Efficient parallel algorithms on optically interconnected arrays of processors

M. Hamdi
Y. Pan

Indexing terms: Array processing, Parallel algorithms, Pipeline communication

Abstract: Arrays of processors with pipelined optical buses are introduced for the efficient implementation of computationally intensive applications. Techniques for the concurrent transmission of messages over the optical bus to avoid collision of messages is shown. Convenient parallel data movement operations are derived for this architecture, which are then used in the design of parallel algorithms for the solution of some important numerical problems. The parallel algorithms implemented in the paper are for solving systems of linear equations and finding the roots of nonlinear equations. Even though this array of processors can function in the MIMD mode of operation, it is more suitable for the SIMD mode of operation, because it can be easily synchronised and scaled to a massive number of processors. Hence, the above parallel algorithms have been designed with the SIMD mode in mind. Their time complexities have been analysed, and are shown to compare favourably with those implemented on processors connected with electronic buses or point-to-point links such as the hypercube. Moreover, whereas a processing element of a hypercube of size N has $\log N$ ports, a processing element of an array with optical buses has a constant number of ports. Thus, it seems that an array of processors with optical buses is a promising, and could be a better, alternative for future supercomputing systems.

1 Introduction

Distributed-memory multicomputer systems hold an advantage over shared-memory multiprocessor systems when it comes to massive parallelism as they can be easily scaled up to a large number of processors. Most of the distributed-memory multicomputer systems use a static interconnection network (e.g. mesh or a hypercube) for the processors to communicate with each other [4]. One deficiency with these networks is that their communication diameter depends on the number of processors (size) of the system [17]. Hence, increasing the size of these networks would not result in a further decrease in the time complexities of most parallel algorithms (e.g. semi-group computations), which are lower-bounded by the communication diameter of these networks. One way to overcome this problem is to use buses for communication as they provide direct communication between any two processors in the system [8, 11]. However, messages cannot be transmitted concurrently by different processors on such buses, and thus these buses become a major bottleneck in the system, especially in communication-intensive algorithms. Another way to address the problem is to use reconfigurable buses [13], where messages can be transmitted concurrently when the bus is partitioned onto many segments; this also solves the diameter problem when all bus segments are reconfigured as a single global bus. However, when there are a large number of data to be transferred between different sections of the network, the bus segments themselves become a potential bottleneck for such a system [13].

To alleviate the above problems, researchers have proposed a distributed-memory computer with pipelined optical buses [14]. In such a system, messages can be transmitted concurrently by different processors, in a pipelined fashion on the optical bus, without partitioning the bus. This is mainly due to the unique properties of optical buses (waveguides), such as the unidirectional propagation of signals and the precise predictable path delays per unit distance. This, in turn, makes the pipelining of optical signals by the synchronised directional coupling of each signal at a specified location along the waveguide quite feasible. The time delay between the farthest processors along the waveguide is just the end-to-end transmission delay of light over the optical waveguide. This architecture has received a lot of attention in the research community as it integrates the advantages of both optical transmission and electronic computation [3, 7, 9, 12, 18–20]. Thus, the above properties and the efficiency of bus structures to perform broadcasting and multicasting make this architecture suitable for quite a wide range of applications, especially communication-intensive applications.

In this paper, we use a synchronous array of processing elements (PEs), which employs an optical bus for

© IEE, 1995

Paper 1621E (C1, E10, E13), first received 12th April and in revised form 7th October 1994

M. Hamdi is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Y. Pan is with the Department of Computer Science, University of Dayton, Dayton, OH 45469-2160, USA

This research was supported in part by the Hong Kong Research Grant Council under grant RGC/HKUST 100/92.

the transmission of messages between PEs to implement some important mathematical algorithms. We review the basic principles of message pipelining on an optical bus and we introduce several fundamental parallel data movement operations on arrays of processors with pipelined optical buses, which we then incorporate into parallel algorithms to solve several computationally intensive numerical problems. Because of the unique features of a pipelined optical bus, many techniques are developed for the scheduling of the transmission and the reception of messages over the bus. Then, we present efficient parallel algorithms for the solution of important numerical problems, such as solving systems of linear equations and finding the roots of nonlinear equations. Finally, we demonstrate that the time complexities of these algorithms compare favourably with those implemented on arrays with traditional electronic buses or point-to-point links.

2 Array of processors with waveguides

A unique property of optics is its ability to pipeline the transmission of signals through a channel. In electronic buses, signals propagate in both directions, whereas optical channels are inherently directional and have precise predictable path delays per unit distance [3, 7, 9, 14, 20]. Consider the system of Fig. 1, where n pro-

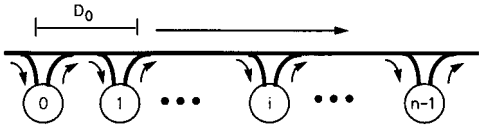

Fig. 1 *System of n processors connected with a single optical waveguide*

cessors, each having a constant number of registers, are connected through a single optical waveguide (bus). Each processor is coupled to the optical waveguide with two passive couplers, one for injecting (writing) signals on the waveguide and the other for receiving (reading) signals from the waveguide. As in the case of electronic buses, each processor j communicates with any other processor i by sending a message to i through the common bus. However, because optical signals propagate in one direction, a processor j may send signals to another processor i only if $i > j$.

Assume that a message on an optical bus consists of a sequence of pulses, each having a width w in seconds. The existence of an optical signal of width w represents a binary bit 1, and the absence of such a signal represents a 0. For analytical reasons, we let D_0 be the optical distance between each pair of adjacent nodes and τ be the time taken for an optical pulse to traverse the optical distance D_0. To transfer a message from a node j to node i, $i > j$, the sender j writes its message on the bus. After a time $(i-j)\tau$, the message will arrive at the receiver i, which then reads the message from the bus.

The properties of undirectional propagation and predictable path delays of optical signals can be used advantageously. Specifically, unlike the electronic case, where the writing access to the bus by each node must be mutually exclusive, all nodes in the system of Fig. 1 can write on the bus simultaneously, provided the following collision-free condition is satisfied [7, 9, 14, 20]:

$$D_0 > bwc_g \tag{1}$$

where b is the number of binary bits in each message, and c_g is the velocity of light in the waveguide. Clearly, if this condition is satisfied and the system is synchronised such that every node starts writing a message on the bus at the same instant, then no two messages injected onto the bus by any two distinct nodes will collide. Here, by colliding we mean that two optical signals injected on the bus by any two distinct nodes arrive at some point on the bus simultaneously. With this condition satisfied, every node can, in parallel, send a message to some other node, and the messages will travel from left to right on the bus in pipelined fashion, as shown in Fig. 2. Thus, we use the

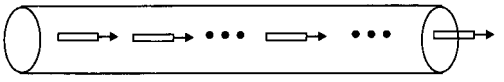

Fig. 2 *Message pipelining on the optical bus*

term pipelined bus. In the rest of the paper, we always assume that the collision-free condition eqn. 1 is satisfied. We term τ as *petit* cycle, and $n\tau$ as a bus cycle. Note that a bus cycle is the time taken for an optical signal to traverse the entire length of the optical bus.

In the system of Fig. 1, messages can be transmitted only from left to right. To allow message-passing from right to left, another optical bus is used, as shown in Fig. 3. In this Figure, we have two optical buses: the upper

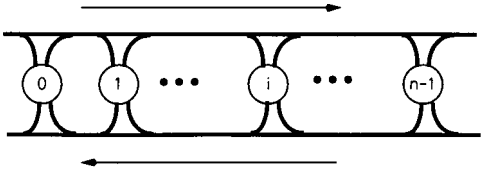

Fig. 3 *1-D APW with two optical waveguides*

one is used for sending messages from left to right, and the lower one is used for sending messages from right to left. Each node can write and read messages on either bus as desired. Obviously, signals in different buses do not disturb one another: that is the two buses can support two separate pipelines. The system in Fig. 3 is the architecture of a linear array of processors with waveguides or a 1-D APW.

To specify the time at which a node should receive a messsage, we introduce a function $t_{wait}(i)$, which is defined as the time that node i should wait, relative to the beginning of a bus cycle, before reading the message sent to it from some other node j. Thus

$$t_{wait}(i) = (i-j)\tau$$

If τ is considered as a time unit, then t_{wait} can be interpreted in terms of the number of such units and thus be written $t_{wait}(i) = i - j$. Clearly, if $t_{wait}(i) > 0$, then the message is to be received from the left; if $t_{wait}(i) < 0$, then the message is to be received from the right. If $t_{wait}(i) = 0$, then no message should be received by node i. The value of $t_{wait}(i)$ can be stored in a wait register, and more than one such register can be used if a node is to receive more than one message in one bus cycle.

This t_{wait} control function, however, has the disadvantages that it depends crucially on timing accuracy and is sensitive to the optical distance D_0 between two adjacent nodes. An equivalent control function m_{wait} that does not

have these disadvantages can be defined if we require that each node inject a message, real or dummy, every bus cycle. In this case, we define $m_{wait}(i)$ as the number of messages that node i should skip before reading its message. For example, if $m_{wait}(i) = \gamma$, then node i should receive the $|\gamma|$th message that passes i on the bus. That is, it has to wait until $|\gamma| - 1$ messages have passed and then it reads its own message. The sign of γ determines on which bus the message should be received. Clearly, m_{wait} is equivalent to t_{wait}, and either control function can be used. For convenience, we simply write the control function as *wait*.

The control function *wait* can only be used when the communication pattern is known to the receiver, in the sense that the receiver knows from which node the message is to be received. This is the case in most SIMD programming environments [7, 20, 22]. In cases where the communication pattern is unknown to the receiver, the coincident pulse techniques [3, 12] can be used, such that an addressing pulse and a reference pulse coincide at the detector of the receiver, thereby addressing it. In this paper we use *wait* for addressing as the communication patterns are assumed to be known to the receiver.

2.1 Message routing in a linear APW

Various message-routing patterns can be realised in a simple, straightforward way. As a routing pattern is determined by the *wait* functions, we need only determine these *wait* functions for each routing pattern. The most common patterns are

(*a*) one-to-one: The system executes a *SEND*(j, i) instruction, which means that a message is to be transferred from node j to node i. Thus, $wait(i) = i - j$, where i is a single specific node.

(*b*) broadcast: The system executes *BROADCAST*(j), which means that node j broadcasts a message, and all other nodes i will receive that message. In this case, $wait(i) = i - j$ for all $i \neq j$.

(*c*) permutations: For each node j to send a message to node $i = PERM(j)$, where *PERM*( ) is an arbitrary permutation, we set $wait(i) = i - j$ for all i.

We see that the computation of $wait(i)$ is very simple and uniform. The only difference between the *wait* functions for different message-routing patterns is that the nodes involved are different. It is clear that all these communication tasks can be performed using a single bus cycle. Note that, in a linear APW, message passing between two non-neighbouring nodes is nearly as efficient as that between two neighbours. Specifically, a message takes τ more time to pass one more node on the optical bus. This is not the case in typical parallel computers with point-to-point connections, where to pass a node, en route to another node, a message has to go through a router. Moreover, τ can be extremely small for tightly coupled parallel computer systems because of the small distance between neighbouring nodes and the high speed of optics.

3 Two-dimensional APW

The linear APW described above has some practical limitations [7–9, 20]. First, the system size cannot be very large as it is limited by the minimum optical power that can be detected by the directional coupler of each PE, which is inversely proportional to the number of optical path lengths an optical pulse would traverse. Secondly, when the number of processors N is very large, the end-to-end transmission delay (waveguide cycle) increases, as the condition stated by eqn. 1 has to be satisfied and the number of *petit* cycles is proportional to N. To overcome these shortcomings in a linear APW, we consider a two-dimensional (2-D) APW. In a 2-D APW, each node is coupled to four buses, as shown in Fig. 4,

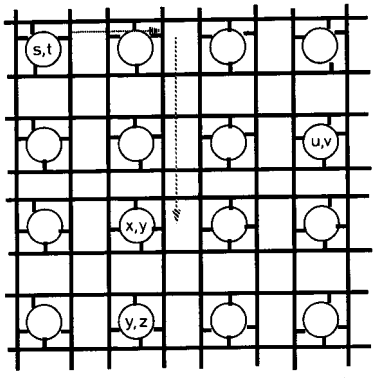

Fig. 4 *2-D APW where each processor is connected to four waveguides*

where the two horizontal buses are used for passing messages horizontally in the same way as before, and the vertical buses are used for passing messages vertically in a similar way. Each node in a 2-D APW of size $N = m \times n$ will be given an identification (x, y), $0 \leq x < m$, $0 \leq y < n$, indicating the row–column position of the node.

A unique issue that arises in the 2-D APW is the relay of messages. Specifically, suppose a message is to be transferred from node (x_1, y_1) to node (x_2, y_2), with $x_1 \neq x_2$ and $y_1 \neq y_2$. Then, the message can first be sent from (x_1, y_1) to (x_1, y_2), which is the node at the intersection of row x_1 and column y_2, in the first bus cycle (a row bus cycle), and then from (x_1, y_2) to (x_2, y_2) in the second bus cycle (a column bus cycle). That is, the message has to be buffered at node (x_1, y_2) at the end of the first bus cycle and then relayed to its destination in the second bus cycle. For the purpose of relaying messages, we define a control function *relay* for node (x_1, y_2) as

$$relay[(x_1, y_2)] = y_2 - y_1$$

which indicates that node (x_1, y_2) will read a message from a row bus at time $|y_2 - y_1|$ (relative to the start of the row bus cycle) and then write that message on the proper column bus at the beginning of the following column bus cycle.

Clearly, all the routing patterns described in the preceding Section for the 1-D APW (one-to-one, broadcast, permutations) can be similarly implemented on the 2-D APW using the *relay* function. For more details, the reader is referred to References 7, 18 and 20. However, they are omitted in this paper because they are not directly needed for the implementation of our algorithms. Rather, as will be shown in the following Section, the routing patterns presented for the 1-D APW are sufficient to implement our algorithms on the 2-D APW.

4 Parallel algorithms for mathematical problems

Problems arising in any scientific or engineering application of computers usually require the solution of some

mathematical problems. These mathematical problems span a wide range of applications and frequently involve the solution of some numerical problems that potentially could involve very complex systems of linear or nonlinear equations [1, 21]. Because of the large number of equations to be solved or the complexity of a single equation, these problems can be computationally intensive, and their solution could take an unreasonable amount of time on single-processor systems [1]. In this Section, we illustrate the fruitful utilisation of arrays of PEs with pipelined optical buses (e.g. 2-D APW) to provide fast and efficient solutions to some important numerical problems, namely the solution of systems of linear equations and finding the roots of nonlinear equations. Our algorithms are tuned towards fine-grain computations, where each PE holds just one element of the input. This is reasonable for massively parallel SIMD machines (e.g. CM-2). The largest number of PEs that can be connected to an optical bus, under current technology, is in the order of a few hundred PEs [3]. Thus, the size of our 2-D APW can be quite large (e.g. 10000 PEs). In case the problem size is larger than the number of PEs, the PEs have to perform computations on a number of elements sequentially (subproblems). The solutions to these subproblems are merged together to obtain the solution to the whole problem. Hence, our algorithms' design would be slightly different from those given in this paper, and this might be a good direction for further research.

4.1 Solving systems of linear equations

Given an $n \times n$ matrix A and an $n \times 1$ vector B, we are to solve the equation $AX = B$ for the unknown $n \times 1$ vector X. When $n = 3$, as an example, we are required to find the values for x_1, x_2 and x_3 from the following system of linear equations:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

Our parallel algorithm for the solution of the above problem on a 2-D array of PEs with pipelined optical buses is essentially a parallelisation of the famous sequential algorithm based on the Gauss–Jordan method [1, 21]. This method uses *eqn.* 1 of the system of linear equations to eliminate x_1 from all other equations through scaling and subtraction. Then, it uses *eqn.* 2 to eliminate x_2 from all other equations through scaling and subtraction. In general, it uses *eqn.* i to eliminate x_i from all other equations. After that is done, our system of linear equations will have the following form:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

If we denote $b_i = a_{i,n+1}$, then to eliminate x_1 from all equations using *eqn.* 1 we have to perform the following transformation on the elements of matrix A:

$$a_{ik} = a_{ik} - a_{i1}a_{ik}/a_{11} \quad 1 \leqslant i \leqslant n \quad 1 \leqslant k \leqslant n+1 \tag{3}$$

In general, to eliminate x_i from all equations using *eqn.* i, we have to perform the following transformations on the elements of matrix A:

$$a_{ik} = a_{ik} - a_{ij}a_{jk}/a_{jj} \quad 1 \leqslant i \leqslant n \quad j \leqslant k \leqslant n+1 \tag{4}$$

Then, once we have performed all the transformations, x_i can be easily solved using the following equation:

$$x_i = a_{i,n+1}/a_{ii} \tag{5}$$

The above method is used on a 2-D APW with $n^2 + n$ PEs that can be thought of as arranged in an $n \times (n+1)$ array. That is, each row will contain $n+1$ PEs, and each column will contain n PEs. Thus, each PE contains one element of the matrix A, and the last column of PEs contains the elements of the vector B. Our parallel algorithm performs steps 1–4 given below n times, where, in each iteration, it succeeds in the elimination of one variable from $n-1$ equations, and performs step 5 just once:

Step 1: The PE holding the element a_{jj} sends this value to all other PEs on the same column. This is basically a broadcasting instruction along a column of PEs. Then, each processor on that column performs the following operation a_{ij}/a_{jj} (keeping in mind that we are trying to perform the operation given by eqn. 4 above on each PE).

Step 2: Each PE holding the value a_{ij}/a_{jj} already calculated sends this value to all the PEs in the same row. This is just a broadcasting instruction along the rows.

Step 3: Each PE holding the element a_{jk} broadcasts that element to all the PEs in the same column.

Step 4: All the PEs now perform the operation of eqn. 4, as each PE now contains all the elements required by eqn. 4.

Step 5: Each PE in the last column holding the value $a_{i,n+1}$ sends that value to the PE in the same row holding the value a_{ii}. These latter PEs perform the operation required by eqn. 5 and, hence, the solution of the linear systems of equations.

By looking at the steps of the algorithm, we see that steps 1–4 are for the implementation of eqn. 4, and step 5 is for the implementation of eqn. 5. Then, analysing the time complexity of the above algorithm, we see that each of the steps 1–4 consists of broadcasting along the column or along the rows, which takes just one bus cycle. The computations involved in these steps take $O(1)$ time. Thus, steps 1–4 take $O(n)$ time as they have to be executed n times. Step 5 takes just one routing step and one computation step, thus, it takes $O(1)$ time. Hence, the whole algorithm has $O(n)$ time complexity.

4.2 Finding the roots of nonlinear equations

In many science and engineering problems, we are frequently asked to find the roots of a nonlinear equation with one variable, such as the following examples:

$$x^7 - e^x + \log x = 0$$

$$x^{-2} + \sin x + x^3 - 153 = 0$$

To solve these types of equation, we have to resort to numerical algorithms, as, generally, it is impossible to find the solution analytically. A very simple and standard sequential numerical algorithm for this problem is the bisection method [1, 21]. The idea is to represent our nonlinear equation by a continuous function $f(x)$. Then, if a_0 and b_0 are two values of the variable x, in such a way that $f(a_0)f(b_0) < 0$, that is, $f(a_0)$ and $f(b_0)$ have opposite signs, a zero of the function f is guaranteed to exist in the interval (a_0, b_0). This means that one root of our nonlinear equation is guaranteed to exist in the interval (a_0, b_0). Once we have found the interval (a_0, b_0), we bisect it in the middle point $m_0 = 1/2(a_0 + b_0)$. Now, if $f(a_0)f(m_0) <$

0, then the root must be in the interval $(a_1, b_1) = (a_0, m_0)$; otherwise it is in the interval $(a_1, b_1) = (m_0, b_0)$. Then, the above process is repeated on the interval (a_1, b_1). Thus, we are trying to make the interval containing the root smaller and smaller until we obtain a very close approximation of it after n steps, that is, $|b_n - a_n| < \varepsilon$, where ε is a very small positive number chosen depending on the accuracy desired.

We can use the bisection method on a 2-D APW using N processors. The idea is to conduct an $(N + 1)$-section search on the 2-D APW. The initial interval known to contain a zero of a function f, (a_0, b_0), is divided into $N + 1$ subintervals of equal length. Then, each PE implements the sequential bisection algorithm on its subinterval. However, only one PE will succeed in finding the interval containing the root. Morever, that chosen subinterval will be further subdivided into $(N + 1)$ subsubintervals. This process, as with the sequential algorithm, continues until we obtain a very close approximation of the root. The parallel algorithm implemented on a 2-D APW performs the following steps as long as the width of the interval containing the root is $> \varepsilon$:

(*a*) Bisect the interval into $N + 1$ equal subintervals. This is achieved by dividing the interval by $N + 1$ and is done simultaneously by all PEs (duplicated).

(*b*) Each PE assigns a single subinterval to itself, e.g. PE i is assigned subinterval i. Only one subinterval is not assigned to any PE.

(*c*) Each PE decides whether its subinterval contains a root or not by checking the sign of the multiplication of its boundary values, e.g. $f(x_1)f(x_2)$ where x_1 and x_2 are the boundary values of the subinterval. Note, only one subinterval contains a root.

(*d*) The PE containing the subinterval containing the root checks if the width of that interval is $< \varepsilon$. If that is true, then we have finished, as the root is equal to the value of any of the two boundaries of that interval. If the width of that interval is $> \varepsilon$, then that PE broadcasts the values of the boundaries of its interval to all the PEs, and steps *a–d* are performed again.

(*e*) If none of the subintervals in the PEs contains a root, then the unassigned subinterval must contain the root. Then, the boundary values of this subinterval are broadcasted to all the PEs to perform steps *a–d*.

All the above steps involve either $O(1)$ computation steps or broadcasting, which takes two bus cycles, as was shown in the data movement operations. Thus, each of the above steps takes $O(1)$ time. The number of iterations that steps *a–e* have to go through can be found as follows. After j iterations, the width of the interval is $w/(N + 1)^j$, where w is the width of the initial interval, that is, $w = |b_0 - a_0|$. The iterative process terminates as soon as $w/(N + 1)^j < \varepsilon$. Thus, the number of iterations is $O(\log_{N+1}(w))$. As steps *a–e* take constant time, then the above algorithm takes $O(\log_{N+1}(w))$ time. The most consuming parts of the algorithms are the function evaluation and the broadcasting operation, which are efficiently executed on such an architecture.

5 Comparison with electronic parallel architectures

In the preceding Section, the computational power of the 2-D APW was characterised by implementing two parallel algorithms for two important numerical problems. Here, the 2-D APW is compared with other parallel architectures, namely the hypercube [1, 10], the 2-D array of PEs with electronic buses [2, 16] and the concurrent read concurrent write (CRCW) ideal model of parallel computations [6].

The communication complexity of our algorithms is a function of the number of optical bus cycles. This is consistent with the analysis carried out by all researchers working on architectures having electronic buses or reconfigurable electronic buses [2, 13]. The communication complexity of their algorithms is a function of the electronic bus cycles. Further, owing to the high bandwidth of waveguides, an optical bus cycle is typically smaller than an electronic bus cycle when the two buses are equal in physical size. Hence, in our comparison with the 2-D array of PEs with electronic buses, we assume that an optical bus cycle is equivalent to an electronic bus cycle, which is fair and reasonable. Consequently, our algorithm analysis is consistent with the analysis performed on electronic buses, which is directly related to the number of bus cycles. Even though our bus cycle might be a multiple number of pipeline (*petit*) cycles, we prefer to analyse our algorithms using bus cycles to be consistent with the analysis carried out on parallel architectures using electronic buses. Hence, in this case, a fair comparison between using optical buses and electronic buses can be made. For more details, please see References 2, 11, 13 and 16.

On the other hand, for the hypercube, the communication complexity of its algorithms is a function of the number of point-to-point link communications. However, these point-to-point link communications are not constant, as their length is not constant [8, 13]. That is, not all the neighbours of a PE are at an equal distance from it, because of the layout properties of the hypercube. However, researchers into parallel algorithm design assume that the point-to-point communication time on the hypercube takes one cycle time, regardless of the length of the link being used. Thus, the cycle time of a point-to-point communication in a hypercube may or may not be larger than an optical bus cycle time; it is hardware and software dependent. Consequently, when comparing our algorithms with those on the hypercube, we leave the time complexity as a function of point-to-point communication time for the hypercube and of optical bus cycle time for the 2-D APW.

The time complexity of the parallel algorithm for the solution of systems of linear equations is $O(N \log(N))$ point-to-point communication cycle times when implemented on a hypercube of size $O(N^2)$ [1, 10]. Thus, the 2-D APW is more time efficient in solving this problem than the hypercube, as long as an optical bus cycle time is less than or equal to $O(\log(N)) \times$ the point-to-point communication cycle time of the hypercube. Hence, even though a hypercube of the same size requires much more hardware and thus is much more expensive because it has many more ports per PE, e.g. $2 \log(N)$ against 4, a 2-D APW has the potential of being more effective owing to its messages' pipelining properties.

If our algorithm for the solution of systems of linear equations is implemented on a 2-D array with electronic buses, it takes $O(N^{7/6})$ time [16]. Hence, even though the hardware complexity of both architectures is the same, the effectiveness of optical buses can be clearly seen through the comparison of both time complexities, assuming that an optical bus cycle is equivalent to an electronic bus cycle. Again, the higher efficiency of a 2-D APW is due to its non-exclusive access to the optical buses. Finally, the solution of systems of linear equations takes $O(N)$ on a CRCW model with $O(N^2)$ PEs [1, 23].

The CRCW model is an idealised parallel machine with shared memory, where it takes only one communication step to access any memory location. Moreover, simultaneous read and simultaneous write to or from any memory location is allowed. This makes CRCW the most powerful model of parallel computations. However, we see that the time complexity of CRCW and that of the 2-D APW are the same for this specific algorithm, if we assume that an optical bus cycle is equivalent to the access time of shared memory in the CRCW model. This is simply a confirmation of the effectiveness of this architecture.

For the second algorithm that implements the bisection method to find the roots of nonlinear equations, it takes $O(\log_N(w) \log(N))$ on a hypercube of size N [1]. Thus, the 2-D APW is again more time and hardware efficient for this algorithm than the hypercube, as long as an optical bus cycle time is less than or equal to $O(\log(N)) \times$ the point-to-point communication cycle time of the hypercube. This same algorithm takes $O(N^{1/6} \log_N(w))$ on a 2-D array of PEs with electronic buses [16], which is much slower than that on the 2-D APW having equivalent bus cycle times. Finally, when this algorithm is implemented on a CRCW model [1, 5, 15], it has the same time complexity as that on the 2-D APW, when an optical bus cycle time is equivalent to the access time of the shared memory of the CRCW model. Hence, this comparison leaves no doubt about the good potential and usefulness of the 2-D array of PEs with optical buses in the execution of computationally intensive algorithms and how favourable it can be when compared with other related architectures. Further, it has also been compared favourably with many other related architectures in the implementation of other parallel algorithms in other areas [8, 18].

6 Conclusions

We have presented a parallel architecture, a 2-D array of PEs with pipelined optical buses (2-D APW), that exploits the advantages of optical communication and electronic computation. Some routing strategies for the efficient concurrent transmission of messages over the optical bus have been demonstrated. Then, we have implemented some fundamental parallel data movement operations on this architecture and analysed their time complexities. These data movement operations have been incorporated in the efficient design of parallel algorithms for the solution of important computationally intensive numerical problems. The time complexities of these parallel algorithms on the 2-D APW compare favourably with other related parallel architectures employing point-to-point links, such as the hypercube, or parallel architectures employing electronic buses, such as the 2-D array of PEs, with electronic buses. Further, the time complexity of these algorithms on the 2-D array with pipelined optical buses is the same as those implemented on the most powerful parallel computer model, CRCW. Thus, it seems that the 2-D array of PEs with pipelined optical buses is a very promising parallel supercomputing architecture for the execution of computationally intensive applications.

7 References

1 BERTSEKAS, D.P., and TSITSIKLIS, J.N.: 'Parallel and distributed computation: numerical methods' (Prentice-Hall, Englewood Cliffs, NJ, 1989)

2 CHEN, Y.C., CHEN, W.T., and SHEU, J.P.: 'Designing efficient parallel algorithms on mesh-connected computers with multiple broadcasting', *IEEE Trans.*, 1990, **PDS-1,** pp. 241–245

3 CHIARULLI, D.M., LEVITAN, S.P., and MELHEM, R.G.: 'Optical bus control for distributed multiprocessors', *J. Parallel Distrib. Comput.*, 1990, **10,** pp. 45–54

4 DUNCAN, R.A.: 'A survey of parallel computer architectures', *IEEE Comput.*, 1990, **23,** pp. 5–16

5 ERIKSEN, O., and STAUNSTRUP, J.: 'Concurrent algorithms for root searching', *Acta Informatica*, 1983, **18,** pp. 361–376

6 FORTUNE, S., and WYLLIE, J.: 'Parallel random access machines'. Proc. ACM Symp. Theory Computing, 1978, pp. 114–118

7 HAMDI, M.: 'Efficient communications in optically interconnected parallel computer systems', *Int. J. High Speed Comput.*, (submitted)

8 HAMDI, M.: 'Communication-efficient interconnection networks for parallel computations'. PhD dissertation, Department of Electrical Engineering, University of Pittsburgh, 1991

9 GUO, Z., MELHEM, R., and HALL, R.W.: 'Pipelined communications in optically interconnected arrays', *J. Parallel Distrib. Comput.*, 1991, pp. 269–282

10 JOHNSSON, S.L.: 'Communication efficient basic linear algebra computations on hypercube architectures', *J. Parallel Distrib. Comput.*, 1987, **4,** pp. 133–171

11 KUMAR, V.K.P., and RAGHAVENDRA, C.S.: 'Array processor with multiple broadcasting'. Proc. 12th Int. Symp. on Comput. Arch., 1985, pp. 2–10

12 LEVITAN, S.P., CHIARULLI, D.M., and MELHEM, R.G.: 'Coincident pulse techniques for multiprocessor interconnection structures', *Appl. Opt.*, 1990, **29,** pp. 2024–2039

13 LI, H., and STOUT, Q.F.: 'Reconfigurable massively parallel computers' (Prentice-Hall, Englewood Cliffs, NJ, 1991)

14 MELHEM, R.G., CHIARULLI, D.M., and LEVITAN, S.P.: 'Space multiplexing of waveguides in optically interconnected multiprocessor systems', *Comput. J.*, 1989, **32,** pp. 362–369

15 MENEZES, A.J., VAN, P.C., and VANSTONE, S.A.: 'Some computational aspects of root finding algorithms on parallel computers'. Int. Symp. on Symbolic & Algebraic Computation, 1989, pp. 143–151

16 MIRSA, M., and KUMAR, V.K.P.: 'Efficient implementation of numerical methods on processor arrays with broadcasting'. Technical Report, University of Southern California, IRIS 264, 1989

17 PAN, Y.: 'The block shift network: interconnection strategies for large parallel systems'. PhD dissertation, Department of Computer Science, University of Pittsburgh, 1991

18 PAN, Y., and HAMDI, M.: 'Computation of singular value decomposition on arrays with pipelined optical buses'. Proc. 1993 Symposium on Applied Computing, pp. 525–532

19 QIAO, C., MELHEM, R.G., CHIARULLI, D.M., and LEVITAN, S.P.: 'Optical multicasting in optical arrays', *Int. J. Opt. Comput.*, 1991, **2,** pp. 31–48

20 QIAO, C., and MELHEM, R.G.: 'Time-division optical communications in multiprocessor arrays', *IEEE Trans.*, 1993, **C-42,** pp. 577–590

21 SCHENDAL, U.: 'Introduction to numerical methods for parallel computers' (Ellis Horwood, Chichester, England, 1984)

22 STOUT, Q.F., and WAGER, B.: 'Intensive hypercube communication: Prearranged communication in link-bound machines', *J. Parallel Distrib. Comput.*, 1990, **10,** pp. 167–181

23 WING, O., and HUANG, J.W.: 'A computational model of parallel solution of linear equations', *IEEE Trans.*, 1980, **C-29,** pp. 632–638