Matrix Multiplication On The OTIS-Mesh Optoelectronic Computer*

Chih-fang Wang†
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901
cfw@cs.siu.edu

Sartaj Sahni
Department of Computer and
Information Science and Engineering
University of Florida
Gainesville, FL 32611
sahni@cise.ufl.edu

Abstract

We develop algorithms to multiply two vectors, a vector and a matrix, and two matrices on an OTIS-Mesh optoelectronic computer. Two mappings, group row and group submesh [25], of a matrix onto an OTIS-Mesh are considered and the relative merits of each compared. We show that our algorithms to multiply a column and row vector use an optimal number of data moves for both the group row and group submesh mappings; our algorithm to multiply a row vector and a column vector is optimal for the group row mapping; and our algorithm to multiply a matrix by a column vector is optimal for the group row mapping.

Keywords: matrix multiplication, OTIS-Mesh, optical interconnect, optoelectronic computer, optimal algorithm

1 Introduction

It is well known that when communication distances exceed a few millimeters, optical interconnects provide speed (bandwidth) and power advantages over electronic interconnects [7, 15]. Therefore, in the construction of very large multiprocessor computers it is prudent to interconnect physically close processors using electronic interconnect and to use optical interconnect for pairs of processors that are distant. We shall assume that physically close processors are in the same physical package (chip, wafer, board) and processors that are not physically close are in different packages. As a result, electronic interconnects are used for intra package communications while optical interconnect is used for inter package communication.

*This work was supported, in part, by the Army Research Office under grant DAA H04-95-1-0111 and in part, by the National Science Foundation under grant CCR-9912395.

†This work was done when Dr. Wang was at the University of Florida and revised when he was at Southern Illinois University.

Various combinations of interconnection networks for intra package (*i.e.*, electronic) communications and inter package (*i.e.*, optical communications) have been proposed. In OTIS computers [11, 18, 35], optical interconnects are realized via a free space optical interconnect system known as the optical transpose interconnection system (OTIS). In an OTIS-Mesh parallel computer, we have N^2 processors organized as N groups of N processors each [35, 25]. The processors in each group form a $\sqrt{N} \times \sqrt{N}$ mesh that employs electronic interconnect. Inter group interconnects are realized using optics. Let (i, j) denote processor j of group i. In the OTIS interconnect system, processor (i, j) is connected via optics to processor (j, i) (*i.e.*, processor i of group j). A 16 processor OTIS-Mesh computer is shown in Figure 1. The small square boxes denote processors. The large square boxes enclose groups of processors. The pair i, j of numbers inside a small square box gives the group (i) and processor (j) indices. Groups are laid out as a two-dimensional array and the groups are numbered in row-major fashion. The pair (a, b) above a group box gives the row (a) and column (b) in which the group lies. Arrows connect pairs of processors that are connected by either an electronic (intra group) or optical (inter group) link. For purposes of this paper the essential differences between electronic and optical links are *(a)* optical links have much larger bandwidth than do electronic links; and *(b)* transfer times including latency are different on optical and electronic links. In our analysis, we count communication along electronic and optical interconnects separately. However, we use the simplifying assumption that any constant amount of data can be communicated over an optical link during an optical communication step while only a unit amount of data can be communicated over an electronic link during an electronic communication step. In this paper, we assume that the processor mesh that represents any group of processors is a SIMD mesh. Therefore, in any given time step, data can be moved in only one of the four mesh dimensions: up, down, left, or right. Extensions to MIMD meshes are straightforward and thus omitted.

Since the value of any architecture lies in our ability to use that architecture to effectively solve problems that are of interest, much attention has been focussed on the development of algorithms for OTIS-based computers. Algorithms for OTIS-Mesh computers have been studied by Zane *et al.* [35], Sahni and Wang [25, 32, 33, 31], Rajasekeran and Sahni [24], and Osterloh [23]. Algorithms for OTIS-Hypercubes (in these each group is a hypercube rather than a mesh) have been developed by Sahni and Wang [26]. Zane *et al.* [35] have shown that an OTIS-Mesh can simulate a $\sqrt{N} \times \sqrt{N} \times \sqrt{N} \times \sqrt{N}$ four-dimensional mesh computer as well as an N^2 processor OTIS-Hypercube employing at most 2 electronic and 1 OTIS (*i.e.*, optical communication) move per move of the 4D-

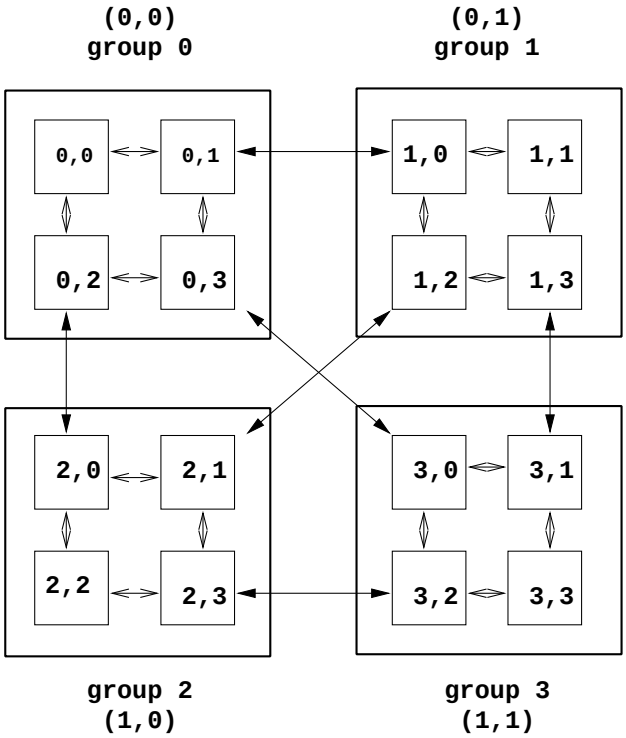

Figure 1: Example of OTIS-Mesh with 16 processors

mesh or hypercube. Sahni and Wang [25] have developed routing algorithms for the OTIS-Mesh. Algorithms for fundamental operations such as data sum, concentrate, rank, sort, broadcast, and shift have been developed by Wang and Sahni [31]. An improved sorting algorithm for the OTIS-Mesh has been developed by Osterloh [23], and ransomized algorithms for sorting, selection, and routing on the OTIS-Mesh have been developed by Rajasekeran and Sahni [24]. Wang and Sahni have proposed OTIS-Mesh algorithms for several image processing problems [33], and an early version of their work on matrix multiplication algorithms for the OTIS-Mesh appears in [32].

The development of matrix multiplication algorithms has been the focus of much research, and matrix multiplication algorithms have been developed for a wide variety of parallel and distributed computers. We cite only some of the more recent developments. Gupta [10] has developed a matrix multiplication algorithm for the EREW Model; McColl and Tiskin [19] have done this for the BSP (Bulk Synchronous Parallel) model; Li and Pan [17] consider matrix multiplication on a linear array with a reconfigurable pipelined bus system; Chen *et al.*[2] focus on systolic arrays; Kaagstroem and Raennar [13] consider a mesh; Gupta and Sadayappan [9], Sanches and Song [27], Lee [16], and Nelson [21] develop matrix multiplication algorithms on a hypercube; Middendorf *et al.*[20] have developed a sparse matrix multiplication algorithm for a reconfigurable mesh; Keqin *et al.*[14] have

an implementation for optical buses; and Oliver [22] has developed a matrix multiplication algorithm for DNA computer. Gupta and Kumar [8], Wu [34], and van de Geijn and Watts [30] discuss the scalability of various matrix multiplication algorithms; Tsay and Yuan [29] consider a cylindrical array and a two-layered mesh array; Choi [3], Chou *et al.*[5] and Choi *et al.*[4] have developed matrix multiplication algorithms for distributed-memory concurrent computers; Johnsson [12] considers the minimization of communication time for matrix multiplication; and Tasic *et al.*[28] compare some parallel matrix multiplication algorithms.

In this paper, we develop algorithms to multiply vectors of size kN and matrices of size $kN \times kN$ on an N^2 processor OTIS-Mesh. These algorithms are developed for both of the matrix to OTIS-Mesh mapping schemes considered by Sahni and Wang [25] – group row-major mapping (GRM) and group submatrix mapping (GSM). We begin, in Section 2, by describing the GRM and GSM schemes and making observations about the complexity of performing the matrix add and transpose operations. In Section 3, we develop the algorithms for various versions of vector and matrix multiplication. In this section, we show also that our algorithms to multiply a column and row vector use an optimal number of data moves for both the group row and group submesh mappings; our algorithm to multiply a row vector and a column vector is optimal for the group row mapping; and our algorithm to multiply a matrix by a column vector is optimal for the group row mapping.

2 Mapping Matrices Onto An OTIS-Mesh

Sahni and Wang [25] have considered two ways – GRM and GSM – to map an $N \times N$ matrix onto an N^2 processor OTIS-Mesh. In the group row mapping (*GRM*), the N processors in each group of an OTIS-Mesh are ordered in snake-like row-major fashion. Similarly, the two-dimensional layout of groups is ordered in snake-like row-major fashion. Row i of the matrix is mapped onto the processors in group i of the OTIS-Mesh, one element per processor. Figure 2(a) shows the GRM mapping of a 4×4 matrix onto the processors of a 16 processor OTIS-Mesh. The pair i, j in each processor square is the row and column index of the matrix element mapped to that processor.

In the group submatrix mapping (*GSM*), each group of the OTIS-Mesh represents a $\sqrt{N} \times \sqrt{N}$ submatrix of the $N \times N$ matrix. The processors in a group are indexed by the row and column indexes that determine a group's position in the two-dimensional layout of the N groups. Thus the four-dimensional index (g_x, g_y, p_x, p_y) refers to the processor in position (p_x, p_y) of group (g_x, g_y). Matrix element (i, j) is mapped to processor (i_f, j_f, i_m, j_m) where $i_f = \lfloor i/\sqrt{N} \rfloor$, $i_m = i \bmod \sqrt{N}$, $j_f = \lfloor j/\sqrt{N} \rfloor$, and $j_m = j \bmod \sqrt{N}$. Figure 2(b) shows the GSM mapping of a 4×4 matrix onto

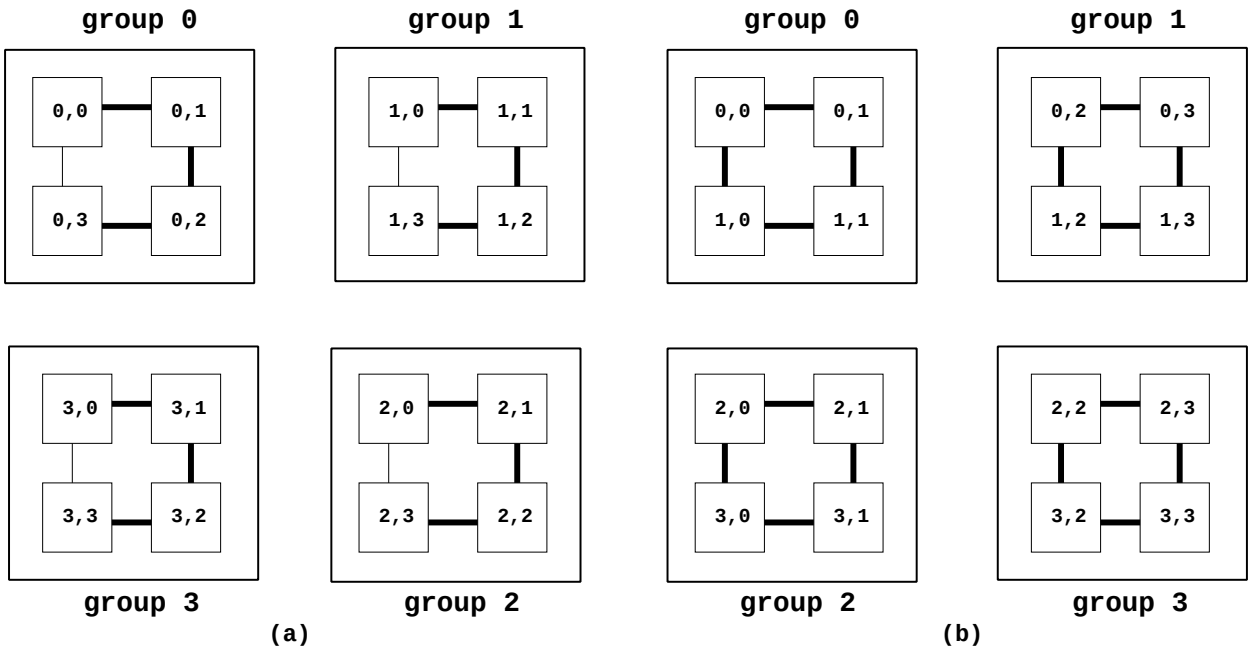

Figure 2: Mapping a $\mathbf{4 \times 4}$ matrix onto a 16 processor OTIS-Mesh: (a) GRM; (b) GSM

a 16 processor OTIS-Mesh.

The GRM and GSM mappings of a row or column vector are obtained from the corresponding mapping of an $N \times N$ matrix by extracting the sub-mapping corresponding to row zero or column zero of the matrix. The GRM and GSM mappings of a $kN \times kN$ matrix are obtained by partitioning the $kN \times kN$ matrix into $N \times N$ blocks of size $k \times k$ each. The $N \times N$ block matrix is then mapped onto the N^2 processor OTIS-Mesh, one block per processor, using the standard GRM and GSM schemes described above. $1 \times kN$ and $kN \times 1$ vectors are mapped by using the sub-mapping corresponding to row zero or column zero of the $kN \times kN$ matrix mapping.

It is easy to see that regardless of which mapping is used, matrix as well as vector addition and subtraction requires no interprocessor communication. Two $kN \times kN$ matrices can be added or subtracted in $O(k^2)$ time and two vectors of size kN can be added or subtracted in $O(k)$ time.

Algorithms for the matrix transpose operation were developed in [25]. A $kN \times kN$ matrix can be transposed using a single OTIS move and no electronic moves when the GRM mapping is used. When the GSM mapping is used, the transpose requires $8k^2(\sqrt{N}-1)$ electronic and 2 OTIS moves. In either case, $O(k^2)$ intraprocessor moves are needed to transpose the $k \times k$ block stored in a processor.

Step 1: Perform an OTIS move on B.

Step 2: Broadcast the A and B data in each group to all processors of the group.

Step 3: Perform an OTIS move on B.

Step 4: Each processor multiplies its A and B data.

Figure 3: GRM Column $\times$ Row Multiplication

3 Multiplication Algorithm

3.1 Column Vector $\times$ Row Vector

3.1.1 GRM

First consider the GRM mapping. When an $N \times 1$ column vector A and a $1 \times N$ row vector B are multiplied, the result is an $N \times N$ matrix. This is to be stored in the OTIS-Mesh using the GRM mapping.

Initially, the element A_i in row i of A is in the 0th processor of group i (*i.e.*, processor $(i, 0)$) and the jth element B_j of B is in processor $(0, j)$. Following the multiplication, the (i, j) element A_iB_j of the product matrix is to be in processor (i, j). The four step algorithm given in Figure 3 performs the multiplication.

Following Step 1, B_j is in processor $(j, 0)$, $0 \leq j < N$; and following the broadcast of Step 2, processor $(i, *)$ has A_i ($*$ denotes all permissible indexes; in this case indexes are in the range $[0, N)$) and processor $(j, *)$ has B_j. After the OTIS move of Step 3, processor (i, j) has A_i and B_j. Consequently, following the multiplication of Step 4, processor (i, j) has the (i, j)th entry of the result matrix. Therefore, the algorithm correctly multiplies the vectors A and B.

For the complexity analysis, we see that 2 OTIS moves are made in Steps 1 and 3 together. Step 2 can be done using $2\sqrt{N}$ electronic moves by first sending A and B data initially in processor 0 of a group down column zero of that group. Since only one piece of data can be moved at a time along an electronic link, this column broadcast of the A and B data can be done in $\sqrt{N}$ moves if we pipeline the data movement down column zero (*i.e.*, the B data trails the A data by one column processor). Next the A and B data in each processor of column 0 are broadcast along rows using a similar pipelining. This requires another $\sqrt{N}$ electronic moves. The complexity of our GRM column $\times$ row algorithm is therefore $2\sqrt{N}$ electronic and 2 OTIS moves.

Theorem 1 *Our column $\times$ row algorithm is an optimal algorithm.*

Proof To see this, first note that all the B values are initially in group 0, and all need to get to group 1 (say) either in the form of A_1B_j or simply B_j. The only way data can move from one group to another is via an OTIS move, and a single OTIS move can only move a constant number of the B_j's accumulated into a single processor of group 0 into a single processor of group 1. Therefore, at least 2 OTIS moves are needed.

Also, $2\sqrt{N}$ electronic moves are necessary. To see this, observe that B_0 is initially in processor (0,0) and its influence must be seen at all processors $(*, 0)$ because the $(*, 0)$ element of the result is A_*B_0, which is to be left in processor $(*, 0)$. OTIS moves can only transpose group and local processor indexes. To affect a change from (0,0) to $(*, 0)$, $2\sqrt{N} - 2$ electronic moves ($\sqrt{N} - 1$ rightward row moves and $\sqrt{N} - 1$ downward column moves) are essential. Further, A_0 is initially in (0,0) and all values in $(0, *)$ depend on A_0. Therefore at least $\sqrt{N} - 1$ rightward row moves and $\sqrt{N} - 1$ downward column moves are needed to communicate the A_0 value directly or indirectly to $(0, *)$. Since only unit data can flow along an electronic link in a single move, we cannot overlap all of the rightward row moves needed for A_0 and B_0. Therefore, at least $\sqrt{N}$ rightward moves must be made. Similarly at least $\sqrt{N}$ downward moves must be made. □

The algorithm of Figure 3 also can be used when A is a $kN \times 1$ vector and B a $1 \times kN$ vector. Now, in Steps 1 and 3, blocks of k values are moved from a single processor via an OTIS move. In Step 2, blocks of size k are to be broadcast. The strategy is the same as for the case $k = 1$; however, now we must pipeline the $2k$ A and B values in a processor for the column and row broadcast steps. This pipelining takes $2\sqrt{N} + 4k - 4$ electronic moves. Steps 1 and 3 still take 2 OTIS moves as we can move k element blocks using a single OTIS move. In Step 4, each processor performs k^2 multiplications to generate a $k \times k$ block of the product matrix.

3.1.2 GSM

For A an $N \times 1$ vector and B a $1 \times N$ vector, we start with A_i in processor $(i_f, 0, i_m, 0)$ and B_j in $(0, j_f, 0, j_m)$ and are to leave the product term $C_{ij} = A_iB_j$ in (i_f, j_f, i_m, j_m). The algorithm of Figure 4 does this.

Step 1 moves B_j from $(0, j_f, 0, j_m)$ to $(0, j_f, *, j_m)$ and Step 2 moves A_i from $(i_f, 0, i_m, 0)$ to $(i_f, 0, i_m, *)$. Following Step 3, B_j is in $(*, j_m, 0, j_f)$ and A_i is in $(i_m, *, i_f, 0)$. Step 4 now moves B_j from $(*, j_m, 0, j_f)$ to $(*, j_m, *, j_f)$ and Step 5 moves A_i from $(i_m, *, i_f, 0)$ to $(i_m, *, i_f, *)$. Following Step 6, processor (i_f, j_f, i_m, j_m) has A_i and B_j. Therefore, Step 7 correctly computes the product element $C_{ij} = A_ib_j$.

Step 1: Processors that have a B value broadcast this B value to all processors in the same column of the group.

Step 2: Processors that have an A value broadcast this A value to all processors in the same row of the group.

Step 3: Perform an OTIS move on the A and B values in a processor.

Step 4: Same as Step 1.

Step 5: Same as Step 2.

Step 6: Same as Step 3.

Step 7: Each processor multiplies its A and B values to produce an element of the product matrix.

Figure 4: GSM Column $\times$ Row multiply algorithm

The number of data moves is $4(\sqrt{N}-1)$ electronic and 2 OTIS moves. The algorithm of Figure 4 is optimal because the A_0 initially in (0,0,0,0) affects the final value in $(\sqrt{N}-1, 0, \sqrt{N}-1, 0)$. This requires $2(\sqrt{N}-1)$ electronic column moves. Further, the B_0 initially in (0,0,0,0) affects the final value in $(0, \sqrt{N}-1, 0, \sqrt{N}-1)$. This requires $2(\sqrt{N}-1)$ electronic row moves. Additionally, $\sqrt{N}$ A values initially in group (0,0) affect the final values in group (0,1). This requires at least 2 OTIS moves (assume that $\sqrt{N} \geq 2$).

The algorithm of Figure 4 can also be used when $k > 1$. Now, the broadcasts and each OTIS move involves 2 blocks of k elements each. The broadcasts are done by pipelining the transfer of the k elements in a block and each OTIS move simply does a block transfer of the k elements. The total number of data move steps becomes $4\sqrt{N} + 4k - 8$ electronic and 2 OTIS. Step 7 produces a $k \times k$ block of the result matrix using k^2 multiplication steps.

3.2 Row Vector $\times$ Column Vector

3.2.1 GRM

For a $1 \times N$ row vector A and an $N \times 1$ column vector B, we begin with A_i in $(0, i)$ and B_i in $(i, 0)$. The result $\sum_{i=0}^{N-1} A_i B_i$ is to be left in (0,0). In the algorithm of Figure 5, B_i is moved from $(i, 0)$ to $(0, i)$ in Step 1. The sum $\sum_{i=0}^{N-1} A_i B_i$ is computed in Step 3 by first moving the products of Step 2 upward to row 0 and adding terms in the row zero processors. Then the partial sums are moved leftward along row zero and the result computed in (0,0). The algorithm requires 1 OTIS and $2(\sqrt{N}-1)$ electronic moves. It is obvious that the algorithm is optimal.

Step 1: Perform an OTIS move on B values.

Step 2: Each processor of group 0 multiplies its A and B values.

Step 3: Sum the products of Step 2 by columns and finally along row zero, leaving the result in (0,0).

Figure 5: GRM Row $\times$ Column Multiply

Step 1: Groups with A values move their A values from row 0 to column 0 using the data paths of Figure 7.

Step 2: Perform an OTIS move on all data.

Step 3: Shift the A values leftward along row 0 of a group and the B values upward along column 0 and compute the sum of products in the (0,0) processor of each group that has A and B values.

Step 4: Perform an OTIS move on the product sums computed in Step 3.

Step 5: Shift the product sums upward along column 0 of group 0, summing these sums in processor (0,0).

Figure 6: GSM Row $\times$ Column Multiply

When the vectors are of size $1 \times kN$ and $kN \times 1$, respectively, Step 2 multiplies a $1 \times k$ block of A with a $k \times 1$ block of B. This takes $O(k)$ time. We assume that the cost of $O(k)$ arithmetics is considerably less than the cost of an electronic move and, therefore, make no attempt to utilize processors from other groups to reduce the time spent on arithmetic operations. The data moves required by the algorithm of Figure 5 still are $2(\sqrt{N}-1)$ electronic and 1 OTIS.

3.2.2 GSM

When multiplying a $1 \times N$ vector and an $N \times 1$ vector using the GSM mapping, the algorithm of Figure 6 can be used.

The algorithm of Figure 6 begins with A_i in $(0, i_f, 0, i_m)$ and B_i in $(i_f, 0, i_m, 0)$. In Step 1, A_i is moved to $(0, i_f, i_m, 0)$ by performing $\sqrt{N}-1$ downward moves and $\sqrt{N}-1$ leftward moves as in Figure 7. Following Step 2, A_i is in $(i_m, 0, 0, i_f)$ and B_i is in $(i_m, 0, i_f, 0)$. In Step 3, $(i_m, 0, 0, 0)$ sums up $\sqrt{N}$ of the terms that contribute to the result. In Step 4, these sums are moved to $(0, 0, i_m, 0)$ and are added together in Step 5. Steps 1 and 3 take $2(\sqrt{N}-1)$ electronic moves each and Step 5 takes $\sqrt{N}-1$ electronic moves. The total number of data moves is therefore $5(\sqrt{N}-1)$ electronic

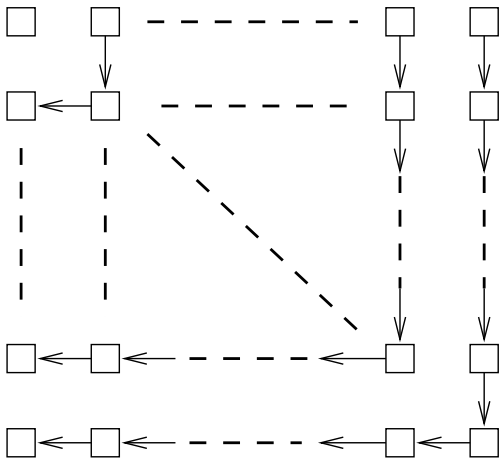

Figure 7: Data Paths Used in Step 1 of Figure 6

and 2 OTIS moves.

A straightforward generalization of the algorithm of Figure 6 to the case when we care multiplying a $1 \times kN$ row with a $kN \times 1$ column results in excessive complexity when $k > 1$. This is so because the pipelining of Step 3 takes $2k(\sqrt{N} - 1)$ electronic moves. When $k > 1$, the number of data moves is reduced by using the algorithm of Figure 8.

The algorithm of Figure 8 begins with the ith block of A in $(0, i_f, 0, i_m)$ and the ith block of B in $(i_f, 0, i_m, 0)$. In Step 1, the ith block of A is moved to $(0, i_f, i_m, 0)$. And following Step 2, the ith block of A is in $(i_m, 0, 0, i_f)$ while the ith block of B is in $(i_m, 0, i_f, 0)$. Step 3 moves the ith block of A from $(i_m, 0, 0, i_f)$ to $(i_m, 0, i_f, 0)$. Now $(i_m, 0, i_f, 0)$ contains block i of A and B. These blocks are multiplied in Step 4 to produce a single number in $(i_m, 0, i_f, 0)$. In Step 5, the numbers computed in group $(i_m, 0)$ are summed in processor $(i_m, 0, 0, 0)$. The OTIS move of Step 6 moves the resultant sums to $(0, 0, i_m, 0)$. These resultant sums are added together in Step 7.

The number of data moves performed by the algorithm of Figure 8 is $6\sqrt{N} + 4k - 10$ electronic and 2 OTIS.

3.3 Row Vector $\times$ Matrix

3.3.1 GRM

We are to multiply a $1 \times N$ row vector A and an $N \times N$ matrix B. The result is a $1 \times N$ vector C such that $C_i = \sum_{j=0}^{N-1} A_j B_{ji}$. Initially, A_i is in $(0, i)$ and B_{ij} is in (i, j) and the result is to be left so that C_i is in $(0, i)$. The multiplication algorithm is given in Figure 9.

In Step 1, A_i is moved from $(0, i)$ to $(i, 0)$. Following Step 2, processor (j, i) has A_j and

Step 1: Groups with A values move their A value blocks from row 0 to column 0 using pipelining and the data paths of Figure 7.

Step 2: Perform an OTIS move on all data blocks.

Step 3: Same as Step 1.

Step 4: Processors with an A and a B block multiply their blocks (these are the column 0 processors of each column 0 group).

Step 5: The column 0 processors, in each column 0 group, shift their Step 4 results upward along column 0. The results are added together by the (0,0) processor in each group.

Step 6: Perform an OTIS move on the sums computed by the (0,0) processors in Step 5.

Step 7: In group (0,0), the column 0 processors shift the values received in Step 6 upward to the (0,0) processor of the group. The (0,0) processor adds these values together.

Figure 8: GSM Row $\times$ Column Multiply for $k > 1$

Step 1: Perform an OTIS move on A values.

Step 2: Processor 0 of each group broadcasts its A value to the remaining processors in its group.

Step 3: All processors multiply their A and B values.

Step 4: Perform an OTIS move on the products computed in Step 3.

Step 5: Processor 0 of each group sums the products from all processors in the same group.

Step 6: Perform an OTIS move on the sums computed in Step 5.

Figure 9: GRM Row Vector $\times$ Matrix Multiply

Step 1: In each group move the A values from row 0 to column 0 using the data paths of Figure 7.

Step 2: The column 0 processors broadcast their A values to all processors in the same group and on the same row.

Step 3: Perform an OTIS move on all A and B values.

Step 4: Same as Step 1.

Step 5: Same as Step 2.

Step 6: All processors multiply their A and B values.

Step 7: The processor in row 0 of each group sum the products of Step 6 that are in the same column.

Step 8: Perform an OTIS move on the sums of Step 7.

Step 9: The processors in row 0 of group $(0, *)$ sum the values received in Step 8 that are in the same column.

Figure 10: GSM Row Vector $\times$ Matrix Multiply

B_{ji}. Processor (j, i) computes $A_j B_{ji}$ in Step 3 and in Step 4, $A_j B_{ji}$ is moved to processor (i, j). Processor $(i, 0)$ computes $C_i = \sum_{j=0}^{N-1} A_j B_{ji}$ in Step 5. In Step 6, C_i is moved from $(i, 0)$ to $(0, i)$. The complexity of the algorithm is $4(\sqrt{N} - 1)$ electronic and 3 OTIS moves.

When A is a $1 \times kN$ vector and B a $kN \times kN$ matrix, a block of k A values are moved in Step 1 of Figure 9; the broadcast of the A block in Step 2 is done in $2(\sqrt{N} + k - 2)$ electronic moves by pipelining the broadcast of the k values; the multiplication of Step 3 is between a $1 \times k$ vector and a $k \times k$ matrix; and the OTIS move of Step 4 moves $1 \times k$ blocks. To do the sum of Step 5, we first sum along rows. This is done in $\sqrt{N} + k - 2$ electronic moves by pipelining the k sums to be computed. Next the partial sums in column 0 are summed; again using pipelining. Step 5 takes $2(\sqrt{N} + k - 2)$ steps. Adding in the OTIS move of Step 6, the total number of moves becomes $4\sqrt{N} + 4k - 8$ electronic and 3 OTIS.

3.3.2 GSM

Our GSM algorithm to multiply a $1 \times N$ row vector A and an $N \times N$ matrix B is given in Figure 10. Note that the algorithm begins with A_j in $(0, j_f, 0, j_m)$ and B_{ji} in (j_f, i_f, j_m, i_m).

Step 1 moves A_j from $(0, j_f, 0, j_m)$ to $(0, j_f, j_m, 0)$ and following Step 2, A_j is in $(0, j_f, j_m, *)$. Following Step 3, A_j is in $(j_m, *, 0, j_f)$ and B_{ji} is in (j_m, i_m, j_f, i_f). After Step 5, A_j is in $(j_m, *, j_f, *)$.

Step 1: Processor 0 of each group broadcasts its B value to all processors in its group.

Step 2: Perform an OTIS move on B values.

Step 3: All processors multiply their A and B values.

Step 4: Processor 0 of each group sums the products computed in Step 3 by all processors in its group.

Figure 11: GRM Matrix × Column Vector Multiply

Therefore, in Step 6, processor (j_m, i_m, j_f, i_f) computes $A_j B_{ji}$. In Step 7, processor $(j_m, i_m, 0, i_f)$ computes $\sum_{j \ni j \bmod \sqrt{N} = j_m} A_j B_{ji}$, which is then sent to $(0, i_f, j_m, i_m)$ in Step 8. Finally in Step 9 $(0, i_f, 0, i_m)$ computes $C_i = \sum_{j=0}^{N-1} A_j B_{ji}$.

Steps 1 and 4 take $2(\sqrt{N}-1)$ electronic moves each; Steps 2, 5, 7, and 9 take $\sqrt{N}-1$ electronic moves each; and Steps 3 and 8 take 1 OTIS move each. The total number of moves is $8(\sqrt{N}-1)$ electronic and 2 OTIS.

When A is a $1 \times kN$ vector and B a $kN \times kN$ matrix, Steps 1 and 4 can be done with $2(\sqrt{N}+k-2)$ electronic moves each by transmitting the k values in each processor in a pipelined fashion; Steps 2 and 5 take $\sqrt{N}+k-2$ (again using pipelining) electronic moves; Steps 7 and 9 can be done in $\sqrt{N}+k-2$ moves each using the pipelined summing scheme used in Step 5 of Figure 9. The total number of moves is $8(\sqrt{N}+k-2)$ electronic and 2 OTIS.

3.4 Matrix × Column Vector

3.4.1 GRM

We start with an $N \times N$ matrix A and an $N \times 1$ column vector B and compute the $N \times 1$ column vector C such that $C_i = \sum_{j=0}^{N-1} A_{ij} B_j$. Initially, A_{ij} is in (i, j) and B_j is in $(j, 0)$. On termination, C_i is to be in $(i, 0)$. Our algorithm to perform the multiplication is given in Figure 11.

Following Step 1, B_j is in $(j, *)$ and following Step 2 it is in $(*, j)$. In Step 3, (i, j) computes $A_{ij} B_j$ and in Step 4, $(i, 0)$ computes $C_i = \sum_{j=0}^{N-1} A_{ij} B_j$. The number of data moves is $4(\sqrt{N}-1)$ electronic (Steps 1 and 4 each require $2(\sqrt{N}-1)$ electronic moves) and 1 OTIS.

Theorem 2 *The GRM matrix × column vector multiplication algorithm of Figure 11 is optimal.*

Proof Since the value of C_0 depends on all A_{0j} values, information about all these A values must get to (0,0) either directly or indirectly. For this to happen, at least $\sqrt{N}-1$ leftward row moves and

Step 1: In each group move the B values from column 0 to row 0 using the data paths of Figure 7 in the reverse direction.

Step 2: The row 0 processors of each group broadcast their B values to all processors in the same group and on the same column.

Step 3: Perform an OTIS moves on all A and B values.

Step 4: Same as Step 1.

Step 5: Same as Step 2.

Step 6: All processors multiply their A and B values.

Step 7: The processor in column 0 of each group sum the products of Step 6 that are in the same row.

Step 8: Perform an OTIS move on the sums of Step 7.

Step 9: The processors in column 0 of group $(*, 0)$ sum the values received in Step 8 that are in the same row.

Figure 12: GSM Matrix $\times$ Column Vector Multiply

$\sqrt{N} - 1$ upward column moves must be made. Let the snake-like row-major index of the bottom right processor of a group be q. Since $C_q = \sum A_{qj}B_j$, information originally in (0,0) (*i.e.*, B_0) must get to $(q, 0)$ directly or indirectly. This requires a minimum of $\sqrt{N} - 1$ rightward row moves and $\sqrt{N} - 1$ downward column moves plus one OTIS move. The row and column moves required for the computation of C_0 and C_q are in opposite directions and cannot be overlapped in the SIMD model. Therefore, at least $4(\sqrt{N} - 1)$ electronic and 1 OTIS moves are needed. □

When A is a $kN \times kN$ matrix and B a $kN \times 1$ vector, we use the algorithm of Figure 11 and pipelining as used for the case when A is a $1 \times kN$ vector and B a $kN \times kN$ matrix. The number of moves is $4(\sqrt{N} + k - 2)$ electronic and 1 OTIS.

3.4.2 GSM

The GSM matrix $\times$ vector multiplication algorithm is very similar to the GSM vector $\times$ matrix algorithm of Figure 10. The steps are given in Figure 12. Note that we start with A_{ij} in (i_f, j_f, i_m, j_m) and B_i in $(i_f, 0, i_m, 0)$.

In Step 1, B_j is moved from $(j_f, 0, j_m, 0)$ to $(j_f, 0, 0, j_m)$. Following Step 2, B_j is in $(j_f, 0, *, j_m)$. The OTIS move of Step 3 moves B_j to $(*, j_m, j_f, 0)$ and A_{ij} to (i_m, j_m, i_f, j_f). Steps 4 and 5 first move B_j to $(*, j_m, 0, j_f)$ and then to $(*, j_m, *, j_f)$. Following Step 5, (i_m, j_m, i_f, j_f) has A_{ij}

Step 1: Perform an OTIS move on B.

Step 2: Each processor of each group accumulates all A and B values in its group.

Step 3: Move the accumulated B values along the OTIS connection.

Step 4: Each processor computes the inner product of the A and B values it has.

Figure 13: $O(N)$ Memory GRM Matrix $\times$ Matrix Multiply

and B_j. In Step 6, (i_m, j_m, i_f, j_f) computes $A_{ij}B_j$. In Step 7, processor $(i_m, j_m, i_f, 0)$ computes $\sum_{j \ni j \bmod \sqrt{N} = j_m} A_{ij}B_j$, which is then sent to $(i_f, 0, i_m, j_m)$ in Step 8. Finally, in Step 9 $(i_f, 0, i_m, 0)$ computes $\sum_{j=0}^{N-1} A_{ij}B_j$. The total number of data moves is $8(\sqrt{N} - 1)$ electronic and 2 OTIS.

In the case when A is a $kN \times kN$ matrix and B a $kN \times 1$ vector, we use the algorithm of Figure 12 and pipelining as used for the case when A is a $1 \times kN$ vector and B a $kN \times kN$ matrix. The number of moves is $8(\sqrt{N} + k - 2)$ electronic and 2 OTIS.

3.5 Matrix $\times$ Matrix

3.5.1 $O(N)$ Memory/Processor Algorithms

When each processor has $O(N)$ memory, it is possible to accumulate an entire column (or row) into each processor. This leads to simplified algorithms. Consider the case when we are to multiply two $N \times N$ matrices A and B.

3.5.1.1 GRM

The GRM algorithm is given in Figure 13.

We begin with A_{ij} and B_{ij} in (i, j). Following Step 1, (i, j) has A_{ij} and B_{ji}. After Step 2, (i, j) has row i of A and column i of B. Following Step 3, (i, j) has row i of A and column j of B. In Step 4, (i, j) computes $C_{ij} = \sum_{k=0}^{N-1} A_{ik}B_{kj}$.

Step 2 can be done in two stages. In the first stage, the B values are accumulated; and in the second stage the A values are accumulated. To accumulate the B values, each processor first accumulates all values from its row. This takes $\sqrt{N} - 1$ rightward and $\sqrt{N} - 1$ leftward moves. Next, the accumulated blocks of $\sqrt{N}$ values are accumulated along columns by making $\sqrt{N} - 1$ upward and $\sqrt{N} - 1$ downward moves of blocks of size $\sqrt{N}$. The total stage 1 moves are $2\sqrt{N}(\sqrt{N} - 1) + 2(\sqrt{N} - 1) = 2(N - 1)$. Stage 2 is done similarly. Step 3 takes N/K OTIS moves where K is the maximum number of B values that can be moved in unit time over an optical link (for simplicity, we assume that K divides N). The total number of moves needed by the algorithm

Step 1: Perform an OTIS move on A and B values.

Step 2: Each processor accumulates all $\sqrt{N}$ A values in the same row and group as well as all $\sqrt{N}$ B values in the same column and group.

Step 3: Move the accumulated A and B values along the OTIS connection.

Step 4: Each processor accumulates all N A values in the same row and group as well as all N B values in the same row and group.

Step 5: Each processor computes the inner product of the A and B values it has.

Figure 14: $O(N)$ Memory GSM Matrix $\times$ Matrix Multiply

of Figure 13 is $4(N-1)$ electronic and $N/K+1$ OTIS. Each processor needs memory for N A values and N B values. The memory requirements can be reduced to $N+\sqrt{N}$ by delaying stage 2 of Step 2 to after Step 3 and coupling Step 4 with the columnwise movement of the $\sqrt{N}$ size packets of A during stage 2.

The algorithm of Figure 13 is easily generalized to the case when A and B are $kN \times kN$ matrices. Operations previously performed on matrix elements are now performed on $k \times k$ blocks of elements. The data movement counts are 1 OTIS in Step 1, $4k^2(N-1)$ electronic in Step 2, and k^2N/K OTIS in Step 3. The total is $4k^2(N-1)$ electronic and $k^2N/K+1$ OTIS.

3.5.1.2 GSM

Our GSM algorithm to multiply two $N \times N$ matrices is given in Figure 14.

Following Step 1, A_{ij} and B_{ij} are in (i_m, j_m, i_f, j_f). Following Step 2, $(i_m, j_m, i_f, *)$ has the $\sqrt{N}$ A values A_{iq} such that $q_m = j_m$ and $(i_m, j_m, *, j_f)$ has the $\sqrt{N}$ B values B_{rj} such that $r_m = i_m$. These $\sqrt{N}$ blocks of A and B values are then moved to $(i_f, *, i_m, j_m)$ and $(*, j_f, i_m, j_m)$, respectively. Following Step 4, $(i_f, *, i_m, *)$ has row i of A and $(*, j_f, *, j_m)$ has column j of B. Therefore, (i_f, j_f, i_m, j_m) has row i of A and column j of B. The inner product computation of Step 5 leaves $C_{ij} = \sum_{k=0}^{N-1} A_{ik}B_{kj}$ in (i_f, j_f, i_m, j_m).

Step 1 takes 1 OTIS move. Step 2 takes $2(\sqrt{N}-1)$ electronic row moves to accumulate the A values and $2(\sqrt{N}-1)$ electronic column moves to accumulate the B values. Step 3 takes $2\sqrt{N}/K$ OTIS moves. In Step 4, each electronic move moves $\sqrt{N}$ data. Since a total of $4(\sqrt{N}-1)$ moves are made, the total cost is $4\sqrt{N}(\sqrt{N}-1)$ unit electronic moves. Hence the total number of moves is $4(N-1)$ electronic and $2\sqrt{N}/K+1$ OTIS.

The algorithm is easily extended to the case when A and B are $kN \times kN$ matrices. The number

Step 1: [Align Matrix Elements] Move $A_{i,(j+i) \bmod N}$ and $B_{(j+i) \bmod N,j}$ to mesh processor (i,j).

Step 2: [Initialize C_{ij}] Processor (i,j) initializes its C value to the product of its A and B values.

Step 3: [Compute and Add Remaining Terms]

Repeat $N-1$ times:
{ Shift A values left circularly by 1;
Shift B values up circularly by 1;
$C = C + A * B$; **}**

Figure 15: Cannon's Matrix Multiplication Algorithm

of moves is $4k^2(N-1)$ electronic and $2k^2\sqrt{N}/K + 1$ OTIS.

3.5.2 $O(1)$ Memory/Processor Algorithms

Our $O(1)$ memory algorithm is based on Cannon's algorithm [1] to multiply two $N \times N$ matrices on an $N \times N$ mesh connected computer. Cannon's algorithm was also used by Dekel, Nassimi, and Sahni [6] in their development of hypercube algorithms for the matrix multiplication. Cannon's algorithm is given in Figure 15.

3.5.2.1 GRM

We simulate Cannon's algorithm on the OTIS-Mesh. While obtaining the alignment of Step 1, we also obtain the reverse of each aligned row of A and each aligned column of B. The process for B is similar to that used for A except that we must precede and follow the algorithm for A by an OTIS move (the preceding OTIS move gets all B elements in the same column into the same group and the following OTIS move gets the columns to the proper processors). We describe the alignment of rows of A only (Figure 16).

Steps 1 and 2 each take $\sqrt{N}-1$ electronic moves. Following Step 2, a processor can have up to four A values – 2 belonging to the aligned ordering of Step 1 of Cannon's algorithm, and 2 belonging to the reverse of this ordering. Each row contains a total of $2\sqrt{N}$ values with each processor in the row having 0, 1, 2, 3, or 4 values. These values can be moved to the proper processors on the same row by making $O(\sqrt{N})$ leftward and rightward data moves. To align B takes $O(\sqrt{N})$ electronic and 2 OTIS moves. Therefore, making $O(\sqrt{N})$ electronic and 2 OTIS moves, we can obtain the Step 1 alignment of Cannon's algorithm as well as the reverse of this alignment.

The circular shift of A in Step 3 of Cannon's algorithm can be implemented as a forward shift

Step 1: In each group, move A values upward along columns. As data moves through a processor, the processor saves a copy in case it is needed in the row the processor is in.

Step 2: Same as Step 1 except that A values are moved downward.

Step 3: In each row of each group, form the forward ordering. In this ordering, mesh processor (i, j) gets $A_{i,(j+i) \bmod N}$.

Step 4: In each row of each group, form the reverse ordering. In this ordering, the A values accumulated in each row, in Step 3, are reversed.

Figure 16: Moving A Values as per Step 1 of Cannon's Algorithm

along the snake of the reverse alignment and a backward shift along the snake of the aligned data. So each circular shift takes 4 electronic moves.

To do the circular shift on B, we retain a copy of the aligned and reversed B in each group prior to the second OTIS move done in Step 1. For each circular shift, we make 4 electronic moves in each group on the copy of B and then do an OTIS move to get the shifted B values to the desired processors. The total moves required by Step 3 is $(N-1) \times$ (8 electronic and 1 OTIS) $= 8(N-1)$ electronic and $N-1$ OTIS. Therefore, the GRM simulation of Cannon's algorithm can be done using $8N + O(\sqrt{N})$ electronic and $N+1$ OTIS moves.

The simulation just described works even when A and B are $kN \times kN$ matrices. Now, each element that is moved is a $k \times k$ block. Therefore an electronic block move takes k^2 electronic move steps. The number of moves becomes $8k^2N + O(k^2\sqrt{N})$ electronic and $N+1$ OTIS.

3.5.2.2 GSM

To multiply two $N \times N$ matrices we use a two level simulation of Cannon's algorithm. At the top level, we view each $N \times N$ matrix as a $\sqrt{N} \times \sqrt{N}$ matrix in which each element is a $\sqrt{N} \times \sqrt{N}$ submatrix. Let A and B be the $N \times N$ matrices to be multiplied and let BA and BB be the corresponding $\sqrt{N} \times \sqrt{N}$ matrices in which each element is a $\sqrt{N} \times \sqrt{N}$ block or submatrix of A and B, respectively. Initially, BA_{ij} and BB_{ij} are in group (i, j) of the OTIS-Mesh. Let $C = A \times B$ and let BC be the corresponding $\sqrt{N} \times \sqrt{N}$ matrix of blocks of size $\sqrt{N} \times \sqrt{N}$ each. Since $BC_{ij} = \sum_{k=0}^{\sqrt{N}-1} BA_{ik} \times BB_{kj}$, we can use Cannon's algorithm to compute BC. The products of Steps 2 and 3 now are products of submatrices or blocks of size $\sqrt{N} \times \sqrt{N}$, each block is in an OTIS group which is a $\sqrt{N} \times \sqrt{N}$ mesh. These submatrix products can in turn be done using Cannon's algorithm (this is the second level application of Cannon's algorithm).

To implement the two level scheme, we use the algorithm of Figure 17.

Step 1: [Align A data within each group/block] Reorder A values in each row of each group so that the A value originally in $(*, *, i, (i+j) \bmod \sqrt{N})$ is now in $(*, *, i, j)$. Call this the forward A ordering. Also create the reverse of this row ordering in each group. Call this the backward ordering.

Step 2: [Align B data within each group/block] Reorder B values in each column of each group so that the B value originally in $(*, *, (i+j) \bmod \sqrt{N}, j)$ is now in $(*, *, i, j)$. Also create the backward column ordering for the Bs.

Step 3: [Align the A blocks] Rearrange the blocks of A values obtained in Step 1 so that the block originally in group $(i, (i+j) \bmod \sqrt{N})$ (*i.e.*, in processors $(i, (i+j) \bmod \sqrt{N}, *, *)$) is now in the group of (i, j). Also create the corresponding backward row ordering for the A blocks.

Step 4: [Align the B blocks] Rearrange the blocks of B values obtained in Step 2 so that the block originally in group $((i+j) \bmod \sqrt{N}, j)$ is now in group (i, j). Also create the corresponding backward column ordering for the B blocks.

Step 5: [Initialize block BC_{ij}] $BC_{ij} = BA_{ij} \times BB_{ij}$.

Step 6: [Compute and add remaining terms]

Repeat $\sqrt{N} - 1$ times:
{ Shift A blocks left circularly by 1 group;
Shift B blocks up circularly by 1 group;
$BC_{ij} = BC_{ij} + BA_{ij} \times BB_{ij}$; **}**

Figure 17: GSM Matrix $\times$ Matrix Multiply

Steps 1 and 2 do the data alignment necessary to perform Steps 2 and 3 of Cannon's algorithm to multiply two blocks/submatrices of size $\sqrt{N} \times \sqrt{N}$ each. The forward and backward ordering of the A values can be obtained by making $\sqrt{N} - 1$ leftward and $\sqrt{N} - 1$ rightward moves of A values. Similarly the forward and backward ordering of B values can be done using $2(\sqrt{N} - 1)$ column moves. Following Step 2, each processor has 2 A values (one from the forward ordering and the other from the backward ordering) and 2 B values.

In Steps 3 and 4 the $\sqrt{N} \times \sqrt{N}$ blocks of submatrices of A and B are aligned. For this, an OTIS move is made on the A and B values, followed by $2(\sqrt{N} - 1)$ electronic row moves and $2(\sqrt{N} - 1)$ electronic column moves, and finally an OTIS move. For the final OTIS move, we leave a copy of the As and Bs in the originating processors also. Now each processor has 8 A and 8 B values.

Step 5 is done using Steps 2 and 3 of Cannon's algorithm at a cost of $4(\sqrt{N} - 1)$ electronic moves. In Step 6, the A and B blocks are shifted by using the copies saved during the second OTIS moves of Steps 3 and 4 followed by an OTIS move. This shifting of A and B blocks takes 2 row electronic moves (both forward and backward A blocks are to be shifted in the opposite direction) plus 2 column electronic moves for B blocks and 1 OTIS move. The block matrix multiply is done using steps 2 and 3 of Cannon's algorithm at a cost of $4(\sqrt{N} - 1)$ electronic and $\sqrt{N} - 1$ OTIS moves. The total number of moves made by the GSM algorithm is $4N + O(\sqrt{N})$ electronic and $\sqrt{N}$ OTIS.

The algorithm of Figure 17 is easily extended to the case when A and B are $kN \times kN$ matrices. The essential difference is that each element of a $\sqrt{N} \times \sqrt{N}$ block is now itself a $k \times k$ block. So, each electronic data move becomes k^2 unit moves. The number of data moves is therefore $4k^2N + O(k^2\sqrt{N})$ electronic and $\sqrt{N}$ OTIS.

4 Conclusion

We have developed OTIS-Mesh algorithms for several variants of the matrix multiplication problem. For each variant, we have considered both the group row mapping and the group submatrix mapping. Our results are summarized in Table 1. As can be seen, the GSM mapping is superior for the case of matrix $\times$ matrix multiplication. However, for all other variants the GRM is superior. As noted in [25], GRM is also superior for the matrix transpose operation.

Embedding Scheme	GRM		GSM	
Operation	Electronic	OTIS	Electronic	OTIS
Column × Row	$2\sqrt{N}$	2	$4(\sqrt{N}-1)$	2
Row × Column	$2(\sqrt{N}-1)$	1	$5(\sqrt{N}-1)$	2
Row × Matrix	$4(\sqrt{N}-1)$	3	$8(\sqrt{N}-1)$	2
Matrix × Column	$4(\sqrt{N}-1)$	1	$8(\sqrt{N}-1)$	2
Matrix × Matrix ($O(N)$)	$4(N-1)$	$N/K+1$	$4(N-1)$	$2\sqrt{N}/K+1$
Matrix × Matrix ($O(1)$)	$8N+O(\sqrt{N})$	$N+1$	$4N+O(\sqrt{N})$	$\sqrt{N}$

Table 1: Comparison between GRM and GSM schemes

References

[1] L. E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm.* PhD thesis, Montana State University, 1969.

[2] S.-G. Chen, J.-C. Lee, and C.-C. Li. New systolic arrays for matrix multiplication. In *Proceedings of 1994 International Conference of Parallel Processing*, pages II–211 – II–215, 1994.

[3] J. Choi. A new parallel matrix multiplication algorithm on distributed-memory concurrent computers. *Concurrency, Practice and Experience*, 10(8):655, 1998.

[4] J. Choi, J. J. Dongarra, and D. W. Walker. PUMMA: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurrency, Practice and Experience*, 6(7):543–570, 1994.

[5] C.-C. Chou, Y.-F. Deng, G. Li, and Y. Wang. Parallelizing Strassen's method for matrix multiplication on distributed-memory MIMD architectures. *Computers & Mathematics with Applications*, 30(2):49, 1995.

[6] E. Dekel, D. Nassimi, and S. Sahni. Parallel matrix and graph algorithms. *SIAM Journal on Computing*, 10(4):657–675, Nov. 1981.

[7] M. Feldman, S. Esener, C. Guest, and S. Lee. Comparison between electrical and free-space optical interconnects based on power and speed considerations. *Applied Optics*, 27(9):1742–1751, May 1988.

[8] A. Gupta and V. Kumar. Scalability of parallel algorithms for matrix multiplication. In *Proceedings of 1993 International Conference on Parallel Processing*, pages III–115 – III–123, 1993.

[9] H. Gupta and P. Sadayappan. Communication-efficient matrix multiplication on hypercubes. *Parallel Computing*, 22(1):75, 1996.

[10] P. Gupta. An efficient matrix multiplication algorithm on EREW model. In *Proceedings of the International Conference on Robotics, Vision and Parallel Processing for Industrial Automation (Rovpia'94)*, pages 104–108, 1994.

[11] W. Hendrick, O. Kibar, P. Marchand, C. Fan, D. V. Blerkom, F. McCormick, I. Cokgor, M. Hansen, and S. Esener. Modeling and optimization of the optical transpose interconnection system. in Optoelectronic Technology Center, Program Review, Cornell University, Sept. 1995.

[12] S. L. Johnsson. Minimizing the communication time for matrix multiplication on multiprocessors. *Parallel Computing*, 19(11):1235, 1993.

[13] B. Kaagstroem and M. Raennar. Distributed general matrix multiply and add for a 2D mesh processing networks. *Lecture Notes in Computer Science*, (1041):333–344, 1996.

[14] L. Keqin, P. Yin, and S.-Q. Zheng. Novel implementations of parallel matrix multiplication algorithms using optical buses. In *Proceedings of 1998 International Conference of Parallel and Distributed Processing Techniques and Applications*, pages 1212–1220, 1998.

[15] A. Krishnamoorthy, P. Marchand, F. Kiamilev, and S. Esener. Grain-size considerations for optoelectronic multistage interconnection networks. *Applied Optics*, 31(26):5480–5507, Sept. 1992.

[16] P. Z. Lee. Parallel matrix multiplication algorithms on hypercube multiprocessors. *International Journal of High Speed Computing*, 7(3), 1995.

[17] K. Li and V. Pan. Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system. In *Proceedings of the International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 31–37, 1999.

[18] G. C. Marsden, P. J. Marchand, P. Harvey, and S. C. Esener. Optical transpose interconnection system architectures. *Optics Letters*, 18(13):1083–1085, July 1 1993.

[19] W. F. McColl and A. Tiskin. Memory-efficient matrix multiplication in the BSP model. *Algorithmica*, 24(3):287, 1999.

[20] M. Middendorf, H. Schmeck, and G. Turner. Sparse matrix multiplication on a reconfigurable mesh. *The Australian Computer Journal*, 27(2):37, 1995.

[21] P. A. Nelson. Hypercube matrix multiplication. *Parallel Computing*, 19(7):777, 1993.

[22] J. S. Oliver. Matrix multiplication with DNA. *Journal of Molecular Evolution*, 45(2):161, 1997.

[23] A. Osterloh. Sorting on the OTIS-Mesh. In *Proceedings of the 14th International Parallel & Distributed Processing Symposium (IPDPS'2000)*, pages 269–274, 2000.

[24] S. Rajasekeran and S. Sahni. Randomized routing, selection, and sorting on the otis-mesh. *IEEE Transactions On Parallel And Distributed Systems*, 9(9):833–840, 1998.

[25] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Mesh optoelectronic computer. In *Proceedings of the fourth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'97)*, pages 130–135, 1997.

[26] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Hypercube optoelectronic computer. *Informatica*, 22:263–269, 1998.

[27] C. A. A. Sanches and S. W. Song. SIMD algorithms for matrix multiplication on the hypercube. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 492–496, 1994.

[28] J. F. Tasic, M. Zajc, and A. Kosir. Comparison of some parallel matrix multiplication algorithms. In *Proceedings of the 8th Mediterranean Electrotechnical Conference*, pages 155–158, 1996.

[29] J.-C. Tsay and S. Yuan. Some combinatorial aspects of parallel algorithm design for matrix multiplication. *IEEE Transactions on Computers*, 41(3):355–360, Mar. 1992.

[30] R. A. van de Geijn and J. Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency, Practice and Experience*, 9(4):255, 1997.

[31] C.-F. Wang and S. Sahni. Basic operations on the OTIS-Mesh optoelectronic computer. *IEEE Transactions on Parallel and Distributed Systems*, 9(12):1226–1236, 1998.

[32] C.-F. Wang and S. Sahni. Matrix multiplication on the OTIS-Mesh optoelectronic computer. In *Proceedings of the sixth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'99)*, pages 131–138, 1999.

[33] C.-F. Wang and S. Sahni. Image processing on the OTIS-Mesh optoelectronic computer. *IEEE Transactions on Parallel and Distributed Systems*, 11(2):97–109, 2000.

[34] X. Wu. An approach to scalability of parallel matrix multiplication algorithms. *Lecture Notes in Computer Science*, (1276):492–501, 1997.

[35] F. Zane, P. Marchand, R. Paturi, and S. Esener. Scalable network architectures using the optical transpose interconnection system (OTIS). In *Proceedings of the second International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'96)*, pages 114–121, 1996.