Journal of Network and Computer Applications (1996) **19,** 235–248

Singular value decomposition on processor arrays with a pipelined bus system

Yi Pan* and Mounir Hamdi†

**Computer Science Department, University of Dayton, Dayton, OH 45469-2160, USA and* †*Computer Science Department, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong*

Singular value decomposition (SVD) is used in many applications such as real-time signal processing where fast computation of these problems is needed. In this paper, parallel algorithms for solving the singular value decomposition problem are discussed. The algorithms are designed for optically interconnected multiprocessor systems where pipelined optical buses are used to connect processors. In a pipelined bus system, messages can be transmitted concurrently in a pipelined fashion. However, certain restrictions may apply in a pipelined bus system. For example, a processor can send at most one message and receive one message during a bus cycle. Pipelined bus interconnection requires us to rethink how we write parallel algorithms. Fully exploring the properties of concurrent message transmissions requires careful mapping of data, an efficient addressing mechanism, and a set of efficient basic data movement operations. In this paper, these issues are addressed in detail. Analysis of the parallel computation times of the SVD algorithms shows that they are asymptotically equivalent to those implemented on the hypercube while using substantially less hardware. The results obtained in this paper further demonstrate that optically interconnected multiprocessor systems are very promising as a new multiprocessor architecture.

1. Introduction

One of the important problems in mathematical science and engineering is singular value decomposition (SVD). It is often used in many applications such as real-time signal processing, where the solution to these problems is needed in the shortest possible time. SVD is one of the most important factorization of a real m-by-n ($m \geqslant n$) matrix, and is a very computationally intensive problem. Given the growing availability of parallel computers and the need for faster solutions to the SVD problem, there has been great interest in the development of parallel implementations of the singular value decomposition.

A singular value decomposition of a real m-by-n ($m \geqslant n$) matrix A is its factorization of this matrix into the product of three matrices:

$$A = UDV^T$$

where U is an m-by-n matrix with orthogonal columns, D is an n-by-n non-negative diagonal matrix, and V is an n-by-n orthogonal matrix. For more details, please refer to Golub and Reisch [1]. The most common method used for the solution of the SVD problem is incorporated in the Golub–Kahan SVD algorithm [1,2], which requires

1084–8045/96/030235 + 14 $18.00/0

$O(mn^2)$ time on a single processor computer. In order to reduce the computation time of this sequential algorithm, there has been much interest recently in developing faster parallel SVD [2–9]. On a linear processor array, the most efficient SVD algorithm is the Jacobi-like algorithm given by Brent and Luk [3]. The algorithm, which is based on a one-sided orthogonalization method due to Hestenes [10], takes $O(mnS)$ time on $O(n)$ processors where S is the number of sweeps. However, Brent and Luk's algorithm is not optimal in terms of communication overhead. Unnecessary costs are incurred by mapping the systolic array architecture onto a ring connected linear array due to the double sends and receives required between pairs of neighboring processors. Eberlein [11], Bischof [12] and others have proposed various modifications of this algorithm for hypercube implementations, which require the embedding of rings via binary reflected *Gray Codes*. Gao and Thomas [13] have investigated this problem using a recursive divide-exchange communication pattern. These algorithms have the same order of magnitude of time complexity as the one described by Brent and Luk, but reduce the constant factor. SVD algorithms on hypercube and shuffle-exchange computers have also been designed by Pan and Chuang [14]. Instead of mapping a column of data onto a processor in a hypercube, as is done in Chuang and Chen [5] and Gao and Thomas [13], they map a column pair of data onto a column of processors in a hypercube or a shuffle-exchange computer. Using this method, the total time is reduced to $O(n \log m)$ per sweep using $(mn)/(2 \log m)$ processors.

Most of the parallel algorithm of the SVD problem has been targeted to be executed on point-to-point parallel architectures. The data transmission in these architectures is quite different from that on arrays of processors with pipelined optical buses. One of the unique features of a pipelined optical bus is its ability to allow the current transmission of messages in a pipelined fashion while having a very short end-to-end message delay. This makes it very efficient in the connection of parallel computers. For this reason, arrays of processors with pipelined optical buses have attracted a lot of attention from researchers recently. We will briefly introduce this novel architecture in Section 3. Many important parallel algorithms have been implemented on such an architecture, such as broadcasting [15], sorting [16], selection [17], numerical problems [18], and Hough transform [19]. In this paper, we use this novel architecture for the implementation of efficient parallel algorithms for the SVD problem.

This paper is organized as follows. Hestenes' method for solving the SVD problem is reviewed in the next section. In Section 3, a parallel SVD algorithm, which is based on the Hestenes method, is implemented on a 1-D array of processors with a pipelined optical bus. In Section 4, a variation of this parallel algorithm is implemented on a 2-D array of processors with pipelined optical buses. In Section 5, oversized SVD problems are discussed. Section 6 gives a summary and assessment of our results.

2. Singular value decomposition

There is a wealth of serial SVD algorithms available in the literature. Most of these algorithms can be transformed into parallel algorithms to be run on parallel computers. Each of these serial algorithms exhibits a certain degree of parallelism. Thus, in choosing the serial algorithm to be transformed into a parallel algorithm, we have to take the potential of inherent parallelism within the algorithm into account. For this reason, we choose the Hestenes method of solving the SVD problem [10] to implement it on

the array of processors with optical pipelined buses because of its high potential of parallelism and its flexibility in being transformed efficiently into a parallel algorithm suited for our architecture, even though its sequential version is less efficient than that of the Golub–Kahan–Reinsch SVD method [1,2].

The basic idea of the decomposition is to generate an orthogonal matrix V such that the transformed matrix $AV = W$ has orthogonal columns. Normalizing the Euclidean length of each non-null column to unity, we get the relation

$$W = U'D \tag{1}$$

where U' is a matrix whose non-null columns form an orthonormal set of vectors, and D is a non-negative diagonal matrix. An SVD of A is then given using the following relation:

$$A = WV^T = U'DV^T \tag{2}$$

As a null column of U' is always associated with a zero diagonal element of D, (1) and (2) are essentially identical.

Hestenes [10] uses plane rotations to construct V. A sequence of matrices $\{A_k\}$ is generated by

$$A_{k+1} = A_k R_k \quad (k = 1, 2, 3, \ldots)$$

where $A_1 = A$, and each R_k is a plane rotation. Let $A_k = (a_1^k, a_2^k, \ldots, a_n^k)$, where a_i^k is the ith column of A_k. Suppose $R_k = (r_{ij}^k)$ is a rotation on the (p, q) plane which orthogonalizes columns a_p^k and a_q^k of A_k with $p<q$. R_k is an orthogonal matrix with all of its elements identical to those of the unit matrix except that

$$\begin{array}{ll} r_{pp}^k = \cos\theta & r_{pq}^k = \sin\theta \\ r_{qp}^k = -\sin\theta & r_{qq}^k = \cos\theta \end{array} \tag{3}$$

We note that post-multiplication of A_k by R_k affects only columns a_p^k and a_q^k, and that

$$(a_p^{k+1}, a_q^{k+1}) = (a_p^k, a_q^k)\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{4}$$

The rotation angle θ should be chosen so that the two new columns are orthogonal. For this we can use the formulas given by Rutishauser [20]. Defining:

$$\gamma = (a_p^k)^T a_q^k,\ \alpha = (a_p^k)^T a_p^k,\ \beta = (a_q^k)^T a_q^k \tag{5}$$

We set $\theta = 0$, if $\gamma = 0$. Otherwise, we compute

$$L = \frac{\beta - \alpha}{2\gamma}, t = \frac{\text{sign}(L)}{|L| + \sqrt{1 + L^2}} \tag{6a}$$

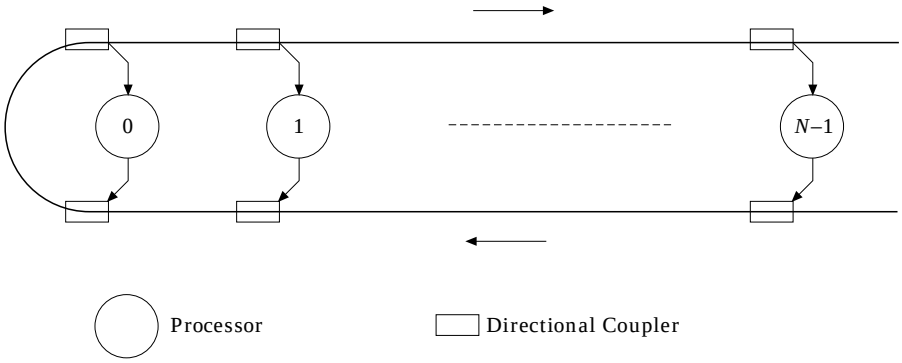

Figure 1. A 1-D array with an optical bus.

$$c = \cos\theta = \frac{1}{\sqrt{1+t^2}}, \quad s = \sin\theta = t\cos\theta \tag{6b}$$

The rotation angle θ always satisfies $|\theta| \leqslant (\pi/4)$.

In this way, we orthogonalize the pth and the qth columns in the kth step. The process in which all column pairs (i, j) for $i<j$ are orthogonalized exactly once is called a sweep. We repeat the sweep until A converges to W, which has orthogonal columns. In actual computation, we can select a small positive number ε and stop the computation when the inner product of every column pair is less than ε. We can use ε as a threshold, and orthogonalize a column pair only when its inner product is larger than ε. This threshold approach accelerates the convergence.

For a matrix with $n = 2^g$ columns, $n(n-1)/2$ column pairs have to be orthogonalized in a sweep. More than one column pair can be orthogonalized simultaneously in a parallel system. In the orthogonalization, the elements of the new columns can be computed in parallel. The inner product operation to compute the rotation parameters γ, α, and β can also be carried out in parallel. The critical issue here is how to compute the rotation parameters efficiently, perform orthogonalization of column pairs, and move columns to create all column pairs, using the pipelined optical buses available in the system.

3. SVD on 1-D arrays with a pipelined optical bus

Before we discuss the detailed implementation of the SVD algorithm on the 1-D array of processors (PEs) with a pipelined optical bus, it is appropriate to give a brief introduction to this novel architecture and discuss its functionality. Figure 1 shows a linear array of processors connected with an optical pipelined bus. Each processor is connected to the bus with two directional couplers (for the conversion of optical signals to electronic signals and vice versa), one for transmitting messages on the lower bus segment and the other for transmitting messages on the upper bus segment [15,17, 21–23]. Because of the properties of the optical bus (waveguides), messages propagate unidirectionally from right to left on the lower bus segment and from left to right on the upper bus segment. That is why two buses are needed for such a computer system,

unlike traditional electronic buses where electronic signals can propagate in either direction on the same bus. Each processor uses a set of control registers to store information needed to control the transmission and reception of messages by that processor which will be explained later. The above computer system can operate in MIMD mode where each processor would have its own controller, or in SIMD mode where a single controller is used for all the processors. In this paper, this parallel computer system is assumed to be operating in SIMD mode, and thus our algorithms are designed with this mode in mind.

For convenience, we let the optical path length between *any* two couplers on the bus be d_0 units. Denote the propagation time of a light signal between two couplers by *tc*; that is, $tc = d_0/c_g$, where c_g is the velocity of light on the bus. Let a *bus cycle*, *Tc*, be Ntc. With a concurrent bus accessing method, often referred to as bus pipelinings [21, 22], processors can transmit their messages simultaneously at the beginning of each bus cycle. As a result of a simultaneous transmission by all processors, a train of N messages is formed on the lower segment of the bus and will propagate from left to right on the upper segment. Processors that wish to receive any message from among the train of messages transmitted must wait for the message to arrive at its receiver [23]. These properties are a result of the unique characteristics of optical buses (waveguides). For more details, readers are referred to Melhem *et al.* [22].

Messages transmitted by different processors may overlap with each other even if they propagate unidirectionally on the bus. We call these message overlappings *transmission conflicts*. Assume each message has b binary bits, where each bit is represented by an optical pulse, with the existence of a pulse for 1 and the absence of a pulse for 0. Let w be the width (or duration) of each pulse in seconds. To ensure free of transmission conflicts, the following condition has to be satisfied:

$$tc > bw \tag{7}$$

We call *tc* a *pipeline* cycle; thus, the term pipelined bus is self-explanatory. The above condition ensures that each message will be 'fit' into a pipeline cycle such that, in a bus cycle, up to N messages can be transmitted by processors simultaneously without collisions on the bus. This is the biggest difference between an optical bus and electronic bus, where the former can handle up to N simultaneous messages when connected to n processors while the latter can handle just one message regardless of the number of processors. In a parallel array, messages normally have very short length (b is very small). Thus, in the remaining discussions, we assume that the condition given by eqn (7) above is always satisfied and no transmission conflicts are possible as long as all processors are synchronized at the beginning of each bus cycle.

In SIMD environments, each processor knows the source (processor id) of the message it is receiving and when it is sent relative to the beginning of a bus cycle. Therefore, the time (relative to the beginning of a bus cycle) can be easily calculated by the processor receiving the message. Let the processor that wishes to receive a message be processor i. Assume that the message is sent by processor j at the beginning of bus cycle C. We use a *receiving* function, $rec(i, j, C)$, to determine the time that processor i has to wait, relative to the beginning of bus cycle C, before receiving the message from processor j. The value of the function, in the number of pipeline cycles, is given in the following equation:

$$rec(i, j, C) = i + j + 1 \tag{8}$$

By calculating the receiving function as given by eqn (8), each processor would be able to selectively receive any message from a train of messages in each bus cycle. These values of receiving functions can be precomputed and stored in a set of *receiving control* registers for fast retrieval during the execution of the algorithm. The precomputing of these values is done, obviously, according to the design of the given algorithm. Readers may refer to Melhem *et al.* [22] for details.

Now, it is time to present the SVD algorithm on the 1-D array with an optical bus of size $n/2$, in which every PE has a local memory large enough to store a pair of columns of data elements. Suppose that the two columns are stored in arrays $A(i)$ and $B(i)$, $0 \leqslant i < n/2$, respectively. Every PE continuously receives commands from the controller to perform the orthogonalization of the column pair concurrently with other PEs according to eqns (3–6). A procedure called 1-D-ORTH is used to orthogonalize a column pair as follows. 1-D-ORTH consists of the following steps, i.e. to update α, γ, and β using (5), to calculate L, t, c, and s using (6), and to orthogonalize the column pair based on (4). Since the length of a column is m, it takes $O(m)$ to update (5). All other steps in 1-D-ORTH can be done in constant time. Hence, the time to perform 1-D-ORTH is $O(m)$.

After performing the orthogonalization of the column pair, each PE exchanges a column of data with another PE. Clearly, any exchange of column data between two PEs can be done in m bus cycles since each PE can put one message onto the bus and receive one message from the bus in one bus cycle and a column contains m data elements. Here, the exchange sequence described in Dekel *et al.* [24] is used to move data column $A(i)$ in PE[i], for all i, to all other PEs sequentially to produce all (A, B) column pairs, i.e. the set $\{(A_p, B_q) | p = 0, 1, 2, \ldots, n/2-1, q = 0, 1, 2, \ldots, n/2-1\}$. In order to obtain all possible column pairs, one still has to consider the pairing among the A columns and among the B columns. This can be achieved by iteratively applying the process of exchanging the A and the B columns between neighboring nodes to create subarrays and then moving the A columns within the subarrays. The entire algorithm is given in procedure 1-D-SVD.

```
    procedure 1-D-SVD
1   While not (all |γ|<ε) do
2   1-D-ORTH;
3   for k := 0 step 1 until g−1 do
4   for p := 1 step 1 until 2^(g−k)−1 do
5       h := f(g−k, p);
6       R_rc(i) ← 1 + i + i^(h);
7     for j := 1 step 1 until m do
8       Bus ← A[j](i);
9       A[j](i) ← Bus;
10    end for j;
11      1-D-ORTH;
12  end for p;
13  if r_k = 0 then
14      R_rc(i) ← 1 + i + i^(h);
```

```
15      for j:=1 step 1 until m do
16        Bus←B[j](i);
17        A[j](i)←Bus;
18      end for j;
19    else
20        R_rc(i)←1+i+i^(h);
21      for j:=step 1 until m do
22        Bus←A[j](i);
23        B[j](i)←Bus;
24      end for j;
25    1-D-ORTH;
26    end for k;
27    end while;
```

In the above algorithm, $f(g-k, p)$ is the pth number in the exchange sequence [24] of length $2^{g-k}-1$. $i^{(h)}$ is the number whose binary representation is $i_q, \ldots, i_h, \ldots, i_0$. $R_{rc}(i)$ is the receiving control register used to store the receiving function value in processor i. Thus, when $R_{rc}(i)$ is assigned the value of $1+i+i^{(h)}$, processor i receives a message automatically from processor $i^{(h)}$. $A[j](i)$ is the jth element of the data column stored in processor i. Column pairs are generated within a subarray by loop p using the exchange sequence [25]. Lines 13 to 24 iteratively exchange the A and B columns between neighboring nodes. Obviously, lines 8, 9, and 11 dominate the time complexity of the algorithm since they are statements within the double loop. The time spent in procedure 1-D-ORTH on line 11 is $O(m)$ since each PE needs to process a column pair whose length is m. The number of times line 11 is executed is given by:

$$\sum_{k=0}^{g-1}\left(\sum_{p=1}^{2^{g-k}-1} 1\right)=\left(\frac{n}{2}-1\right)+\left(\frac{n}{4}-1\right)+\left(\frac{n}{8}-1\right)\cdots+1\leqslant n. \tag{9}$$

Hence, the time spent in line 11 is $O(mn)$. Similarly, it can be shown that the number of times lines 8 and 9 are executed is

$$\begin{aligned}&\sum_{k=0}^{g-1} m(2^{g-k}-1)\\ &=m\left\{\left(\frac{n}{2}-1\right)+\left(\frac{n}{4}-1\right)+\left(\frac{n}{8}-1\right)\cdots+1\right\}\\ &\leqslant mn\end{aligned} \tag{10}$$

Thus, the time spent in lines 8 and 9 is $O(mn)$. Therefore, the total time of algorithm 1-D-SVD is $O(mn)$, which is on the same order of magnitude as that on a hypercube of the same size [5]. While a PE in a linear array of size n with a pipelined optical bus has only two communication ports, a PE in a hypercube of size n has $\log n$ ports.

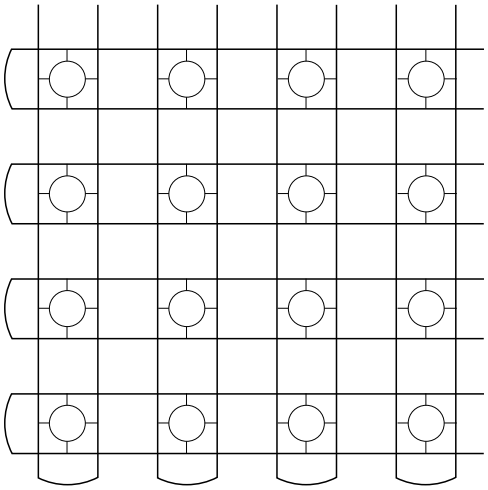

Figure 2. A 2-D array with row and column optical buses.

4. SVD on 2-D arrays with pipelined optical buses

The 1-D array with a single pipelined optical bus has certain limitations [23]. First, the system size cannot be very large since it is limited by the minimum power that can be detected by the PE at the very end of the bus (similar to electronic buses). Second, when the number of PEs, N, is very large, the end-to-end transmission delay increases since the condition given by eqn (3) has to be held and the number of pipeline cycles is proportional to N. Moreover, many problems, such as SVD, are 2-D in nature and are more efficient when implemented on a 2-D array of processors. To overcome the shortcomings of the linear array of processors, we consider a 2-D array of processors where each row of processors and each column of processors are connected by a pipelined optical bus. Thus, a PE in a 2-D array is coupled to four waveguides: two for transmitting messages and two for receiving messages (see Fig. 2). Communication between PEs in the same row or in the same column in a 2-D array can be carried the same way as in the linear case. If two PEs in different rows or different columns need to communicate, then two bus cycles are needed. That is, at the end of the first bus cycle a message has to be buffered at a node which is in the same row (or column) as the destination [15,21–23]. It is expected that parallel algorithms can be run more efficiently on a 2-D array than on a 1-D array as can be deduced from our SVD algorithms implemented on a 1-D array and a 2-D array. Nevertheless, a 1-D array with a pipelined optical bus is a legitimate architecture with its own merits [22].

Now we consider the implementation of the SVD problem on the 2-D array of processors with pipelined optical buses of size $m \times (n/2)$, where $n = 2^{g+1}$. Each PE in a row has an index which is between $[0, m-1]$ and each PE in a column has an index which is between $[0, n/2-1]$. Hence, each PE can be uniquely identified by its row index and column index. Each PE(i, j) has several registers; registers $A(i, j)$ and $B(i, j)$ are used to store matrix elements. The given matrix A is initially stored as follows:

$$A(i, j) = a_{ij},\ B(i, j) = a_{i,(n/2)+j}, \quad \text{for } 0 \leqslant i < m,\ 0 \leqslant j < n/2 \tag{11}$$

where a_{ij} is the (i, j) element of matrix A. Thus, the n columns of the matrix are partitioned into $n/2$ pairs and each column pair is stored in a column of m processors.

A column of m processors orthogonalizes the column pair stored in it by using the procedure 2-D-ORTH explained below. The procedure 2-D-ORTH is similar to 1-D-ORTH except that a column pair of data is stored in a column of processors instead of one processor. 2-D-ORTH is divided into several distinct phases and each phase corresponds to its counterpart in 1-D-ORTH. In the first phase, the local products

$$\begin{aligned} \gamma(i,j) &= A(i,j)^* B(i,j) \quad 0 \leqslant i < m \\ \alpha(i,j) &= A(i,j)^* A(i,j) \quad 0 \leqslant i < m \\ \beta(i,j) &= B(i,j)^* B(i,j) \quad 0 \leqslant i < m \end{aligned} \tag{12}$$

are calculated in the m processors concurrently. In the second phase, the sums

$$\sum_{i=0}^{m-1} \gamma(i,j), \sum_{i=0}^{m-1} \alpha(i,j), \text{ and } \sum_{i=0}^{m-1} \beta(i,j) \tag{13}$$

are computed by combining the partial sums using the reporting operation described in Hamdi and Pan [18], and the results are stored in PE$(0, j)$. In the third phase, the values $L(j)$, $t(j)$, $c(j)$, and $s(j)$ are computed locally in $P(0, j)$. In the fourth phase, the rotation parameters $c(0, j) = c(j)$ and $s(0, j) = s(j)$ are broadcast over the column buses to all processors in the same columns. The broadcast operation described in Hamdi and Pan [18] can be used here. In the fifth phase, elements of the two new matrix columns are computed locally using the rotation parameters received. The time complexity of the 2-D-ORTH can be derived as follows. The first phase takes a constant time since all operations are local. The second phase needs $O(\log m)$ time since these are reporting operations over columns of m PEs [18]. The third phase uses a constant time. In the fourth phase, one bus cycle is needed since it is a broadcasting operation over columns [18]. The fifth phase requires a constant time again. Hence, the time taken in 2-D-ORTH is $O(\log m)$.

After the procedure 2-D-ORTH is performed in parallel on all $n/2$ columns of processors, half of the newly computed matrix columns are exchanged in parallel between neighboring columns of processors. The procedure 2-D-ORTH is then carried out on all columns of processors in parallel again. Clearly, any column exchange between two PEs can be done in one bus cycle since this is a one-to-one operation. Here again the exchange sequence described in Dekel *et al.* [24] is used to move column $A[i, j]$ in PE$[i, j]$, for all i and j, to all other PEs sequentially to produce all (A, B) column pairs in a column of PEs, i.e. the set $\{(A_p, B_q) | p = 0, 1, 2, \ldots, n/2-1, q = 0, 1, 2, \ldots, n/2-1\}$. In order to obtain all possible column pairs, one still has to consider the pairing among the A columns and among the B columns. This can be achieved by iteratively applying the process of exchanging the A and the B columns between neighboring columns of PEs to create subsystems and then moving the A columns within the subsystems. The entire algorithm is very similar to its 1-D counterpart and is specified in 2-D-SVD:

```
    procedure 2-D-SVD
1   While not (all |γ|<ε) do
2   2-D-ORTH;
3   for k := 0 step 1 until g−1 do
4   for p := 1 step 1 until 2^(g−k)−1 do
5       h := f(g−k, p);
6       R_rc(i, j)←1+i+i^(h);
7     RowBust(i)←A(i, j);
8     A(i, j)←RowBus(i);
9     2-D-ORTH;
10  end for p;
11  if r_k = 0 then
12      R_rc(i,j)<−1+i+i^(h);
13    RowBus(i)←B(i, j);
14    A(i, j)←RowBus(i);
15  else
16      R_rc(i,j)←1+i+i^(h);
17    RowBus(i)←A(i, j);
18    B(i, j)←RowBus(i);
19  2-D-ORTH;
20  end for k;
21  end while;
```

In the 2-D-SVD algorithm, $A(i, j)$ and $B(i, j)$ denote memory locations within processor (i, j). RowBus(i) is the optical bus at row i in the array. All the other symbols have the same meaning as described in the 1-D-SVD algorithm. The algorithm is similar to the 1-D-SVD except that a column pair is mapped onto a column of processors in the 2-D-SVD instead of onto a single processor. The time spent in procedure 2-D-ORTH is $O(\log m)$ as analysed before. Most time of the algorithm is spent within the double loop (lines 5 to 9). Since lines 5 and 6 require a constant time and each data transfer in lines 7 and 8 takes one bus cycle, the time is dominated by line 9. Similar to the analysis of the 1-D-SVD algorithm, the number of times line 9 is executed is at most n. Therefore the total time of 2-D-SVD is $O(n \log m)$, which is on the same order of magnitude as that on a hypercube of the same size [14]. While a PE in a 2-D array with a pipelined optical bus has only four communication ports, a PE in a hypercube of size $mn/2$ has $\log(mn/2)$ ports.

5. Large SVD problems

Finally, let us discuss the time complexity of the parallel SVD algorithm on an array where the number of processors is much smaller than the size of the problem. Suppose we have a 2-D $v \times w$ array with optical pipelined buses. Without loss of generality, suppose $v \leqslant m$ and $w \leqslant n$. Also assume that m and n are multiples of v and w, respectively. We partition the m-by-n matrix A into h groups of w columns each: $C_1, C_2, \ldots, C_h$, as shown in Fig. 3 for $h = 3$. In this example, all column pairs which are orthogonalized in a sweep can be obtained by pairing the columns in the following way: (C_1, C_1), (C_1, C_2), (C_1, C_3), (C_2, C_2), (C_2, C_3). Therefore, the whole computation consists of processing

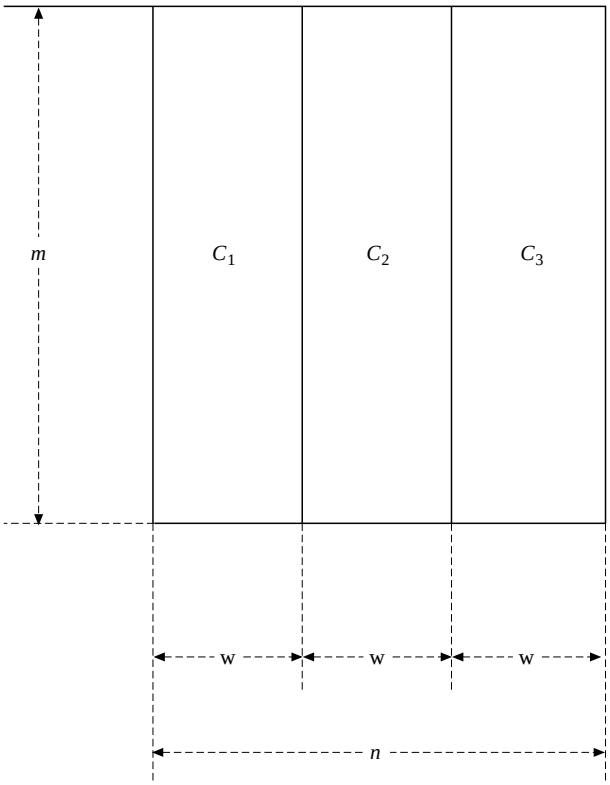

Figure 3. Column groups in a large matrix.

all pairs of column groups by algorithm 2-D-SVD. To process a pair of column groups, m/v matrix columns are stored in each column of processors with m/v pairs of matrix elements stored in the local memory of each processor. $O(w(\log v + m/v))$ time is required to process a pair of column groups because each processor has to perform m/v multiplications and additions locally, and the summation of partial sums and the broadcast of parameters within a column of processors each requires $O(\log v)$ time. The number of pairs of column groups is

$$h + (h-1) + (h-2) + \cdots + 2 + 1 = h(h+1)/2 \tag{14}$$

where $h = n/w$. Thus, the total time required to compute a sweep of SVD for an $m \times n$ matrix is $O((h(h+1)/2)w(\log v + m/v)) = O(nh(\log v + m/v))$ on a 2-D $v \times w$ array with optical pipelined buses. For large SVD problems on a 1-D array with optical pipelined buses, similar analysis can be applied. Assume that a 1-D array with optical pipelined buses has w processors, $h(h+1)/2$ pairs of column groups need to be processed, where $h = n/w$. To process a pair of column groups, each processor has to process w column pairs. The time to process each column pair is $O(m)$. Hence, the time required to process a sweep of SVD for an $m \times n$ matrix is $O((h(h+1)/2)mw)$ which is (hmn).

6. Conclusions

Parallel computers using static networks such as the mesh and hypercube have the problem of large diameters due to their limited connectivities. The time complexities of parallel algorithms on these networks are lower bounded by their diameters. Adding buses to these networks is one way to solve the problem. However, messages cannot be transmitted concurrently on an electronic bus. A pipelined optical bus, on the other hand, not only has a shorter end-to-end propagation delay, but also can be accessed concurrently by messages. It has been established that much higher bandwidth can be achieved with pipelined optical bus communication than with exclusive electronic bus access communication [21]. Thus, many parallel algorithms can be implemented on an array with a pipelined optical bus efficiently. In this paper, we have shown that the time complexities used in the algorithms on arrays with optical interconnects are asymptotically equivalent to those implemented on the hypercube while they use substantially less hardware.

In this paper, we have concentrated on algorithm design on the optical bus system. During the description of the singular value decomposition algorithms and the optical bus system, many assumptions from Qiao and Melhem [23] are used. Many technical details such as clock distribution, frame synchronization, electronically controlled optical switching, conversion between optical and electronic signals, and packet size limitation are either discussed briefly or completely omitted. Interested readers may refer to references [15,17,21,23,25] for details.

Recently, the authors have proposed another new computational model called Linear Array with a Reconfigurable Pipelined Bus System (LARPBS) based on optical waveguides [26]. In this model, messages can be transmitted in a pipelined fashion on an optical bus system and the bus can be dynamically reconfigured into independent segments to satisfy different communication requirements during a computation. Using conditional optical delays, many basic operations such as binary prefix summation, split, compression, and multiple broadcasting can be done in one bus cycle [26]. It is clear that LARPBS is much more powerful than the model used in this paper although only several switches are added to a pipelined optical bus system. We believe that the SVD problem can be solved more efficiently on the LARPBS and we are doing research in this direction.

References

1. G. H. Golub and C. Reisch 1971. Singular value decomposition and least squares solutions. In *Handbook for Automatic Computation, Vol. 2 (Linear Algebra)* (J. H. Wilkinson and C. Reinsch, eds), 134–151. New York: Springer-Verlag.
2. G. H. Golub, W. Kahan and F. T. Luk 1980. Computing the singular-value decomposition on the ILLIAC-IV. *ACM Transactions on Mathematical Software*, **6**, 524–539.
3. R. P. Brent and F. T. Luk 1985. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM Journal of Scientific and Statistical Computation*, **6**, 69–84.
4. R. P. Brent, F. T. Luk and C. F. Van Loan 1985. Computation of the singular value decomposition using mesh-connected processors. *Journal of VLSI Computer Systems*, **1**, 243–270.
5. Henry Y. H. Chuang and L. Chen 1989. Efficient computation of the singular value decomposition on cube connected SIMD machine. *Supercomputing '89*, 276–282, Reno, November.

6. A. M. Finn, F. T. Luk and C. Pottle 1982. Systolic array computation of the singular value decomposition. *Proceedings of the SPIE Symposium*, **341**, *Real Time Signal Processing V*, 35–43.
7. D. E. Heller and I. C. F. Ispen 1982. Systolic networks for orthogonal equivalence transformations and their applications. *Proceedings 1982 Advanced Research in VSLI, MIT, Cambridge*, pp. 113–1convolution22.
8. F. T. Luk 1984. A triangular processor array for computing the singular value decomposition. TR84-625, Computer Science Department, Cornell University.
9. R. Schreiber 1983. A systolic architecture for singular value decomposition. *Proceedings 1st International Colloquium on Vector and Parallel Computing in Scientific Applications*, Paris, 143–148.
10. M. R. Hestenes 1958. Inversion of matrices by biorthogonalization of related results. *Journal of Social and Industrial Applied Mathematics*, **6**, 51–90.
11. P. J. Eberlein 1987. On using the Jacobi method on the hypercube. *SIAM Proceedings of the Second Conference on Hypercube Multiprocessors*, 605–611.
12. C. Bischof 1987. The two-sided Jacobi method on a hypercube. *SIAM Proceedings of the Second Conference on Hypercube Multiprocessors*, 612–618.
13. G. R. Gao and S. J. Thomas 1988. An optimal parallel Jacobi-like solution method for the singular value decomposition. *Proceedings of the 1988 International Conference on Parallel Processing*, 47–53, August.
14. Yi Pan and Henry H. Y. Chuang 1992. Highly parallel algorithms for singular value decomposition on hypercube SIMD machines. *Computers and Mathematics with Applications*, **24**, 23–32.
15. C. Qiao, R. Melhem, D. Chiarulli and S. Levitan 1991. Optical multicasting in linear arrays. *International Journal of Optical Computing*, **2**, 31–48.
16. Z. Guo 1992. Sorting on array processors with pipelined buses. *1992 International Conference on Parallel Processing*, 289–292, St. Charles, IL, August 17–21.
17. Y. Pan 1994. Order statistics on optically interconnected multiprocessor systems. *The First International Workshop on Massively Parallel Processing Using Optical Interconnections*, 162–169, Cancun, Mexico, April 26–27. (Also in the Special Issue on Optical Computing, Optics and Laser Technology, **26**, 281–287, 1994.)
18. M. Hamdi and Y. Pan 1995. Efficient parallel algorithms for some numerical problems on arrays with pipelined optical buses. *IEE Proceedings – Computers and Digital Techniques*, **142**, 87–92, March.
19. Y. Pan 1992. Hough transform on arrays with an optical bus. *Fifth International Conference on Parallel and Distributed Computing and Systems*, 161–166, Pittsburgh, PA, October 1–3.
20. H. Rutishauser 1971. The Jacobi method for real symmetric matrices. In *Handbook for Automatic Computation, Vol. 2* (C. Reinsch, ed.), 202–211. Berlin: Springer-Verlag.
21. Z. Guo, R. Melhem, R. Hall, D. Chiarulli and S. Levitan 1991. Array processors with pipelined optical buses. *Journal of Parallel and Distributed Computing*, **12**, 269–282.
22. R. Melhem, D. Chiarulli and S. Levitan 1989. Space multiplexing of waveguides in optically interconnected multiprocessor systems. *Computer Journal*, **32**, 362–369.
23. C. Qiao and R. Melhem 1993. Time-division optical communications in multiprocessor arrays. *IEEE Transactions on Computers*, **42**, 577–590, May.
24. E. Dekel, D. Nassimi and S. Sahni 1981. Parallel matrix and graph algorithms. *SIAM Journal of Computing*, **10**, 657–675.
25. D. Chiarulli, R. Melhem and S. Levitan 1987. Using coincident optical pulses for parallel memory addressing. *IEEE Computer*, **20**, 48–58.
26. Yi Pan and Mounir Hamdi 1994. Quicksort on a linear array with a reconfigurable pipelined bus system. Working Paper 94-03, Center for Business and Economics Research, University of Dayton.

Yi Pan was born in Jiangsu, China, on May 12, 1960. He entered Tsinghua University in March 1978 with the highest college entrance examination score of all 1977 high school graduates in Jiangsu Province. He received his BEng degree in computer engineering from Tsinghua University, China, in 1982, and his PhD degree in computer science from the University of Pittsburgh, USA, in 1991.

Since 1991, he has been an assistant professor in the Department of Computer Science at the University of Dayton, Ohio, USA. His research interests include parallel algorithms and architectures, distributed computing, task scheduling, and networking. He has published 12 papers in refereed international journals and many papers in various international conferences. He has received several awards including Research Opportunity Award from the US National Science Foundation, Andrew Mellon Fellowship from the Mellon Foundation, and Summer Research Fellowship from the Research Council of the University of Dayton. His research has been supported by the US National Science Foundation and US Air Force.

Dr Pan is a member of the IEEE Computer Society and the Association for Computing Machinery.

Dr Hamdi received the BSc degree with distinction in Electrical Engineering from the University of Southwestern Louisiana in 1985, and MSc and PhD degrees in Electrical Engineering from the University of Pittsburgh in 1987 and 1991, respectively. While at the University of Pittsburgh, he was a Research Fellow in various research projects on interconnection networks, high-speed communication, parallel algorithms, and computer vision. In 1991 he joined the Computer Science Department at the Hong Kong University of Science and Technology as an Assistant Professor. His main areas of research are parallel computing, high-speed networks, and ATM packet switching architectures. Dr Hamdi has published over 40 papers on these areas in various journals and international conferences.