log m do
              C_i = C_i ∨ (C_i →2^j); j = j+1;
          od;
          i = i+1;
      od
end
```

$$A = \begin{bmatrix} 11 & 9 \\ 4 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = A_3A_2A_1A_0$$

$$B = \begin{bmatrix} 4 & 13 \\ 7 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} = B_3B_2B_1B_0$$

step 1:

$$C = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = C_3C_2C_1C_0$$

$$D = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = D_3D_2D_1D_0$$

step 2 (i=0):

$$C = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

step 3 (i=1):

$$C = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

step 4:

$$C_4 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

step 5:

$$C_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}, \quad C_2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad C_3 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

step 6:

$$C_0 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

result:

$$A+B = \begin{bmatrix} 15 & 22 \\ 11 & 13 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = C_4C_3C_2C_1C_0$$

Figure 3. Illustration of algorithm AM

Remark: Using bitwise parallel Boolean operations and power-of-two shifts on binary matrices this algorithm has time complexity O(e log m). The principle of this algorithm is easy to understand, demonstrating the utility of power-of-two shifts using a divide-and-conquer approach. The construction of matrices $C[2],\ldots,C[m],R[1],\ldots,R[m]$ can be performed within the same time, using an analogous algorithm.

(4) Generation of $C[p]$ and $R[p]$ in MIMD mode.

The construction of $C[1]$ can be done within time O(log m) using the following algorithm in MIMD mode.

Algorithm CM

Change i-loop in algorithm CS into a parallel instruction.

Remark: The generation of the matrices $C[1],\ldots,C[m],R[1],\ldots,R[m]$ can be done within time O(log m) in MIMD mode.

(5) Modified matrix product in SIMD mode.

In what follows we shall assume that bit-wise parallel Boolean operations and power-of-two shifts on Boolean matrices are basic operations in our computational model. We will call this model a Parallel Binary Matrix Processing System (PBS). (For a more detailed description of this model see [3,4,5].) In this model the size of the matrix registers is restricted to the original size of the input matrices. In Pratt et al. [9], however, a similar model for computation is described (with restrictions on one-dimensional registers) with unrestricted registers. It is our view that PBS offers a more realistic model

for computation, since programs in the PBS model take linear space while programs in Pratt/Stockmeyer's model [9] use exponential space for problems such as matrix multiplication.

The essential advantage of PBS for computing the modified product is that well-known techniques used for computations with binary numbers can be translated directly to the case of binary-matrix computations. In fact, using the usual $O(e^2)$ algorithm for multiplying e-bit numbers, one can realize the modified matrix product of two matrices $A_{e-1},A_{e-2},\ldots,A_0$ and $B_{e-1},B_{e-2},\ldots,B_0$ in time complexity $O(e^2)$ in SIMD mode.

Algorithm MS

```
begin binary matrices A0,...,Ae-1,B0,...,Be-1,C0,...,C2e-1,E0,...
                      ...,Ee-1,D0,...,De-1,De;
    integer e,i,j;
    C0 = A0^B0;
    for i = 0 step 1 until e-2 do Di=Ai+1^B0; od;
    De-1 identical 0;
    for j=0 step 1 until e-1 do
        for i=0 step 1 until e-1 do Ei=Ai^Bj; od;
        addition of De-1De-2...D0 and Ee-1Ee-2...E0 using
        algorithm AS, resulting in DeDe-1...D0;
        Cj=D0;
        for i=0 step 1 until e-1 do Di=Di+1; od;
    od;
    for i=0 step 1 until e-1 do Ce+i=Di ; od;
end
```

Remark: The sequence $C_{2e-1}, C_{2e-2} \ldots C_0$ represents the modified product A*B. This program runs within time $O(e^2)$.

(6) Modified matrix product in MIMD mode.

In Pratt et al. [9], a SIMD algorithm can be found for the parallel realization of the multiplication of two e-bit numbers in O(log e) steps using parallel bitwise Boolean operations and shifts on vector registers. Using this same basic idea we can give an algorithm in MIMD mode for computing the modified multiplication of matrices $A_{e-1}A_{e-2} \ldots A_0$ and $B_{e-1}B_{e-2} \ldots B_0$ running in time O(log e). This translation can be done in a manner similar to the method used in algorithm AM: One parallel operation on vector registers can be translated into one parallel instruction on sequences of Boolean matrices. Unfortunately, the resulting MIMD algorithm for PBS requires a very large number of matrix registers ($O(m^2)$).

Algorithm MM

Translation of the SIMD algorithm for multiplication of binary numbers in Pratt et al. [9] into MIMD program for PBS. With this algorithm all operations used in the algorithms PS and PM are implemented on PBS, either in SIMD mode or in MIMD mode.

4. Time complexity

In Section 2, we described two basic algorithms for computing the matrix product, one in SIMD mode and one in MIMD mode. In Section 3 we described two ways to implement these algorithms on PBS in SIMD and MIMD modes. Thus altogether we have at least four different ways to compute the matrix products in parallel; we will now summarize the time complexity for these four cases assuming that A and B are mxm matrices (with m a power of two) and that the elements in A and B are from the set $\{0,1,2,\ldots,2^e-1\}$.

Case i: The general SIMD algorithm with SIMD implementation on PBS has time complexity O(m e (log m + e)).

Case ii: The general SIMD algorithm with MIMD implementation on PBS has time complexity O(m(log m+log e)).

Case iii: The general MIMD algorithm with SIMD implementation on PBS has time complexity O(e(log m+e)).

Case iv: The general MIMD algorithm with MIMD implementation on PBS has time complexity O(log m log e).

In our opinion, Case i is presently a realistic mode for technical implementation (see [10]). In what follows we refer to Case i as the single-instruction-multiple-data algorithm. In Section 6 we will use these two approaches and discuss ways to improve the speed by using residue or modular arithmetic and other procedures; see Knuth [6] and Krishnamurthy et al. [7].

Since in practical numerical analysis of matrix operations, the precision of the result is of considerable importance, it is necessary to examine whether the time complexity $O(e^2)$ required for SIMD implementation (Cases i and iii) and $O(\log e)$ for MIMD implementation (Cases ii and iv) can be reduced, assuming m to be a constant.

5. Use of fast multiplication schemes

It is well known (Knuth [6]) that the order-e^2 method is not the quickest way to multiply e-bit numbers. For example Karatsuba's [2] method involves only $e^{\log 3} \approx e^{1.59}$ steps using the conventional sequential implementation. This method leads to a SIMD algorithm for the modified matrix product on PBS running within the same time $O(e^{1.59})$. We will call this algorithm MSK.

Algorithm MSK

Translation of Karatsuba's method into a SIMD algorithm for PBS for the modified product A*B.

Remarks: For a detailed description of this and other methods for multiplying e-bit numbers, see Knuth [6]. In our view algorithm MSK is suitable for practical realization and so we will use this for our purposes. For Cases (i) and (iii) described in Section 4, this algorithm leads to the following improvements:

Case i: Time Complexity $O(m\ e \log m + m\ e^{1.59})$.

Case iii: Time Complexity $O(e \log m + e^{1.59})$.

In the next section we will discuss the application of residue arithmetic and consider the time complexities for the SIMD and MIMD algorithms.

6. Modular arithmetic procedures

Another significant and practical approach to speeding up the processing time is to use modular or residue or congruence arithmetic. For discussions of modular arithmetic reference is made to Knuth [6]; for applications to matrix inversion see Krishnamurthy et al. [7].

For this purpose let us assume that $A=(a_{ij})$, $B=(b_{ij})$ are two mxm matrices with elements in $\{0,1,2,\ldots,k-1\}$ and that the product $A\cdot B$ has elements in the same set. (Note that this differs from our earlier discussions where we develop products in conventional arithmetic.) Let $k_1,k_2,\ldots,k_r$ be a set of pairwise relatively prime integers, with $k = \sum_{i=1}^{r} k_i$. It is well known (Chinese remainder theorem) that there is exactly one integer a which satisfies the conditions

$$0 \leq a \leq k-1 \text{ and } a \equiv a_j \pmod{k_j}$$

for $1 \leq j \leq r$, for any given sequence $(a_1,a_2,\ldots,a_r)$ of integers. Using this result the given matrices A and B can be encoded and $A\cdot B$ can be computed in modular representation for each k_j, and then combined using the Chinese remainder theorem. Figure 4 gives an example of multiplying two matrices A and B in this manner.

As illustrated, this procedure involves conversion of A and B to modular representation modulo each k_j, and computation of $A(\text{mod } k_j)\cdot B(\text{mod } k_j) \text{ mod } k_j$, for $1 \leq j \leq r$. These results (namely each element in the product) should then be combined using the Chinese remainder theorem or other related procedures; see

$$A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 4 & 5 \\ 2 & 4 \end{bmatrix}$$

Let $k = 30 = 2 \cdot 3 \cdot 5$; $k_1 = 2$, $k_2 = 3$, $k_3 = 5$.

Step 1: Matrices A and B are encoded modulo k_1, k_2, k_3.

$$A(\text{mod } 2) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \| \quad A(\text{mod } 3) = \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} \quad \| \quad A(\text{mod } 5) = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

$$B(\text{mod } 2) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \| \quad B(\text{mod } 3) = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \quad \| \quad B(\text{mod } 5) = \begin{bmatrix} 4 & 0 \\ 2 & 4 \end{bmatrix}$$

Step 2: Product of $A(\text{mod } k_i)$ and $B(\text{mod } k_i)$ modulo k_i.

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \quad \| \quad \begin{bmatrix} 2 & 1 \\ 0 & 0 \end{bmatrix} \quad \| \quad \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}$$

Step 3: Product of the result in Step 2 with k/k_i for $1 \leq i \leq 3$.

$$\begin{bmatrix} 0 & 0 \\ 0 & 15 \end{bmatrix} \quad \| \quad \begin{bmatrix} 20 & 10 \\ 0 & 0 \end{bmatrix} \quad \| \quad \begin{bmatrix} 24 & 12 \\ 12 & 6 \end{bmatrix}$$

Step 4: Addition of the results in Step 3 modulo k.

$$\begin{bmatrix} 14 & 22 \\ 12 & 21 \end{bmatrix} = A \cdot B$$

Figure 4. Modular approach to matrix product.

Szabo et al. [11], Knuth [6], Krishnamurthy et al. [7], Young et al. [12]. Here we will use a straightforward multiplicative-additive approach, in which each element of the product $A(\text{mod } k_j) \cdot B(\text{mod } k_j) \text{ mod } k_j$ is multiplied by $(k/k_j)^{\phi(k_j)}$ (where ϕ denotes Euler's totient function) for $1 \leq j \leq r$ and summed modulo k, as illustrated in Figure 4.

An important aspect in using modular arithmetic is the choice of k_j's suitable for practical implementation. It seems convenient from a practical design viewpoint that each k_j be a prime of the form $(2^{e_j}-1)$, since this would then correspond to bitwise computation on PBS. (As an example $e_j = 2,3,5,7$).

Since we are interested in implementing the modular approach on PBS, let us describe the general algorithms for implementation in SIMD and MIMD modes.

Let $A_i = A(\text{mod } k_i)$, $B_i = B(\text{mod } k_i)$ for $1 \leq i \leq r$.

(1) SIMD mode.

Algorithm PMS

```
begin matrices A,B,A*,B*,C,R,D, A·B;
      integer m,r,k,k_1,...,k_r,i,j;
      initialize A·B identical 0;
      for i = 1 step 1 until r do
          production of A*=A_i and B*=B_i using A and B
          respectively;
          for j = 1 step 1 until m do
              initialize D identical 0;
```

```
                C = C[j] using A*, and R = R[j] using B*,
                C = C*R mod k_i;
                D = D+C mod k_i;
            od;
            D = k/k_i · D,
            A·B =  A·B+D mod k;
        od;
end
```

Remark: This algorithm has time complexity O(rm), regarding the operations used as primitive operations.

(2) MIMD mode.

For this purpose we assume that r is a power of 2; where it is not we assume log r is replaced by its higher integral part in the algorithm described below.

Algorithm PMM

```
begin matrices A,B,A_1,...,A_r,B_1,...,B_r,C_1[1],...,C_r[m],
               R_1[1],...,R_r[m];
    integer m,r,k,k_1,...,k_r,i,j,h;
    for i = 1 step 1 until r do in parallel
        production of A_i and B_i using A and B respectively;
        for j=1 step 1 until m do in parallel
            compute C_i[j],R_i[j] using A_i,B_i respectively;
            C_i[j]=C_i[j]*R_i[j] mod k_i;
        od;
```

```
        for j=1 step 1 until log m do
            for h=1 step 2^j until m-1 do in parallel
                C_i[h]=C_i[h]+C_i[h+2^(j-1)] mod k_i;
            od;
        od;
        C_i[1]=k/k_i · C_i[1];
    od;
    for i=1 step 1 until log r do
        for j=1 step 2^i until r-1 do in parallel
            C_j[1]=C_j[1]+C_(j+2^(i-1))[1] mod k;
        od;
    od;
end
```

Remark: This algorithm has time complexity O(log m + log r), regarding the operations used as primitive operations. The output $A \cdot B$ is computed in matrix register $C_1[1]$.

Implementation in PBS

We first consider the addition and the modified product modulo k_h, $1 \le h \le r$. According to the well-known techniques for modular arithmetic (Knuth [6]), for matrices F and G with elements f_{ij}, g_{ij} respectively, each with e_h-bit length (with bit positions $e_h-1, e_h-2, \ldots, 0$, the sum $F+G$ modulo $k_h = 2^{e_h}-1$ is easily obtained thus:

$$f_{ij}+g_{ij} \pmod{k_h} = \begin{cases} f_{ij}+g_{ij} & \text{if } f_{ij}+g_{ij} < 2^{e_h} \\ ((f_{ij}+g_{ij}) \bmod 2^{e_h}+1, & \text{otherwise.} \end{cases}$$

This operation is easy to perform on PBS: after the usual addition operation, the most significant bit plane, corresponding to bit position e_h, is added to the bit planes e_h-1, $e_h-2,\ldots,0$. In other words, the addition modulo k_h needs twice the time of addition of two e_h-bit matrices, i.e. $O(e_h)$ in SIMD mode if we use algorithm AS, and $O(\log e_h)$ in MIMD mode if we use algorithm AM.

For obtaining the elements of the modified product $F*G \pmod{k_h}$, the rule is:

$$f_{ij} \cdot g_{ij} \pmod{k_h} = (f_{ij} \cdot g_{ij} \bmod 2^{e_h}) + \lfloor f_{ij} \cdot g_{ij} / 2^{e_h} \rfloor \pmod{k_h}.$$

To do this calculation, we perform the modified product of two e_h-bit matrices F and G; then we perform an addition modulo k_h of the first e_h+1 bit planes and the last e_h bit planes. Using the algorithms MS and AS we can perform this on PBS in time $O(e_h^2)$ in SIMD mode. Using algorithms MM and AM, the modified product can be computed in $O(\log e_h)$ steps in MIMD mode.

Now consider the construction of the matrices A_i ($A \bmod k_i$) and B_i ($B \bmod k_i$). For this purpose, we can group together blocks of e_i bit planes of a given matrix A. Let these blocks of bit planes be denoted by the matrices $A^1, A^2, \ldots, A^t$. Then we perform the addition

$$A^1 + A^2 + \ldots + A^t \bmod k_i \quad (k_i = 2^{e_i} - 1)$$

(see Knuth [6]). For this, in SIMD mode we need $O(e_i(t-1))$ $= O(e_i \cdot \frac{\log k}{e_i}) = O(e)$ steps on PBS if we assume $k \approx 2^e$, and in MIMD mode we need $O(\log e_i \log t) = O(\log e_i \log \frac{e}{e_i}) = O(\log e_i \log e - (\log e_i)^2)$ steps on PBS.

Finally we consider the operation of matrix addition modulo k. In the algorithms PMS and PMM we have the operations

$$A \cdot B = A \cdot B + D \bmod k, \text{ and}$$

$$C_j[1] = C_j[1] + C_{j+2^{i-1}}[1] \bmod k.$$

To explain this, at each step of the operation, we first perform the (usual) addition and then we subtract from each element in parallel the value k; if the result is negative we retain the element value, otherwise we retain the subtracted result as the new value of the element. This is because of the fact that each element mod k_i when multiplied by k/k_i can be in the range $0,1,\ldots,k-1$ and so when the elements are added modulo k the result can lie in the range $0,1,\ldots,2k-2$. For implementation on PBS, note that in the resulting matrix after the subtraction, an overflow appears in the most significant bit plane in all places where the value after the addition was greater than k, and vice versa. We use this most significant bit plane as a mask for the decision between the original value and the subtracted value, for all log k bit planes. Altogether, the matrix addition modulo k used in the algorithms PMS and PMM can be performed on PBS within time $O(\log k) = O(e)$ in SIMD mode, and within time $O(\log e)$ in MIMD mode.

We will now summarize the time complexity for the SIMD and MIMD algorithms using modular arithmetic. For the algorithm PMS in SIMD implementation we have the time

$$O(re^2 + m(e \log m + \sum_{i=1}^{r} e_i^2))$$

For the algorithm PMM in MIMD implementation we have the time

$$O(\log r \log e + (\log em)(\max_i \log e_i))$$

A comparison of the time needed for the SIMD algorithms here and in Section 4 shows that modular arithmetic is to be preferred when e is large (high precision) and m is greater than r. The comparison of the needed time for the MIMD algorithms here and in Section 5 also indicates that modular arithmetic is to be preferred when m is greater than $\max_i re_i$.

7. Concluding remarks

We have presented several fast parallel matrix multiplication algorithms in this paper. Some of these algorithms can be speeded up further when the computations involve certain kinds of structured matrices. Such structured matrices, as we know, arise in practical problems involving various kinds of transformations needed in matrix computations, e.g. elementary, orthogonal transformations. This will be a fruitful area for further study.

References

[1] Flynn, M.J., Some computer organizations and their effectiveness, IEEE Trans. Comp. C-21, pp. 948-960, 1972.

[2] Karatsuba, A., Doklady Akademia Nauk SSSR, 145, pp. 293-295, 1962.

[3] Klette, R., Fast matrix multiplication by Boolean RAM in linear storage, Lecture Notes in Computer Science (Springer Verlag) 64, pp. 308-314, 1978.

[4] Klette, R., A parallel computer for digital image processing, EIK 15, pp. 237-263, 1979.

[5] Klette, R., Parallel operations on binary images, University of Maryland, Computer Science Center, TR-822, November 1979.

[6] Knuth, D. E., The Art of Computer Programming Vol. 2, Addison-Wesley, Reading, Mass., 1971.

[7] Krishnamurthy, E. V., Rao, T.M., and Subramanian, K., Residue arithmetic algorithms for computing g-inverses of matrices, SIAM J. Num. Anal. 13, pp. 155-171, 1976.

[8] Milgram, D. L., Rosenfeld, A., Willett, T., and Tisdale, G., Algorithms and hardware technology for image recognition, Final Report to U.S. Army Night Vision Laboratory, Fort Belvoir, VA, March 1978.

[9] Pratt, V. R., and Stockmeyer, L. J., A characterization of the power of vector machines, J. Comp. Sys. Sci. 12, pp. 198-227, 1976.

[10] Schaefer, D. H., and Strong, J. P., TSE computers, Goddard Space Flight Center Report X-943-75-14, January 1975.

[11] Szabo, S. and Tanaka, R., Residue arithmetic and its Applications to Computer Technology, McGraw-Hill, New York, 1967.

[12] Young, D. M. and Gregory, R. T., A Survey of Numerical Mathematics, Vol. 2, Addison-Wesley, Reading, Mass., 1973.