All-to-All Communication on the Connection Machine CM-200

KAPIL K. MATHUR[1] AND [2]S. LENNART JOHNSSON

[1]*David Shaw & Co., New York, NY 10036; e-mail: Mathur@deshaw.com*

[2]*Computer Science, Mathematics, and Electrical Engineering, University of Houston, 4800 Calhoun Rd., Houston, TX 77204-3475; e-mail: johnsson@cs.uh.edu*

ABSTRACT

Detailed algorithms for all-to-all broadcast and reduction are given for arrays mapped by binary or binary-reflected Gray code encoding to the processing nodes of binary cube networks. Algorithms are also given for the local computation of the array indices for the communicated data, thereby reducing the demand for the communications bandwidth. For the Connection Machine system CM-200, Hamiltonian cycle-based all-to-all communication algorithms yield a performance that is a factor of 2 to 10 higher than the performance offered by algorithms based on trees, butterfly networks, or the Connection Machine router. The peak data rate achieved for all-to-all broadcast on a 2,048-node Connection Machine system CM-200 is 5.4 Gbyte/s. The index order of the data in local memory depends on implementation details of the algorithms, but it is well defined. If a linear ordering is desired, then including the time for local data reordering reduces the effective peak data rate to 2.5 Gbyte/s.

1 INTRODUCTION

All-to-all broadcast and reduction on distributed memory architectures are fundamental operations in several important linear algebra computations, such as matrix–vector and vector–matrix multiplication, rank-1 updates, and matrix–matrix multiplication. All-to-all broadcast is also critical for the performance of so-called direct N-body algorithms, where the evaluation of the pairwise interactions between all particles form the computational kernel.

An all-to-all broadcast can be accomplished by each node sending its data to a dedicated node, either one source node at a time, or all at once, followed by a broadcast of the data from the dedicated node to all other nodes. All-to-all communication can also be realized by shifting data along a Hamiltonian cycle (ring of all nodes). For high-degree networks, like binary cubes, this idea can be extended to the use of multiple Hamiltonian cycles that balance the communication load and maximize the bandwidth utilization [1, 15]. All-to-all reduction is, in effect, the reverse operation of a broadcast where combiners such as +, *max*, or *min* replace the copy operation. Figure 1 shows a single example of all-to-all reduction. The left part of the figure shows the initial data distribution. Components with the same index are added together. The result consists of eight components distributed evenly across all nodes in a consecutive (block) [12] manner. All nodes contain initial as well as final data.

The work reported here considers two forms of all-to-all communication in multiprocessor, dis-

This work was carried out while the authors were with Thinking Machines Corp.

Received January 1993
Revised April 1995

Scientific Programming, Vol. 4, pp. 251–273 (1995)
CCC 1058-9244/95/040251-23

P0	P1	P2	P3
0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7

Before reduction

P0	P1	P2	P3
0	2	4	6
1	3	5	7

After reduction

FIGURE 1 All-to-all reduction on a four-node system.

tributed memory architectures. In all-to-all broadcast, each processing node broadcasts its content to every other node in the system. In all-to-all reduction, reduction operations are performed concurrently on different data sets, each distributed over all nodes such that the results of the different reductions are evenly distributed over all nodes. Algorithms for all-to-all broadcast and reduction based on single and multiple Hamiltonian cycles in binary d-cubes are presented. The performance of implementations of the Hamiltonian cycles-based algorithms is compared with the performance of all-to-all communication based on edge-disjoint, multiple spanning trees of minimum height, and the performance of butterfly network-based algorithms. The primary intent of this article is to make available communication primitives that make run-time decisions to choose an optimal algorithm. Such primitives form the core of scientific libraries on distributed memory architectures.

In Section 2, we discuss the use of all-to-all broadcast and reduction in some matrix computations. Section 3 presents the relevant aspects of the Connection Machine system CM-200. Section 4 discusses in detail all-to-all communication based on Hamiltonian cycles for binary cubes. Section 5 discusses all-to-all communication based on spanning tree-based algorithms and compares the expected performance of the different approaches. Section 6 gives actual performance data for all-to-all communication on the Connection Machine system CM-200.

2 APPLICATIONS OF ALL-TO-ALL COMMUNICATION

An efficient implementation of all-to-all broadcast is of great importance for the performance of classical, direct N-body algorithms, in which every particle interacts with every other particle. In a distributed memory architecture, each processing node must communicate the particle information it stores in its memory to all other nodes. All-to-all communication is also required in iterative solvers for the finite element method [17] and in neural network simulations [25]. In both of these cases, the source of the all-to-all communication requirement is matrix–vector multiplication.

In the case of the direct N-body algorithms for gravitational calculations, the identity of the particles is not of interest. The coordinate and mass of each particle suffice, i.e., the array values suffice (with the particle coordinates stored in separate arrays). For matrix operations, the indices of array elements are not stored explicitly but are required for correct computations. In Section 4, we show how the indices of the array elements can be computed locally, thus reducing the need for communications bandwidth. Below, we illustrate the use of all-to-all communication in matrix computations.

The required data motion for matrix–vector and vector–matrix multiplication and for rank-1 updates (outer products) depends on the data allocation. As an example, consider matrix–vector multiplication, $y \leftarrow Ax$, with the matrix allocated to a one-dimensional nodal array with partitioning by rows and with the input and output vectors distributed evenly over all nodes as shown in Figure 2. An all-to-all broadcast of the input vector is required to carry out the matrix–vector product. No communication is required for the result vector. The matrix-vector multiplication can be expressed as (1) All-to-all broadcast of the input vector and (2) Local matrix–vector multiplication. If, instead, the matrix is allocated to a one-dimensional nodal array with partitioning by columns, as shown in Figure 3, and the input and output vectors are distributed evenly over the processing nodes, then no communication is required for the input vector, but an all-to-all reduction is required for the result vector. The matrix–vector

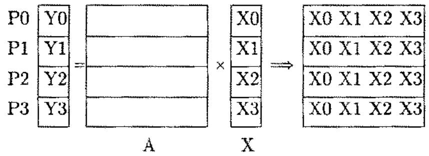


FIGURE 2 All-to-all broadcast for matrix–vector multiplication.

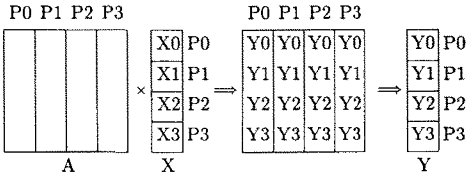


FIGURE 3 All-to-all reduction for matrix–vector multiplication.

multiplication can be expressed as (1) Local matrix–vector multiplication and (2) All-to-all reduction for the output vector

With the processing nodes configured as a two-dimensional nodal array for the matrix, but as a one-dimensional nodal array for the vectors, both all-to-all broadcast and all-to-all reduction are required in evaluating the matrix–vector product. Figure 4 illustrates the data allocation for both row major and column major ordering of the matrix allocation. The data allocation shown in Figure 4 is typical on Connection Machine systems, as explained in Section 3.

For a matrix of shape $P \times Q$ allocated to a two-dimensional nodal array in column major ordering, an all-to-all broadcast [8, 15, 19, 21] is required within the columns of the nodes for any shape of the nodal array and for any length of the matrix Q-axis.

After the all-to-all broadcast, each node performs a local matrix–vector multiplication. After this operation, each node contains a segment of the result vector y. The nodes in a row contain partial contributions to the same segment of y, while different rows of nodes contain contributions to different segments of y. No communication between rows of nodes is required for the computation of y. Communication within the rows of the nodes suffices.

The different segments of y can be computed by all-to-all reduction within processor rows, re-

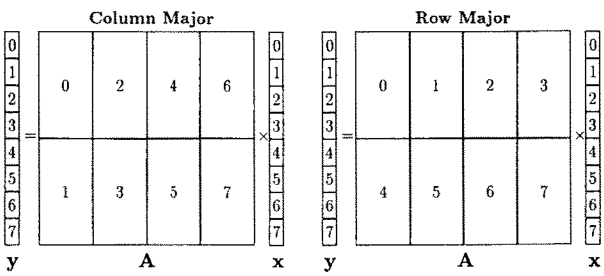


FIGURE 4 Data allocation on a rectangular nodal array.

sulting in a row major ordering of y. But, the node labeling is in column major ordering, and a reordering from row to column major ordering is required to establish the final allocation of y. Thus, for a column major ordering of the matrix elements to the nodes, matrix–vector multiplication can be expressed as:

1. All-to-all broadcast of the input vector within columns of nodes
2. Local matrix–vector multiplication
3. All-to-all reduction within rows of nodes to accumulate partial contributions to the result vector
4. Reordering of the result vector from row major to column major order

The reordering from row major ordering to column major ordering is equivalent to a shuffle or matrix transposition.

If the elements of the matrix A had been allocated in row major order instead of column major order, then a reordering from row major order to column major order must be performed prior to the all-to-all broadcast of the input vector. No reordering is required for y. Thus, for a row major ordering of matrix elements to nodes, the sequence of operations is:

1. Reordering of the input vector from row major to column major order
2. All-to-all broadcast of the input vector within columns of nodes
3. Local matrix–vector multiplication All-to-all reduction within rows of nodes to accumulate partial contributions to the result vector

With the matrix uniformly distributed across all nodes, the arithmetic is load balanced for both row major and column major order. The all-to-all broadcasts and all-to-all reductions are performed within the columns of the nodes and within the rows of the nodes, respectively. The different broadcast operations and the different reduction operations are completely independent of each other.

The communication requirements for vector–matrix multiplication are similar to those for matrix–vector multiplication. For outer products, yx^T, where y and x are column vectors, the communication issues for x are the same as in matrix–vector multiplication. For y, the communication issues are the same as for the input vector in vector–matrix multiplication. All-to-all broadcast and all-to-all reduction are also required in matrix–matrix multiplication [2, 6, 14, 18].

3 THE CONNECTION MACHINE SYSTEM CM-200

The Connection Machine system CM-200 [22] has up to 2048 nodes each consisting of a floating-point process or 4 Mbyte of local memory, and communication circuitry. The nodes are interconnected via a binary d-cube network, with a pair of bidirectional channels between adjacent nodes. In a binary cube network, each node has a neighbor for each bit in its binary address. The number of nodes is $N = 2^d$. There exist d edge-disjoint paths between each pair of nodes. Using multiple paths between nodes for maximum bandwidth utilization is the objective of the algorithms presented here. We then compare the performance of these algorithms with a few alternative implementations.

Each node in a Connection Machine system CM-200 can communicate concurrently on all its communication channels. The primitive communication operation is an exchange. The memory accesses in a node for each communication step are serialized. Each node supports one 4-byte wide access at a time to its local memory. The clock frequency is 10 MHz.

The programming model used for the Connection Machine systems uses a global address space, and each array is distributed as evenly as possible across all nodes. In a consecutive data allocation [12], a number of successive data elements along each axis are allocated to a node. For a one-dimensional data array of M elements allocated to N nodes, $\lceil \frac{M}{N} \rceil$ successive elements (a block) are assigned to the same node. In cyclic data allocation [12] of a one-dimensional array, elements $\{j | i = j \bmod N,\ 0 \leq j < M\}$ are allocated instead to the same node. Cyclic data allocation is currently not supported on the Connection Machine systems but is included in Fortran D [7], Vienna Fortran [5, 26], and the proposed high-performance Fortran (HPF) standard [10]. Cyclic allocation may yield improved load–balance with respect to arithmetic [12] or with respect to communication [16, 24]. In the case of multidimensional arrays, it is also necessary to determine how many elements along the different axes shall be allocated to the same processing node, or equivalently, how the set of processing nodes shall be configured. The Connection Machine run-time system determines the nodal array shape based on the data array shape, such that the local subarrays have axes of lengths as equal as possible. We refer to such a layout as a canonical layout. In the following, we assume consecutive, canonical layouts. Modifying the derivations to cyclic allocation is straightforward.

Regular grids are subgraphs of binary d-cubes. A Gray code has the property that successive integers differ in the code by a single bit, which, with a suitable labeling of the nodes in the binary cube, corresponds to the traversal of a single edge. Thus, Gray codes can be used in preserving adjacency in data arrays when mapped to binary cube networks. For multidimensional arrays, encoding each axis separately in a Gray code preserves adjacency. But, such an embedding makes efficient use of the processing nodes only when the data array axes have lengths equal to powers of 2. For lengths of other axes, adjacency cannot be preserved for a node-efficient mapping [3, 4, 11]. On the Connection Machine system CM-200, the default mapping of data arrays is based on a binary-reflected Gray code encoding [9, 12, 18] of the index along each axis separately. Only the part of the index corresponding to the node address is encoded in a binary-reflected Gray code. Binary encoding is always used for local addresses.

A d-bit binary-reflected Gray code, $\hat{G}_d$, is a sequence of 2^d nonnegative numbers in the range $\{0, 1, \ldots, 2^d - 1\}$, $\hat{G}_d = (G_d(0), G_d(1), \ldots, G_d(2^d - 1))$ defined recursively [18] by:

$$\hat{G}_1 = (G_1(0), G_1(1)), \text{ where } G_1(0) = G_1(1) = 1.$$

$$\hat{G}_{d+1} = \begin{pmatrix} 0\|G_d(0) \\ 0\|G_d(1) \\ \vdots \\ 0\|G_d(2^d-2) \\ 0\|G_d(2^d-1) \\ 1\|G_d(2^d-1) \\ 1\|G_d(2^d-2) \\ \vdots \\ 1\|G_d(1) \\ 1\|G_d(0) \end{pmatrix}.$$

In the following, we refer to this binary-reflected Gray code simply as Gray code. The 3-bit Gray code given in Table 1 clearly shows the recursive reflections in the code. It is also easily seen

Table 1. A Binary-Reflected Gray Code on 3 Bits

Integer	Gray Code
0	000
1	001
2	011
3	010
4	110
5	111
6	101
7	100

that the Gray code defines a Hamiltonian cycle. The sequence of bits that change in traversing the Gray code from beginning to end is known as the transition sequence. In the example of eight integers, the transition sequence is 0, 1, 0, 2, 0, 1, 0, and 2, with the least significant bit being bit 0.

4 ALL-TO-ALL ALGORITHMS USING HAMILTONIAN CYCLES

4.1 A Single Hamiltonian Cycle

Figure 5 illustrates the idea of all-to-all broadcast using a single cycle, whereas Figure 6 shows all-to-all reduction. In these figures, it is implicitly assumed that node addresses are encoded in Gray code, such that all communications are nearest neighbor. By performing the cyclic shifts in Figure 5 as left cyclic shifts, all elements arrive in order in node $P0$. In this node, local memory address s contains array element s, $0 \leq s < N$ for N nodes. The local memory reordering required for node j is $s \leftarrow (s - j) \bmod N$, i.e., a cyclic shift on the local memory addresses.

If the Gray code path is used for node addresses in binary order, then a local code conversion is required after the cyclic rotation among nodes has been completed. Figure 7 illustrates this fact. The array index in local memory address s of node 0 is $G(s)$. In general, let PA be the node address in binary code. Then, local memory address s in node PA contains the array element with index $G((s + G^{-1}(PA)) \text{ mode } N)$. For instance, consider $PA = 101$ and $s = 1$. The integer with Gray code 101 is 6. The Gray code of $1 + 6 = 7$ is 100, which is the second entry in the column for node 5.

Note that if each element in the examples in Figures 5 and 7 represents a block of elements, then moving these blocks as indicated in the figures results in a final distribution consistent with a consecutive data allocation. Conversely, a block partitioning of the data in each node prior to all-to-all reduction also yields a final data distribution consistent with a consecutive allocation.

Step	P0	P1	P2	P3
0	X0	X1	X2	X3
1	X0	X1	X2	X3
	X1	X2	X3	X0
2	X0	X1	X2	X3
	X1	X2	X3	X0
	X2	X3	X0	X1
3	X0	X1	X2	X3
	X1	X2	X3	X0
	X2	X3	X0	X1
	X3	X0	X1	X2

FIGURE 5 All-to-all broadcast through cyclic rotation.

Step	P0	P1	P2	P3
0	Y0	Y0	Y0	Y0
	Y1	Y1	Y1	Y1
	Y2	Y2	Y2	Y2
	Y3	Y3	Y3	Y3
1	Y0	-	Y0+Y0	Y0
	Y1	Y1		Y1+Y1
	Y2+Y2	Y2	Y2	-
	-	Y3+Y3	Y3	Y3
2	Y0		-	Y0+Y0+Y0
	Y1+Y1+Y1	Y1	-	-
	-	Y2+Y2+Y2	Y2	-
	-		Y3+Y3+Y3	Y3
3	Y0+Y0+Y0+Y0		-	
	-	Y1+Y1+Y1+Y1	-	-
	-		Y2+Y2+Y2+Y2	
	-		-	Y3+Y3+Y3+Y3

FIGURE 6 All-to-all reduction through cyclic rotation.

4.2 Multiple Hamiltonian Cycles

Broadcast

Johnsson and Ho [15] show that d Hamiltonian cycles fully exploit the communications band-

Node							
0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111
001	011	110	010	000	100	111	101
011	010	111	110	001	000	101	100
010	110	101	111	011	001	100	000
110	111	100	101	010	011	000	001
111	101	000	100	110	010	001	011
101	100	001	000	111	110	011	010
100	000	011	001	101	111	010	110

FIGURE 7 The index allocation resulting from 2^d cyclic shifts along a Gray code path for binary-encoded node indices.

Step	Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
Init.	0	00	01	02	03	04	05	06	07	
	1	10	11	12	13	14	15	16	17	
	2	20	21	22	23	24	25	26	27	
0	0	01	00	03	02	05	04	07	06	0
	1	12	13	10	11	16	17	14	15	1
	2	24	25	26	27	20	21	22	23	2
1	0	03	02	01	00	07	06	05	04	1
	1	16	17	14	15	12	13	10	11	2
	2	25	24	27	26	21	20	23	22	0
2	0	02	03	00	01	06	07	04	05	0
	1	14	15	16	17	10	11	12	13	1
	2	21	20	23	22	25	24	27	26	2
3	0	06	07	04	05	02	03	00	01	2
	1	15	14	17	16	11	10	13	12	0
	2	23	22	21	20	27	26	25	24	1
4	0	07	06	05	04	03	02	01	00	0
	1	17	16	15	14	13	12	11	10	1
	2	27	26	25	24	23	22	21	20	2
5	0	05	04	07	06	01	00	03	02	1
	1	13	12	11	10	17	16	15	14	2
	2	26	27	24	25	22	23	20	21	0
6	0	04	05	06	07	00	01	02	03	0
	1	11	10	13	12	15	14	17	16	1
	2	22	23	20	21	26	27	24	25	2

FIGURE 8 All-to-all broadcast using d channels in a d-cube with nodes labeled in binary order.

width in a binary d-cube for all-to-all broadcast and reduction. Figure 8 shows the $2^d - 1$ steps required to perform an all-to-all broadcast using d Hamiltonian cycles on a binary d-cube. In Figure 8, node addresses are in binary order. Figure 9 shows an all-to-all broadcast with node addresses in Gray code order. Initially, there are d distinct elements in each node. After the broadcast, each node has a total of $d2^d$ elements. With d channels per node, this operation requires at least $2^d - 1$ communications, because d elements are already present in each node before the broadcast. The algorithm below [15] requires precisely that many communications.

Step	Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
Init.	0	00	01	03	02	07	06	04	05	
	1	10	11	13	12	17	16	14	15	
	2	20	21	23	22	27	26	24	25	
0	0	01	00	02	03	06	07	05	04	0
	1	13	12	10	11	14	15	17	16	1
	2	27	26	24	25	20	21	23	22	2
1	0	02	03	01	00	05	04	06	07	1
	1	14	15	17	16	13	12	10	11	2
	2	26	27	25	24	21	20	22	23	0
2	0	03	02	00	01	04	05	07	06	0
	1	17	16	14	15	10	11	13	12	1
	2	21	20	22	23	26	27	25	24	2
3	0	04	05	07	06	03	02	00	01	2
	1	16	17	15	14	11	10	12	13	0
	2	22	23	21	20	25	24	26	27	1
4	0	05	04	06	07	02	03	01	00	0
	1	15	14	16	17	12	13	11	10	1
	2	25	24	26	27	22	23	21	20	2
5	0	06	07	05	04	01	00	02	03	1
	1	12	13	11	10	15	14	16	17	2
	2	24	25	27	26	23	22	20	21	0
6	0	07	06	04	05	00	01	03	02	0
	1	11	10	12	13	16	17	15	14	1
	2	23	22	20	21	24	25	27	26	2

FIGURE 9 All-to-all broadcast using d channels in a d-cube with nodes labeled in Gray code order.

For the algorithm using d Hamiltonian cycles, each node exchanges d elements concurrently in each step. When there are $M' > d$ elements in each node, the local memory is viewed as $\lceil \frac{M'}{d} \rceil$ blocks, of d elements each. The local memory address s consists of a block index, k $(0 \le k < \lceil \frac{M'}{d} \rceil)$, and an address, i $(0 \le i < d)$, within the block. For $d = 3$, the exchange sequence for location zero $(i = 0)$ within a block is 0, 1, 0, 2, 0, 1, 0, i.e., the same as the transition sequence in Table 1. The exchange sequence for location one is 1, 2, 1, 0, 1, 2, 1; for location two, it is 2, 0, 2, 1, 2, 0, 2. In general, if $t_0, t_1, \ldots, t_{2^d-2}$ is the exchange sequence for location zero, then the exchange sequence for location i is $(t_0 + i) \bmod d$, $(t_1 + i)$ mode d, $(t_2 + i) \bmod d, \ldots, (t_{2^d-2} + i) \bmod d$. Clearly, no two exchanges use the same dimension in any step.

For node addresses in binary code order, it can be shown that on completion, the index in local memory address $s = j \cdot M' + k \cdot d + i$ is: $PA \oplus sh^i(G(j)) \cdot M' + k \cdot d + i$, where PA is the node address in binary code as before, $sh(\cdot)$ is a left cyclic shift of the bit string representing the argument, and $0 \le j < 2^d$. For node addresses in Gray code order, the index in memory location zero initially is $G^{-1}(PA)$. On completion of the all-to-all broadcast, local memory address $s = j \cdot M' + k \cdot d + i$ in node PA contains data with index $[G^{-1}(PA) \oplus G^{-1}(sh^i(G(j)))] \cdot M' + k \cdot d + i$.

Note that the quantities $sh^i(G(j))$ and $G^{-1}(sh^i(G(j)))$ are identical for all nodes. Only PA, the binary address, and $G^{-1}(PA)$, the Gray code address, are unique to each node.

Note further that the index order for $i = 0$ in the d cycles algorithm is the same as in the single Hamiltonian cycle algorithm.

Reduction

The all-to-all broadcast algorithm, using d Hamiltonian cycles in a binary d-cube, can be adapted to all-to-all reduction. With $d \cdot 2^d$ variables in each node initially, each node in a 2^d cube accumulates d distributed variables. In each communication step, one exchange is performed on all d channels in each node, as in the broadcast algorithm.

Step	Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0	0					●	+			0
	1		●		+					1
	2			●				+		2
1	0					○	●		+	1
	1		○		●				+	2
	2			○				●	+	0
2	0					○	○	+	●	0
	1		○		○		+		●	1
	2			○	+			○	●	2
3	0			+		○	○	●	○	2
	1		○		○	+	●		○	0
	2		+	○	●			○	○	1
4	0			●	+	○	○	○	○	0
	1		○		○	●	○	+	○	1
	2		●	○	○		+	○	○	2
5	0		+	○	●	○	○	○	○	1
	1		○	+	○	○	○	●	○	2
	2		○	○	○	+	●	○	○	0
6	0	+	●	○	○	○	○	○	○	0
	1	+	○	●	○	○	○	○	○	1
	2	+	○	○	○	●	○	○	○	2

FIGURE 10 All-to-all reduction performed on d distributed variables with all d results resident in node zero on completion. Local memory addresses and node addresses in binary order.

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0					●	+			0
1		●		+					1
2			●				+		2
3					+	●			0
4	●		+						1
5				●				+	2
6							●	+	0
7		+		●					1
8	●				+				2
9							+	●	0
10	+		●						1
11		●				+			2
12	●	+							0
13						●		+	1
14			+				●		2
15	+	●							0
16					●		+		1
17				+				●	2
18			●	+					0
19						+		●	1
20	+				●				2
21			+	●					0
22					+		●		1
23		+				●			2

FIGURE 11 All-to-all reduction step 0 on a 3-cube. Memory addresses and node addresses in binary order.

For the description of the reduction algorithm, we first consider the accumulation of a single set of d distributed variables. Each of the d distributed variables has one element per node. Each distributed variable is accumulated independently of the others, with the d results accumulated to node zero. The reduction is illustrated in Figure 10. A filled circle denotes a partial sum being sent, + denotes a partial sum being received and added to a local variable, and an unfilled circle denotes values already added into a partial sum. Comparing Figure 10 with Figure 8, we notice that the data motion in Figure 10 is simply the reversed data motion of the elements originally in node zero in Figure 8.

The example in Figure 10 is an all-to-one reduction. An all-to-all reduction is obtained by considering an initial data set per node of $d \cdot 2^d$ elements, instead of d elements. Each block of d variables distributed across all nodes is accumulated to a single node, with different blocks of d distributed variables accumulated to different nodes. Each block is accumulated in a way similar to the single block of d distributed variables in an all-to-one reduction. By performing an exclusive-or operation with node address j on all node addresses used in communications for the block with destination node zero, the destination of the result of the reduction for the block becomes node j instead of node zero. The effect of the exclusive-or operation for each step is shown in Figures 11 through 17. In each step, each node exchanges one element on each of its channels, and performs one addition for each of d distinct sums. The total number of sums computed in each step is $d \cdot 2^d$.

The blocking used for the all-to-all reduction is identical to the blocking for all-to-all broadcast. This blocking is consistent with a consecutive data

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0					○	●		+	1
1		○		●				+	2
2			○				●	+	0
3					●	○	+		1
4	○		●				+		2
5				○			+	●	0
6						+	○	●	1
7		●		○		+			2
8	○				●	+			0
9					+		●	○	1
10	●		○		+				2
11		○			+	●			0
12	○	●		+					1
13				+		○		●	2
14			●	+			○		0
15	●	○	+						1
16			+		○		●		2
17			+	●				○	0
18		+	○	●					1
19		+				●		○	2
20	●	+			○				0
21	+		●	○					1
22	+				●		○		2
23	+	●				○			0

FIGURE 12 All-to-all reduction step 1 on a 3-cube. Memory addresses and node addresses in binary order.

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0					○	○	+	●	0
1		○		○		+		●	1
2			○	+			○	●	2
3					○	○	●	+	0
4	○		○		+		●		1
5			+	○			●	○	2
6					+	●	○	○	0
7		○		○		●		+	1
8	○	+			○	●			2
9					●	+	○	○	0
10	○		○		●		+		1
11	+	○			●	○			2
12	○	○	+	●					0
13		+		●		○		○	1
14			○	●			○	+	2
15	○	○	●	+					0
16	+		●		○		○		1
17			●	○			+	○	2
18	+	●	○	○					0
19		●		+		○		○	1
20	○	●			○	+			2
21	●	+	○	○					0
22	●		+		○		○		1
23	●	○			+	○			2

FIGURE 13 All-to-all reduction step 2 on a 3-cube. Memory addresses and node addresses in binary order.

allocation, i.e., d successive sums are allocated to the same node on completion. Furthermore, with both memory addresses and node addresses in binary order, successive blocks have successive nodes in binary code as their destinations. Thus, the local block index is the address of the node where the final sums shall be allocated.

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0			+		○	○	●	○	2
1		○		○	+	●		○	0
2		+	○	●			○	○	1
3				+	○	○	○	●	2
4	○		○		●	+	○		0
5	+		●	○			○	○	1
6	+				●	○	○	○	2
7		○		○		○	+	●	0
8	○	●		+	○	○			1
9		+			○	●	○	○	2
10	○		○		○		●	+	0
11	●	○	+		○	○			1
12	○	○	●	○			+		2
13	+	●		○		○		○	0
14			○	○		+	○	●	1
15	○	○	○	●				+	2
16	●	+	○		○		○		0
17			○	○	+		●	○	1
18	●	○	○	○	+				2
19		○	+	●		○		○	0
20	○	○			○	●		+	1
21	○	●	○	○		+			2
22	○		●	+	○		○		0
23	○	○			●	○	+		1

FIGURE 14 All-to-all reduction step 3 for a 3-cube. Memory addresses and node addresses in binary order.

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0			●	+	○	○	○	○	0
1		○		○	●	○	+	○	1
2		●	○	○		+	○	○	2
3			+	●	○	○	○	○	0
4	○		○		○	●	○	+	1
5	●		○	○	+		○	○	2
6	●	+			○	○	○	○	0
7		○		○	+	○	●	○	1
8	○	○		●	○	○		+	2
9	+	●			○	○	○	○	0
10	○		○		○	+	○	●	1
11	○	○	●		○	○	+		2
12	○	○	○	○			●	+	0
13	●	○	+	○		○		○	1
14		+	○	○		●	○	○	2
15	○	○	○	○			+	●	0
16	○	●	○	+	○		○		1
17	+		○	○	●		○	○	2
18	○	○	○	○	●	+			0
19	+	○	●	○		○		○	1
20	○	○		+	○	○		●	2
21	○	○	○	○	+	●			0
22	○	+	○	●		○		○	1
23	○	○	+		○	○	●		2

FIGURE 15 All-to-all reduction step 4 on a 3-cube. Memory addresses and node addresses in binary order.

Data Reordering: All Addresses in Binary Code

The Data Motion for Block Zero. We first consider the data motion for block $j = 0$. The transition sequence for a binary-reflected Gray code is symmetric with respect to its midpoint. Thus, per-

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0		+	○	●	○	○	○	○	1
1		○	+	○	○	○	●	○	2
2		○	○	○	+	●	○	○	0
3	+		●	○	○	○	○	○	1
4	○		○	+	○	○	○	●	2
5	○		○	○	●	+	○	○	0
6	○	●		+	○	○	○	○	1
7	+	○		○	●	○	○	○	2
8	○	○		○	○	○	+	●	0
9	●	○	+		○	○	○	○	1
10	○	+	○		○	●	○	○	2
11	○	○	○		○	○	●	+	0
12	○	○	○	○		+	○	●	1
13	○	○	●	○		○	+	○	2
14	+	●	○	○		○	○	○	0
15	○	○	○	○	+		●	○	1
16	○	○	○	●	○		○	+	2
17	●	+	○	○	○		○	○	0
18	○	○	○	○	○	●		+	1
19	●	○	○	○	+	○		○	2
20	○	○	+	●	○	○		○	0
21	○	○	○	○	●	○	+		1
22	○	●	○	○	○	+	○		2
23	○	○	●	+	○	○	○		0

FIGURE 16 All-to-all reduction step 5 on a 3-cube. Memory addresses and node addresses in binary order.

Mem	P0	P1	P2	P3	P4	P5	P6	P7	Dim
0	+	●	○	○	○	○	○	○	0
1	+	○	●	○	○	○	○	○	1
2	+	○	○	○	●	○	○	○	2
3	●	+	○	○	○	○	○	○	0
4	○	+	○	●	○	○	○	○	1
5	○	+	○	○	○	●	○	○	2
6	○	○	+	●	○	○	○	○	0
7	●	○	+	○	○	○	○	○	1
8	○	○	+	○	○	○	●	○	2
9	○	○	●	+	○	○	○	○	0
10	○	●	○	+	○	○	○	○	1
11	○	○	○	+	○	○	○	●	2
12	○	○	○	○	+	●	○	○	0
13	○	○	○	○	+	○	●	○	1
14	●	○	○	○	+	○	○	○	2
15	○	○	○	○	●	+	○	○	0
16	○	○	○	○	○	+	○	●	1
17	○	●	○	○	○	+	○	○	2
18	○	○	○	○	○	○	+	●	0
19	○	○	○	○	●	○	+	○	1
20	○	○	●	○	○	○	+	○	2
21	○	○	○	○	○	○	●	+	0
22	○	○	○	○	○	●	○	+	1
23	○	○	○	●	○	○	○	+	2

FIGURE 17 All-to-all reduction step 6 on a 3-cube. Memory addresses and node addresses in binary order.

forming the exchanges in reverse order is identical to the original transition sequence. The exchange sequence for a given local memory location is the same in broadcast and reduction. However, the starting location for the reduction is the location where the last copy is to be deposited in the broadcast algorithm.

From item 3, we see that for local memory address zero, the exchanges generate the binary-reflected Gray code addresses in reverse order. The addresses for sequence i is obtained by an i step cyclic rotation of the addresses of sequence zero. Thus, for block $j = 0$ and memory and node addresses in binary code the sending node address for local memory address i in block $j = 0$ in step u is $sh^i(G(2^d - 1 - u))$, and the receiving node address for local memory address i in block $j = 0$ in step u is $sh^i(G(2^d - u))$.

The Data Motion for Block j. The destination node for block j is node j. The starting node for block j is obtained by performing an exclusive-or operation with j (translation) on the starting addresses for block $j = 0$. Exchange sequence i is used for all local memory addresses i relative to the beginning of the blocks, i.e., all local memory addresses such that s mode $d = i$. Thus,

1. the starting address for local memory address i in block j is: $j \oplus 2^{(d-1+i) \bmod d}$.
2. The exchange sequence for local memory address i in block j as: $(t_0 + i) \bmod d$, $(t_1 + i) \bmod d$, $(t_2 + i) \bmod d$, ... $(t_{2^d-2} + i) \bmod d$.
3. The sending node address for local memory address i in block j in step u is: $j \oplus sh^i(G(2^d - 1 - u))$.
4. The receiving node address for local memory location i in block j in step u is: $j \oplus sh^i(G(2^d - u))$.

The above formulas give the sending and receiving nodes for a given block j and local memory address i within the block. However, at every step each node only sends the contents of d local memory addresses and receives d data elements to be added into the contents of d local memory addresses. Thus, it is of more direct interest to determine for each node which local memory address of what blocks is participating in communication step u. Node PA is sending and receiving data from local memory address i within block j if $PA = j \oplus sh^i(G(2^d - 1 - u))$ and $PA = j \oplus sh^i(G(2^d - u))$, respectively. Thus, the block number j for local memory address i within node PA in step u is $j = PA \oplus sh^i(G(2^d - 1 - u))$ for sending and $j = PA \oplus sh^i(G(2^d - u))$ for receiving.

Note that the expressions $sh^i(G(2^d - 1 - u))$ and $sh^i(G(2^d - u))$ are common to all nodes. Thus, on the Connection Machine systems, these expressions can be evaluated on the front-end.

Data Reordering: Node Addresses in Gray Code Order

In the broadcast algorithm, the final destination for local memory address zero ($j = 0$, $i = 0$) of node zero is node 2^{d-1}, with memory and node addresses in binary code. The exchange sequence used for this memory location is the same as the transition sequence in a binary-reflected Gray code with the dimensions taken in order. Thus, the final destination is the last address in the Gray code, with dimensions in order, i.e., node 2^{d-1}. The final destination of local memory address i of block $j = 0$ within node zero is node $2^{(d-1+i) \bmod d}$, because the ith exchange sequence is obtained from the exchange sequence of local address zero by adding $i \bmod d$ to the exchange dimension for local address zero. For instance, for $d = 3$ the last destination of the content of local memory address zero in node zero is node four, of location one it is node one, and of location two it is node two, as seen from Figure 8.

For the all-to-all reduction algorithm with memory and node addresses in binary code, we have for block $j = 0$:

1. Starting address for local memory address i within block $j = 0$ is: $2^{(d-1+i)\bmod d}$.
2. The exchange sequence for local memory address i in block $j = 0$ is: $(t_0 + i) \bmod d$, $(t_1 + i) \bmod d$, $(t_2 + i) \bmod d$, . . . , $(t_{2^d-2} + i) \bmod d$.
3. The address of the receiving node for local memory address i in block $j = 0$ step u is: $2^{(d-1+i)\bmod d} \oplus 2^{(t_u+i)\bmod d}$, $u = \{0, 1, 2, \ldots, 2^d - 1\}$,
4. The sending node in step $u \geq 1$ is the receiving node in step $u - 1$.

When the result of the all-to-all reduction is desired in Gray code order, then instead of selecting block j as described above, the block whose Gray code is j should be selected. For instance, for $d = 3$, 7 has the Gray code 100. Thus, in every instance when block 4 is selected for the result in binary order, block 7 is selected for the result in Gray code order. Then, on completion, the reduction on block 7 is available in node 4.

Thus, for the result in Gray code order, the block index j chosen for local memory address and exchange sequence i is

1. $j = G^{-1}\ (PA \oplus sh^i(G(2^d - 1 - s))) = G^{-1}(PA) \oplus T^{-1}(shG(2^d - 1 - s)))$ for sending.
2. $j = G^{-1}\ (PA \oplus sh^i(G(2^d - s))) = G^{-1}(PA) \oplus G^{-1}(sh^i(G(2^d - s)))$ for receiving.

The address $G^{-1}(PA)$ is the node address in Gray code. Thus, the operations unique to a node consists in determining its Gray code address and an exclusive-or operation.

All-to-All Reduction on Local Data Sets of Arbitrary Size

For a local data set of size M, 2^d blocks are created for all-to-all reduction on a binary d-cube. When $M \bmod 2^d \neq 0$, then not all blocks have the same size. On the Connection Machine systems any array with a size that is not divisible by the number of nodes over which it is distributed is allocated such that $\lceil \frac{M}{2d} \rceil$ elements are assigned to the first $\alpha = \lfloor M/\lceil \frac{M}{2d} \rceil \rfloor$ nodes. The remaining $M - \alpha \lceil \frac{M}{2d} \rceil 2^d$ elements are assigned to a single node, and the remaining $2^d - \alpha - 1$ nodes are assigned no elements. Thus, in blocking a data set for all-to-all reduction, we first create $\alpha + 1$ nonempty blocks, α of which consists of $\lceil \frac{M}{2d} \rceil$ consecutive memory locations. Each block is further subdivided into $2d$ subblocks. For the first α subblocks, the maximum number of elements in a subblock is $\beta = \lceil \frac{\lceil M/2d \rceil}{2d} \rceil$. All elements within a subblock are subject to the same exchange sequence, while different subblocks are subject to different exchange sequences. The number of subblocks with β elements each is $\gamma = \lceil M/2^d \rceil - d(\beta - 1)$. The memory partitioning is illustrated in Figure 18.

For each exchange step u, a pair of successive elements is transmitted from a subblock (j, i) selected for transmission in dimension i in that exchange step. The exchange step u is not completed until all data elements within a subblock selected for transmission have been transmitted.

The actual transmission can be viewed as consisting of three phases:

1. The movement of data to be transmitted in one exchange to a buffer area, a departure lounge.
2. An exchange of $2d$ 32-bit data elements, with the received data being stored in a buffer area, an arrival lounge.
3. Reduction on local data, and data in the arrival lounge.

Successive pairs of elements in the departure lounge are taken from subblocks with indices $i = \{0, 1, 2, \ldots, d - 1\}$ in blocks j determined by the expressions given previously. If there are no more elements in subblock i to be transmitted, then the buffer location is empty, and the corresponding channel not utilized. The number of exchanges for each step u is $\lceil \frac{\beta S}{2} \rceil$, where S is the number of 32-bit words required for the data type ($S = 2$ for real-8 and complex-8 and $S = 4$ for complex-16). The reduction after each exchange step is performed by adding the contents of the arrival lounge for step u to the contents of the departure lounge for step $u + 1$, with the elements for the same exchange sequence i being added together.

5 OTHER ALGORITHMS

A few alternatives to the Hamiltonian cycle-based algorithms for all-to-all communication are:

1. Each node broadcasts the values directly to all other nodes (one course node at a time)

Block	sub-block	subblock size	Block size
	0	β	
	1	β	
	2	β	
	·	β	$\lceil M/2^d \rceil$
	·	β	
0	$\gamma - 1$	β	
	γ	$\beta - 1$	$(\beta = \lceil \frac{\lceil M/2^d \rceil}{d} \rceil)$
	$\gamma + 1$	$\beta - 1$	
	·	$\beta - 1$	
	·	$\beta - 1$	
	$d - 1$	$\beta - 1$	
	0	β	
	1	β	
	2	β	
	·	β	$\lceil M/2^d \rceil$
	·	β	
1	$\gamma - 1$	β	
	γ	$\beta - 1$	$(\beta = \lceil \frac{\lceil M/2^d \rceil}{d} \rceil)$
	$\gamma + 1$	$\beta - 1$	
	·	$\beta - 1$	
	·	$\beta - 1$	
	$d - 1$	$\beta - 1$	
	·	·	
	·	·	
	·	·	
	·	·	
	0	β	
	1	β	
	2	β	
	·	β	$\lceil M/2^d \rceil$
	·	β	
$\alpha - 1$	$\gamma - 1$	β	
	γ	$\beta - 1$	$(\beta = \lceil \frac{\lceil M/2^d \rceil}{d} \rceil)$
	$\gamma + 1$	$\beta - 1$	
	·	$\beta - 1$	
	·	$\beta - 1$	
	$d - 1$	$\beta - 1$	
	0	δ	
	1	δ	
	2	δ	
	·	δ	$M - \alpha\lceil M/2^d \rceil$
	·	δ	
α	$\varepsilon - 1$	δ	
	ε	$\delta - 1$	$(\delta = \lceil \frac{M - \alpha\lceil M/2^d \rceil}{d} \rceil)$
	$\varepsilon + 1$	$\delta - 1$	
	·	$\delta - 1$	
	·	$\delta - 1$	
	$d - 1$	$\delta - 1$	

FIGURE 18 Memory partitioning.

using multiple spanning trees, each of which uses all the communication channels of the binary cube (d edge-disjoint spanning trees for a d-cube) [15].

2. Each node sends its data to a dedicated node that broadcasts the data to all other nodes with an algorithm using all channels of a d-cube.
3. All nodes send their data to a dedicated node concurrently, followed by a broadcast from the dedicated node to all other nodes.
4. All nodes send their data to all other nodes using minimum height spanning trees, such as d rotated spanning binomial trees [15]. This algorithm is equivalent to a butterfly network-based algorithm.

Alternatives 1 and 2 use multiple spanning trees to maximize the bandwidth utilization in broadcasting the data from a single node. Nodes are treated sequentially. Alternative 3 combines a gather operation (all-to-one personalized communication [14]) with a broadcast as in alternative 1, but there is only a single broadcast of all data to be received by a node. In all-to-one personalized communication, each node sends data to one node. (In all-to-all personalized communication, each node sends unique data to every other node.) Alternatives 1 and 4 both have optimum time complexity, whereas alternatives 2 and 3 require extra data motion. Table 2 summarizes the performance estimates for the different algorithms [15]. For the reduce-and-spread estimate in Table 2, a subselection is assumed before carrying out the transpose required for data in column major order. The transposed data volume is a factor of 2^d less than the data volume in the reduce-and-spread function.

On the Connection Machine system CM-200 fairly well-optimized routines have been implemented for broadcast from a single node, shifts along a single Hamiltonian cycle, shifts along $k \leq d$ Hamiltonian cycles for a d-cube, and reduce-and-spread in d-cubes based on rotated binomial trees.

The broadcast function currently available on the Connection Machine system CM-200 assumes that the source for the broadcast operation is the first node in a segment (first row or column). Thus, this function can only be used in alternatives 2 and 3. The overhead in the spread function is quite significant, as is apparent from the timings shown in Table 3. This table shows timings for 2 to 32,768 32-bit elements per node on binary cubes with 2 to 2048 nodes. Because of the large overhead, only alternative 3 of the first three tree-based algorithms is considered further.

The first step in alternative 3 is an all-to-one personalized communication. The optimum time for this operation is $\lceil \frac{M}{2d} \rceil$ [15], where M is the number of elements gathered into a node. There is currently no optimized routine available on the Connection Machine systems CM-200 for this

Table 2. Estimated Number of Element Transfers in Sequence for Different Broadcast Algorithms*

Ordering	Operation	Time
Column major	Broadcast using single Hamiltonian cycle	$\lceil \frac{M}{2\cdot 2^d} \rceil (2^d - 1)$
	Broadcast, d Hamiltonian cycles	$\lceil \frac{M}{2d\cdot 2^d} \rceil (2^d - 1)$
	Broadcast from one node at a time (alternative 1)	$(\lceil \frac{M}{2d\cdot 2^d} \rceil + d)2^d$
	Gather followed by broadcast (alternative 3)	$\lceil \frac{M}{2d} \rceil + \lceil \frac{M}{2d} \rceil + d = 2\lceil \frac{M}{2d} \rceil + d$
	Broadcast based on rotated binomial trees (alternative 4)	$\lceil \frac{M}{2d\cdot 2^d} \rceil (2^d - 1)$
	Reduce-and-spread, transpose	$\lceil \frac{M}{2d\cdot 2^d} \rceil d + \lceil \frac{M}{2\cdot 2\cdot 2^{2d}} \rceil$
Row major	Transpose, broadcast using single Hamiltonian cycle	$\lceil \frac{M}{2\cdot 2\cdot 2^d} \rceil + \lceil \frac{M}{2\cdot 2^d} \rceil (2^d - 1)$
	Transpose, broadcast using d Hamiltonian cycles	$\lceil \frac{M}{2\cdot 2\cdot 2^d} \rceil + \lceil \frac{M}{2d\cdot 2^d} \rceil (2^d - 1)$
	Transpose, broadcast from one node at a time (alternative 1)	$\lceil \frac{M}{2\cdot 2\cdot 2^d} \rceil + (\lceil \frac{M}{2d\cdot 2^d} \rceil + d)2^d$
	Transpose, gather, broadcast (alternative 3)	$\lceil \frac{M}{2\cdot 2\cdot 2^d} \rceil + \lceil \frac{M}{2d} \rceil + \lceil \frac{M}{2d} \rceil = 2\lceil \frac{M}{2d} \rceil + \lceil \frac{M}{2\cdot 2^d} \rceil$
	Reduce-and-spread	$\lceil \frac{M}{2d} \rceil d$

* M is the total number of 32-bit elements prior to the broadcast (or reduction) operation.

communication. The Connection Machine router is used instead. For matrix–vector multiplication with the matrix allocated to a one-dimensional nodal array through partitioning by rows, and the vectors distributed evenly across all nodes, the nodes send their segment of the input vector to node zero, which then broadcasts the entire input vector to every node. With a two-dimensional nodal array shape in column major order for the matrix, and a one-dimensional nodal array shape for the vectors, all nodes within a node column send their segment of the input vector to the first node within the column. Then, this node broadcasts all segments of the input vector within a node column to all nodes in that node column. In row major ordering, the send operation must also accomplish a transposition from row to column major order. Although the transposition is implicit in the send, it has a significant impact on the routing time for the send, as shown in the performance measurements in Section 6.

The optimal transpose time for a binary cube with two channels between each pair of nodes is $\lceil \frac{M}{2\cdot 2\cdot 2^d} \rceil$ [13, 15]. The optimal time is proportional to

Table 3. Time (ms) for Broadcast of Different Size 32-bit Data Sets and Connection Machine Systems CM-200 of Various Sizes

Number of Elements	Number of Nodes 2	4	8	16	32	64	128	256	512	1024	2048
2	0.0366	0.383	0.448	0.502	0.573	0.640	0.719	0.789	0.878	0.965	1.036
4	0.0366	0.383	0.448	0.502	0.573	0.640	0.719	0.789	0.878	0.965	1.036
8	0.0623	0.416	0.480	0.505	0.576	0.644	0.722	0.792	0.881	0.968	1.040
16	0.1136	0.470	0.515	0.542	0.615	0.684	0.763	0.798	0.888	0.975	1.046
32	0.2162	0.577	0.598	0.600	0.691	0.730	0.811	0.848	0.939	1.027	1.101
64	0.4214	0.793	0.741	0.720	0.773	0.856	0.907	0.947	1.042	1.133	1.168
128	0.8318	1.223	1.055	0.962	0.963	0.984	1.044	1.045	1.247	1.305	1.344
256	1.6527	2.084	1.654	1.445	1.377	1.352	1.360	1.338	1.410	1.434	1.503
512	3.3139	3.805	2.882	2.410	2.203	2.054	1.990	1.925	1.946	1.953	2.001
1024	6.7319	7.247	5.310	4.342	3.825	3.491	3.287	3.098	3.021	2.991	2.957
2048	13.4530	14.129	10.193	8.204	7.067	6.331	5.844	5.445	5.210	5.028	4.910
4096	26.9059	27.895	19.931	15.929	13.585	12.048	10.960	10.137	9.588	9.102	8.782
8192	53.8115	55.426	39.437	31.379	26.619	23.444	21.226	19.522	18.344	17.288	16.520
16384	107.6210	110.476	78.415	62.279	52.657	46.269	41.721	38.291	35.818	33.662	31.997
32768	215.2400	220.577	156.396	124.070	104.723	91.874	82.711	75.829	70.764	66.366	62.991

Step	P0	P1	P2	P3	P4	P5	P6	P7	Dim.
Init.	00 10 20	01 11 21	02 12 22	03 13 23	04 14 24	05 15 25	06 16 26	07 17 27	
0	00+01 10+12 20+24	00+01 11+13 21+25	02+03 10+12 22+26	02+03 11+13 23+27	04+05 14+16 20+24	04+05 15+17 21+25	06+07 14+16 22+26	06+07 15+17 23+27	0 1 2
1	00+01+02+03 10+12+14+16 20+21+24+25	00+01+02+03 11+13+15+17 20+21+24+25	00+01+02+03 10+12+14+16 22+23+26+27	00+01+02+03 11+13+15+17 22+23+26+27	04+05+06+07 10+12+14+16 20+21+24+25	04+05+06+07 11+13+15+17 20+21+24+25	04+05+06+07 10+12+14+16 22+23+26+27	04+05+06+07 11+13+15+17 22+23+26+27	1 2 0
2	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	00 – 07 10 – 17 20 – 27	2 0 1

FIGURE 19 Reduce-and-spread through butterfly network emulation.

the size of the local data set but is independent of the partitioning of nodes between rows and columns.

Alternative 4 has been implemented for reduce-and-spread functionality on the Connection Machine system CM-200. It uses a constant size data set in all stages. Exchanges are performed between pairs of nodes in d stages, as shown in Figure 19. Each node computes d partial sums, one for each of d distributed variables. The accumulation and broadcast of distributed variable zero use the dimensions in increasing order. The accumulation and broadcast of distributed variable i use dimension $(u + i) \bmod d$ in step u. With M elements per node and a double cube, the number of element transfers is $\lceil \frac{M}{2d} \rceil d$, which is a factor d higher than an all-to-all reduce using d Hamiltonian cycles. The reduce-and-spread function yields 2^d times as much local data as is desired for an all-to-all reduction. The desired data are selected from the result of the reduce-and-spread operation.

With the matrix allocated to a two-dimensional nodal array in row major ordering, the result of the all-to-all reduction within rows yields the result in the desired order. With a column major ordering, a reordering from row to column major ordering (transposition) is required after the all-to-all reduction.

If the node addresses are encoded in a Gray code instead of a binary code, then the ordering of the elements in local memory is different. But, except for local memory operations, all other operations are identical.

6 PERFORMANCE MEASUREMENTS

For all-to-all broadcast, we have implemented algorithms based on a single Hamiltonian cycle, d Hamiltonian cycles, and based on gathering all column data into a single node followed by broadcast from that node. For all-to-all reduction, we implemented the first two algorithms and compared them with reduce-and-spread. We first present the results for broadcast, then for reduction.

6.1 All-to-All Broadcast

Tables 4 and 5 summarize measurements for a 256-node Connection Machine system CM-200.

Table 4. Execution Times (ms) for All-to-All Broadcast Using Three Different Methods on the Connection Machine System CM-200*

		All-to-All Broadcast										
Words per	No. of	Send-and-Spread				*d*-Cycles				One Cycle		
Node Initially	Nodes	Send	Spread	Total	Transpose	Addr.	Indir.	AABC	Total	Indir.	AABC	Total
4	16	2.20	0.92	3.12	0.90	1.84	0.57	1.00	4.31	1.17	1.00	3.07
8	16	3.82	1.38	5.20	1.13	3.52	0.62	1.10	6.57	1.21	1.66	4.00
16	16	8.00	2.30	10.30	1.68	3.36	0.73	1.85	7.62	1.31	3.03	6.02
32	16	9.59	4.14	13.73	1.80	3.39	0.89	3.35	9.43	1.56	5.76	9.12
64	16	19.19	7.82	27.01	3.34	4.14	1.55	6.41	15.44	2.03	11.21	16.58
128	16	38.60	15.17	53.77	6.55	5.24	2.87	12.55	27.21	2.98	22.13	31.66
256	16	77.74	29.88	107.62	12.90	8.57	5.52	24.82	51.81	4.88	43.97	61.75
512	16	156.70	59.31	216.01	25.66	15.44	10.80	49.43	101.33	8.68	87.63	121.97
1024	16	315.80	118.20	434.00	51.43	29.07	21.38	98.49	200.37	16.27	175.00	242.70

* 64-bit precision; row major ordering; node addresses in Gray code.

Table 5. Execution Times (ms) for All-to-All Broadcast Using Three Different Methods on the Connection Machine System CM-200*

		All-to-All Broadcast									
		Send-and-Spread				*d*-Cycles			One Cycle		
Words per Node Initially	No. of Nodes	Send	Spread	Total	Transpose	Addr. + Indir.	AABC	Total	Indir.	AABC	Total
256	1	3.28	0.13	3.41	4.93	1.52	0.00	6.45	0.33	0	5.26
256	2	22.41	6.27	28.68	13.59	3.24	5.70	22.53	0.63	2.93	17.15
256	4	35.93	13.46	49.39	12.98	4.69	9.33	27.00	1.24	8.79	23.01
256	8	49.67	18.98	68.65	13.13	8.47	15.16	36.76	2.45	20.51	36.09
256	16	77.74	29.88	107.62	12.90	14.02	24.82	51.74	4.88	43.97	61.75
256	32	100.20	50.14	150.34	12.98	32.35	42.51	87.84	9.75	90.91	112.64
256	64	165.10	87.50	252.60	12.98	63.46	74.38	150.82	19.47	184.80	217.25
256	128	254.20	156.90	411.10	12.91	131.20	131.80	276.91	38.92	372.80	424.63
256	256	457.70	286.70	744.40	4.93	226.10	228.80	459.83	77.80	748.80	831.53

* 64-bit precision; row major ordering; node addresses in Gray code.

In Table 4, the nodes are configured as a 16×16 array. The length of the local segment of the vector to be broadcast varies. In Table 5, the vector length per node is fixed, while the number of nodes along an axis is varied. Both tables assume a row major ordering and node addresses in Gray code. The column marked "total" includes the transpose time. Thus, all columns marked total correspond to the same functionality.

From Table 5, we first note that the transpose time is independent of nodal array shape, as predicted for an optimal algorithm. The transposition is done by the router, which thus shows a very good performance behavior. The time per 32-bit data element amounts to about 25.1 μs.

From the performance measurements, we conclude that both the single cycle and the d-cycles algorithm are always faster than the send-and-spread algorithm for all-to-all broadcast. Table 6 summarizes the relative performance of the cycle-based algorithms over the send-and-spread algorithm. The improvement is in the range 1.27–2.08. The measured data motion rate for all-to-all broadcast on a 256-node Connection Machine system CM-200 is about 0.3 Gbyte/s.

Table 6. Ratio of Data Motion Rates for the Three Methods of All-to-All Broadcast on the Connection Machine System CM-200*

	All-to-All Broadcast		
		256 Words/Node	
No. of Nodes	Send-and-Spread	*d*-Cycles	One Cycle
2	1	1.27	1.67
4	1	1.83	2.15
8	1	1.87	1.90
16	1	2.08	1.74
32	1	1.71	1.33
64	1	1.67	1.16
128	1	1.48	0.97
256	1	1.62	0.90

* 64-bit precision; row major ordering; node addresses in Gray code.

To compare the single cycle and d-cycles algorithms, we consider the performance characteristics of the two algorithms in detail. The execution time for all-to-all broadcast through a single Hamiltonian cycle is expected to behave as

$$T^B_{1-H,c} = \left(a_1 + a_2 \left\lceil \frac{M' \cdot S}{2} \right\rceil\right) (2^d - 1),$$

where a_1 is the overhead for each step of the algorithm and a_2 is the exchange time for two 32-bit data elements. M' is the initial number of elements per node and S is the number of 32-bit words per data element ($S = 2$ for real-8 and complex-8). From Table 4, we derive $a_1 = 24.3$ μs and $a_2 = 5.69$ μs.

With a Gray code ordering, the indirect addressing required for the reordering on completion requires a time of

$$T^B_{1-H,l} = a_3 + (a^B_4 + a^B_5 M' \cdot S) 2^d,$$

where, from our performance measurements, $a_3 = 17.9$ μs, $a_4 = 50.33$ μs, and $a_5 = 0.5$ μs. Adding $T^B_{1-H,c}$ and $T^B_{1-H,l}$ we get the total time for the all-to-all broadcast based on a single Hamiltonian cycle and nodes in Gray code order:

$$T^B_{1-H} = -6.38 - 5.69 \cdot M' \cdot S + 74.6 \cdot 2^d + 6.19 \cdot M' \cdot S \cdot 2^d.$$

In the d-cycles all-to-all broadcast algorithm, the data motion between nodes is expected to show a performance behavior of the form

$$T^B_{d-H.c} = \left(b_1 + (b_2 \cdot 2d + b_3)\left\lceil \frac{M' \cdot S}{2d} \right\rceil\right)(2^d - 1),$$

where b_1 is the overhead for each step of the algorithm, b_2 is the time for local memory references for each step, and b_3 the time for exchanging d pairs of 32-bit words between a node and its d neighbors. From the column labeled d-cycles in Tables 4 and 5, we derive: $b_1 = 41.94\ \mu s$, $b_2 = 0.17\ \mu s$, and $b_3 = 24.18\ \mu s$.

In addition to the data motion time, a significant time is also required for local memory operations, even when nodes are labeled in Gray code order. For the single-cycle algorithm, a node-dependent block cyclic shift is required. In the d-cycles algorithm, the reordering at the end, as illustrated in Figures 8 and 9, requires full indirect addressing. In addition, if the array indices are not moved along with the data, index computation is required according to the formulas given in "Data Reordering: All Addresses in Binary Code". For the index computation, there are $2^d - 1$ blocks, one for each remote node, for which index computations are needed. Each such block consists of $2d$ subblocks of $\lceil \frac{M/2^d}{2d} \rceil$ elements each, with the final subarray of size M (initial subarrays of size $M' = M/2^d$ elements each). Then, there is a remainder of $d' = (M/2^d) \bmod 2d$ elements for each of the $2^d - 1$ blocks. Thus, the time for index computation $T_{d-H.ic}$ behaves as

$$T^B_{d-H.ic} = b_4 + \left[d\left(b_5 + b_6\left\lceil \frac{M'}{2d} \right\rceil\right) + b_7 M' \bmod 2d\right](2^d - 1).$$

From our measurements, we derived: $b_4 = 2056.35$, $b_5 = 41.08$, $b_6 = 2.02$, and $b_7 = 22.24$, all in μs. The index computation has a faster growth rate in the number of nodes than the motion of the data. This is clear in Table 7.

The time for local data reordering with indirect addressing is

$$T^B_{d-H.I} = (b_8 + b_9 \cdot M' \cdot S) \cdot 2^c,$$

where the constants derived from our measurements are: $b_8 = 14.78\ \mu s$ and $b_9 = 0.645\ \mu s$.

The total time for all-to-all broadcast based on d-cycles and local index computation is obtained as $T^B_{d-H} - T^B_{d-H.c} + T^B_{d-H.ic} + T^B_{d-H.I}$, or

$$\begin{aligned} T^B_{d-H} = {} & 2071.13 + 0.645 \cdot M' \cdot S \\ & + (56.72 + 0.645 \cdot M' \cdot S \\ & + 24.18\lceil (M' \cdot S)/(2d) \rceil \\ & + d(41.08 + 0.34\lceil (M' \cdot S)/2d \rceil \\ & + 2.02\lceil M/(2d) \rceil) \\ & + 22.24(M' \bmod 2d))(2^d - 1). \end{aligned}$$

Because the efficiency in the index computation on the CM-200 is poor, we have also imple-

Table 7. Execution Time (ms) for d-Cycles All-to-All Broadcast with Index Computation and Index Motion*

Words per Node Initially	No. of Nodes	d-Cycles All-to-All Broadcast					
				Comp. Index		Move Index	
		AABC	Indir.	Comp.	Total	Move	Total
128	2	2.95	0.36	1.39	4.70	2.05	5.36
128	4	4.79	0.72	1.85	7.36	2.97	8.48
128	8	7.90	1.44	3.43	12.77	4.56	13.90
128	16	12.64	2.88	5.30	20.82	6.96	22.48
128	32	21.64	5.75	16.83	44.22	11.60	38.99
128	64	37.82	11.50	35.56	84.88	19.93	69.25
128	128	69.90	23.01	60.10	153.01	36.49	129.40
128	256	116.50	46.01	119.50	282.01	60.37	222.88
128	512	232.40	92.05	280.20	604.65	120.80	445.25
128	1024	416.40	184.10	727.80	1328.30	216.90	817.40
128	2048	738.40	370.30	1973.00	3081.70	386.00	1494.70

* 64-Bit precision; node addresses in Gray code.

mented all-to-all broadcast using a d-cycles algorithm in which the indices are moved along with the data. The indices are represented as 32-bit integers regardless of the type of the array values. Local reordering is still required and requires the same time as when the array indices are computed locally. Table 7 shows the results for a 2048-node CM-200. As seen in the table, moving indices is faster than computing indices for 32 or more nodes. The time for moving indices, $T_{d-H,im}$ is

$$T_{d-H,im} = \left(b_{10} + (b_{11} \cdot 2d + b_{12}) \left\lceil \frac{M'}{2d} \right\rceil\right) (2^d - 1) + b_{13}$$

where $b_{10} = 33.61\ \mu s$, $b_{11} = 0.63\ \mu s$, $b_{12} = 16.98\ \mu s$, and $b_{13} = 1834.05\ \mu s$ and the total time is $T_{d-H,c} + T_{d-H,im} + T_{d-H,l}$

$$\begin{aligned} T^B_{d-H,m} = {} & 1848.83 + 0.645 \cdot M' \cdot S \\ & + (90.33 + 0.645 \cdot M' \cdot S \\ & + (0.34d + 24.18)\lceil (M' \cdot S)/(2d) \rceil \\ & + (1.26d + 16.98)\lceil M'/(2d) \rceil)(2^d - 1). \end{aligned}$$

Comparing the times for all-to-all broadcast based on a single cycle and d-cycles with either local index computation or index motion, we conclude that the actual communication time for the d-cycles algorithm is less than that for the single-cycle algorithm for eight or more nodes. The gain is less than a factor of d, because the single-cycle algorithm can use machine instructions that require fewer cycles per word transfer. However, the performance gain in the communication part of the d-cycles algorithm is largely lost due to the time required for the index calculations, and the full indirect addressing required in the reordering of the data. From the expressions for the total execution time for the single-cycle and d-cycles algorithms, we can derive the size of the local data set as a function of d for which the d-cycles algorithm yields better performance. The results are summarized in Table 8. From the table we conclude that, in practice, at least 16 nodes ($d \geq 4$) must be assigned to an axis before the d-cycles algorithm performs better than the single-cycle algorithm, largely due to the expense for index calculation. This expense grows sufficiently rapidly to make moving the data indices more efficient than computing them locally for 32 or more nodes, despite the simple operations required for index computation. Figure 20 shows the execution times for all-to-all broadcast on up to 512 nodes using either a single-cycle algorithm, or the d-cycles algorithm with either index computation or index motion. Table 9 gives the corresponding measured data.

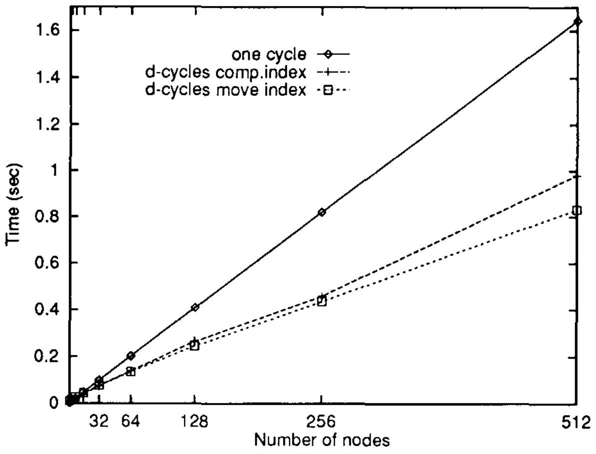


FIGURE 20 All-to-all broadcast time in seconds for 256 64-bit elements.

Remark 1. It is interesting to compare the performance of the spread function with that of the d-cycles algorithm. Ideally, both should have the same execution time (see Table 2). From Tables 4 and 5, the measured performance is comparable when the address calculation time is excluded, as expected. The total execution times are compared in Table 10.

Table 8. Size of the Initial Data Set (M') per Node in 64-Bit Precision for Which the d-Cycles All-to-All Broadcast Algorithms with Index Computation (d – H, c) and Index Motion (d – H, m) Yield Better Performance than a Single-Cycle Algorithm*

No. of Nodes	2	4	8	16	32	64	128	256	512	1024	2048
d – H, c			$M' \geq 240$	$M' \geq 80$	$M' \geq 60$	$M' \geq 48$	$M' \geq 56$	$M' \geq 48$	$M' \geq 54$	$M' \geq 60$	$M' \geq 66$
d – H, m				$M' \geq 55$	$M' \geq 14$	$M' \geq 12$	$M' \geq 14$	$M' \geq 16$	$M' \geq 18$	$M' \geq 20$	$M' \geq 22$

* Row major ordering: node addresses in Gray code.

Table 9. Execution Time (ms) for *d*-Cycles All-to-All Broadcast with Index Computation and Index Motion and for the One-Cycle Algorithm*

		d-Cycles						One Cycle		
				Comp. Index		Move Index				
Words per Node Initially	No. of Nodes	AABC	Indir.	Comp.	Total	Move	Total	AABC	Indir.	Total
256	2	5.84	0.69	2.54	9.07	3.96	10.49	2.93	0.54	3.47
256	4	9.47	1.38	3.31	14.16	5.79	16.64	8.80	1.09	9.89
256	8	15.31	2.76	5.74	23.81	8.73	26.80	20.54	2.18	22.72
256	16	24.97	5.52	8.59	39.08	13.60	44.09	44.05	4.37	48.42
256	32	42.69	11.06	21.58	75.33	22.60	76.35	91.08	8.75	99.83
256	64	74.59	22.11	41.98	138.68	38.79	135.49	185.10	17.49	202.59
256	128	132.10	44.22	88.37	264.69	68.05	244.37	373.40	34.98	408.38
256	256	229.20	88.41	140.30	457.91	117.10	434.71	750.20	69.98	820.18
256	512	434.00	177.00	369.70	980.70	222.00	833.00	1504.00	140.00	1644.00

* 64-Bit precision; node addresses in Gray code.

Remark 2. It is also interesting to note that although the optimal time for the send and the spread is the same (Table 2), the measured time for the send is about 2.7 times higher than for the spread in the 16-node case. For the eight-node case, the ratio is about 2.6 and for the 256-node case, the ratio is 1.6. The send (gather) operation uses the router whereas the spread uses an optimized algorithm. Table 10 gives a comparison of the execution times for send and spread.

The sensitivity of the send times to row or column major layout ordering was examined by measuring the execution times for both orderings. In a column major ordering, the send is confined to within subcubes and no transpose is required for all-to-all communication. Tables 11 and 12 summarize the results. The send is faster by close to a factor of 2 for the column major ordering, but the cycles-based algorithms are also faster for this ordering. However, the speed advantage is not as large as for row major ordering.

Finally, we also measured the performance for all-to-all communication with node addresses in binary code. The execution times for the *d*-cycles algorithm were almost identical to the times for node addresses in Gray code order. The single-cycles algorithm would require a different implementation on the Connection Machine system CM-200, because the CSHIFT intrinsic function used in our implementation uses the general router for node addresses in binary code. A special implementation of our single-cycle algorithm should yield comparable performance for node addresses in binary and Gray code.

Table 10. Relative Execution Times for Spread, *d*-Cycles All-to-All Broadcast with Reordering and Send (Gather)*

No. of Nodes	Spread	*d*-Cycles AABC	Send (Gather)
2	1	3.60	3.57
4	1	2.01	2.67
8	1	1.94	2.62
16	1	1.74	2.60
32	1	1.76	2.00
64	1	1.74	1.89
128	1	1.77	1.62
256	1	1.62	1.60

* 64-bit precision; row major ordering; node addresses in Gray code.

6.2 Reduction

Tables 13 and 14 give the measured execution times for all-to-all reduction based on reduce-and-spread, and the single-cycle and *d*-cycles algorithms.

The reduce-and-spread alternative for all-to-all reduction results in an excessive amount of data in each node, and a subselection is required to arrive at the final result. This subselection is performed by a call to the Connection Machine router, even though no communication is required. The router is the only general mechanism currently available on the Connection Machine system CM-200 for this subselection. Performing the all-to-all reduction in this manner is always

Table 11. Execution Times (ms) for All-to-All Broadcast Using Three Different Methods on the Connection Machine System CM-200*

		All-to-All Broadcast									
		Send-and-Spread			d-Cycles				One Cycle		
Words per Node	No. of Nodes	Send	Spread	Total	Addr.	Indir.	AABC	Total	Indir.	AABC	Total
4	16	1.23	0.92	2.15	1.84	0.57	1.00	3.41	1.17	1.00	2.17
8	16	2.30	1.38	3.68	3.52	0.62	1.10	5.24	1.21	1.66	2.87
16	16	4.79	2.30	7.09	3.36	0.73	1.85	5.94	1.31	3.03	4.34
32	16	5.42	4.14	9.56	3.39	0.89	3.35	7.63	1.56	5.76	7.32
64	16	10.78	7.82	18.60	4.14	1.55	6.41	12.10	2.03	11.21	13.24
128	16	21.59	15.17	36.76	5.24	2.87	12.55	20.66	2.98	22.13	25.11
256	16	43.42	29.88	73.30	8.57	5.52	24.82	38.91	4.88	43.97	48.85
512	16	87.41	59.31	146.72	15.44	10.80	49.43	75.67	8.68	87.63	96.31
1024	16	176.10	118.20	294.30	29.07	21.38	98.49	148.94	16.27	175.00	191.27

* 64-bit precision; column major ordering; node addresses in Gray code.

Table 12. Execution Times (ms) for All-to-All Broadcast Using Three Different Methods on the Connection Machine System CM-200*

		All-to-All Broadcast								
		Send-and-Spread			d-Cycles			One Cycle		
Words per Node	No. of Nodes	Send	Spread	Total	Addr. + Indir.	AABC	Total	Indir.	AABC	Total
256	1	3.30	0.13	3.45	1.52	0.00	1.52	0.33	0	0.33
256	2	11.97	6.27	18.38	3.24	5.70	8.94	0.63	2.93	3.56
256	4	17.69	13.46	31.88	4.69	9.33	14.02	1.24	8.79	10.03
256	8	26.67	18.98	46.54	8.47	15.16	23.63	2.45	20.51	22.96
256	16	43.42	29.88	74.63	14.02	24.82	38.84	4.88	43.97	49.85
256	32	69.46	50.14	121.82	32.35	42.51	74.86	9.75	90.91	100.66
256	64	114.66	87.50	206.33	63.46	74.38	137.84	19.47	184.80	204.27
256	128	225.43	156.90	388.90	131.20	131.80	263.00	38.92	372.80	411.72
256	256	—	286.70	—	226.10	228.80	454.90	77.80	748.80	826.60

* 64-bit precision; column major ordering; node addresses in Gray code.

Table 13. Execution Times (ms) for All-to-All Broadcast Using Reduce-and-Spread, d-Cycles, and One Cycle on the Connection Machine System CM-200*

		All-to-All Reduction									
		Reduce-and-Spread			d-Cycles				One Cycle			
Words per Node	No. of Nodes	Reduce-and-Spread	Send	Total	Indir.	Arit.	Comm.	Total	Indir.	Arit.	Comm.	Total
4	16	1.17	1.62	3.33	2.43	0.26	1.11	3.80	0.59	0.26	1.07	1.92
8	16	3.36	3.57	6.93	4.20	0.30	1.22	5.72	0.64	0.30	1.74	2.68
16	16	6.70	8.11	14.81	4.26	0.38	2.03	6.67	0.73	0.38	3.10	4.21
32	16	13.39	15.32	28.71	4.32	0.51	3.61	8.44	0.98	0.51	5.83	7.32
64	16	26.77	32.74	59.51	5.68	0.87	6.88	13.43	1.45	0.87	11.29	13.61
128	16	53.54	67.81	121.30	8.16	1.61	13.48	23.25	2.40	1.60	22.20	26.20
256	16	107.10	138.50	245.60	14.26	3.06	26.66	43.98	4.30	3.04	44.03	51.37
512	16	214.10	281.10	495.20	26.29	5.97	53.04	85.30	8.09	5.96	87.70	101.74
1024	16	428.20	568.30	996.50	50.55	11.78	105.70	168.03	15.69	11.74	175.00	202.43

* 64-bit precision; row major ordering; node addresses in Gray code.

Table 14. Execution Times (ms) Data for All-to-All Reduction Using Reduce-and-Spread, *d*-Cycles, and One Cycle on the Connection Machine System CM-200*

		All-to-All Broadcast									
					d-Cycles						
		Reduce-and-Spread						One Cycle			
Words per Node	No. of Nodes	Spread	Send	Total	Indir. + Addr. + Arithm.	Comm.	Total	Indir.	Arithm.	Comm.	Total
256	2	12.73	22.68	35.41	1.17	11.30	12.47	0.95	0.20	2.93	4.08
256	4	26.88	44.77	71.65	2.51	12.81	15.32	1.49	0.61	8.80	10.90
256	8	53.30	80.32	133.62	7.15	17.55	24.70	2.57	1.43	20.54	24.54
256	16	107.00	138.60	245.60	12.74	26.63	39.37	4.73	3.07	44.05	51.85
256	32	215.90	237.60	453.50	38.68	44.00	82.68	9.04	6.33	91.08	106.45
256	64	436.20	410.90	847.10	78.99	75.65	154.64	17.67	12.87	185.10	215.64
256	128	882.90	722.00	1604.90	172.54	133.00	305.54	34.92	25.94	373.40	434.26
256	256	1786.00	1293.00	3079.00	298.95	229.70	528.65	69.42	52.10	750.20	871.72
256	512	—	—	—	—	—	—	138.50	104.40	1504.00	1746.90

* 64-bit precision; row major order; node addresses in Gray code.

less efficient than using either a single-cycle algorithm or the *d*-cycles algorithm.

For a 16 × 16 nodal array, the single-cycle algorithm is more efficient than the *d*-cycles algorithm for a final data set per node of at most 64 elements. The single-cycle all-to-all reduction is about 6% slower than the corresponding broadcast operation, whereas the *d*-cycles all-to-all reduction is about 12% slower than the corresponding all-to-all broadcast. (In these percentage calculations, we excluded the time for the transpose required in all-to-all broadcast for row major ordering, in order to highlight the difference between broadcast and reduction.) The performance trade-off between the single-cycle and the *d*-cycles algorithms is approximately the same as for the broadcast.

Table 15 gives a comparison of the total execution times for the three different all-to-all reduction methods: reduce-and-spread followed by a subselection, a single-cycle algorithm, a *d*-cycles algorithm. The cycle-based algorithms yield a speedup of a factor of 5 or better over the reduce-and-spread function.

Remark

Note that the send (scatter) that follows the reduce-and-spread may require more time than the reduce-and-spread function itself. Because all nodes have all the results after the reduce-and-spread, the desired result can be obtained either as a local subselection, or as a one-to-all personalized communication from the first node in a row.

Table 15. Execution Times (ms) and Speedups for Three Methods of Performing an All-to-All Reduction on the Connection Machine System CM-200*

		All-to-All Reduction				
			d-Cycles		One Cycle	
Words per Node	No. of Nodes	Reduce-and-send	Time	Speedup	Time	Speedup
256	2	35.41	12.47	2.84	4.08	8.68
256	4	71.65	15.32	4.68	10.90	6.57
256	8	133.62	24.70	5.41	24.54	5.44
256	16	245.60	39.37	6.24	51.85	4.74
256	32	453.50	82.68	5.48	106.45	4.26
256	64	847.10	154.64	5.48	215.64	3.93
256	128	1604.90	305.54	5.25	434.26	3.70
256	256	3079.00	528.65	5.82	871.72	3.53
256	512	—	—	—	1746.90	—

* 64-bit precision; row major ordering; node addresses in Gray code.

Table 16. Execution Time (ms) and Relative Speeds of All-to-One Personalized Communication (Gather) and One-to-All Personalized Communication (Scatter) by the Connection Machine System CM-200 Router*

Words per Node	No. of Nodes	All-to-One (Gather)	One-to-All (Scatter)	Gather / Scatter
256	2	22.41	22.68	0.99
256	4	35.93	44.77	0.80
256	8	49.67	80.32	0.62
256	16	77.74	138.60	0.56
256	32	100.20	237.60	0.42
256	64	165.10	410.90	0.40
256	128	254.20	722.00	0.35
256	256	457.70	1293.00	0.35

* 64-bit precision; row major ordering; node addresses in Gray code.

On the Connection Machine system CM-200, both methods require approximately the same time. The latter operation is like a vector transpose, with the vector initially stored in the first node of a row, and stored uniformly across all nodes in a row after the transpose. This operation is the reverse communication of the send that gathers data to a single node before the broadcast in the send-and-broadcast algorithm for all-to-all broadcast.

However, the one-to-all personalized communication performed by the router requires considerably more time than the all-to-one personalized communication. Table 16 compares the timings for all-to-one and one-to-all personalized communication. Table 17 compares the times for all-to-all reduction using the d-cycles algorithms with local index computation and index motion. Moving indices is faster than computing them locally for 64 or more nodes.

7 SUMMARY

We have presented detailed schedules for all-to-all communication algorithms for broadcast and reduction based on Hamiltonian cycles. The cycle-based algorithms perform the all-to-all broadcast in $2^d - 1$ steps. In each step, a pair of successive memory locations are transmitted in the same cube dimension, thereby exploiting the fact that there are two channels between each pair of Connection Machine system CM-200 processing nodes.

For broadcast, both the single-cycle and the d-cycles algorithms always yield better performance than an algorithm using the router for gathering all data into one node, followed by a spread. The speedup is in the range $1.5 - 3.2$ for four or more nodes along an axis. The measured peak data motion rate for the d-cycles algorithm with indices moved along with the data is 2.54 Gbyte/s on a 2048-node Connection Machine system CM-200. Without the index computations and corresponding local data reordering, the measured all-to-all broadcast peak rate is 5.4 Gbyte/s. The measured peak data motion rates for all-to-all broadcast are summarized in Table 18. The data motion rate for spread is included for comparison but does not represent the time for all-to-all broadcast using spreads.

For all-to-all reduction, the speedup of our Hamiltonian cycle-based algorithms is even

Table 17. Execution Time (ms) for d-Cycles All-to-All Reduction with Index Computation and Index Motion*

Words per Node Initially	No. of Nodes	d-Cycles All-to-All Broadcast						
					Comp. Index		Move Index	
		AABC	Indir.	Arith.	Comp.	Total	Move	Total
256	2	11.41	0.82	0.22	0.13	12.58	3.97	16.42
256	4	12.93	1.51	0.61	0.39	15.44	5.79	20.84
256	8	17.70	2.89	1.43	2.83	24.85	8.74	30.76
256	16	26.79	5.63	3.07	4.04	39.53	13.60	49.09
256	32	44.20	11.12	6.35	20.90	82.88	22.60	84.27
256	64	75.91	22.12	12.91	44.00	154.90	38.80	149.70
256	128	133.20	44.10	26.03	102.40	305.70	68.10	271.40
256	256	230.30	88.08	52.27	158.60	529.30	117.10	487.80
256	512	435.00	176.00	104.70	—	—	222.00	937.70

* 64-bit precision; node addresses in Gray code.

Table 18. Data Motion Rates in Mbyte s^{-1} per Node on CM-200*

Number of Nodes	Spread	All-to-All Broadcast: No Index and Reorder: One Cycle	No Index and Reorder: d-Cycles	Index and Reorder: One Cycle	Index and Reorder: d-Cycles	Index and Reorder: Move Index
2	0.65	0.66	0.33	0.52	0.20	0.18
4	0.61	0.66	0.61	0.57	0.39	0.34
8	0.87	0.66	0.86	0.59	0.52	0.48
16	1.10	0.66	1.16	0.60	0.68	0.63
32	1.31	0.66	1.40	0.60	0.66	0.75
64	1.50	0.66	1.62	0.60	0.70	0.85
128	1.67	0.66	1.77	0.60	0.78	0.92
256	1.87	0.66	2.13	0.60	0.85	1.07
512	—	0.66	2.14	0.60	0.79	1.07
1024	—	0.66	2.40	0.60	0.73	1.16
2048	—	0.66	2.70	0.60	0.63	1.27

* All-to-all times computed from 128 elements per node prior to broadcast. The data motion rate for spread is included for comparison, but does not represent the time for all-to-all broadcast using the spread algorithm.

greater than for broadcast, with the range being 5–8.

The performance for the cycles-based algorithms is fairly independent of whether the data allocation is in row or column major ordering, and whether the nodal addresses are in binary or Gray code. However, the router performance depends significantly on whether the data allocation is in row or column major ordering.

The d-cycles algorithm offers a good improvement in performance over the single-cycle algorithm with respect to data motion. However, the local computation of indices is quite inefficient. This offset of the gain in communication time makes the single-cycle algorithm preferable for moderate size initial data sets, and few nodes assigned to the axis. For 64 or more nodes assigned to an axis, it is more efficient in the d-cycles algo-

Table 19. Performance Data for Matrix–Vector and Vector–Matrix Multiplication on Different Connection Machine System CM-200 Configurations (64-Bit Precision)

Matrix Shape $P \times P$	Mflops/s: No. of Nodes: 256	512	1024	2048	Time (ms): No. of Nodes: 256	512	1024	2048
Matrix–vector multiplication								
512	—	—	—	—	—	—	—	—
1024	304	—	—	—	6.90	—	—	—
2048	723	898	—	—	11.6	9.34	—	—
4096	1190	1834	2486	—	28.2	18.3	13.5	—
8192	1382	2621	4358	6101	97.1	51.2	30.8	22.0
12288	—	2796	4992	—	—	—	60.5	—
16384	—	—	5162	9833	—	—	104.0	—
24576	—	—	—	10785	—	—	—	112.0
Vector–matrix multiplication								
512	—	—	—	—	—	—	—	—
1024	344	—	—	—	6.09	—	—	—
2048	799	1037	—	—	10.5	8.09	—	—
4096	1370	2059	2844	—	24.5	16.3	11.8	—
8192	1846	3093	5103	6991	72.7	43.4	26.3	19.2
12288	—	3553	6252	—	—	85.0	48.3	—
16384	—	—	6918	11621	—	—	77.6	46.2
24576	—	—	—	13742	—	—	—	87.9

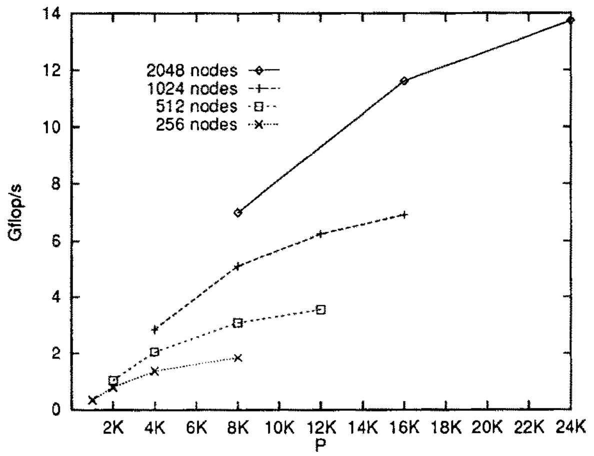


FIGURE 21 Execution rate in Gflop/s for multiplication of a $P \times P$ matrix by a vector on Connection Machine system CM-200 (64-bit precision).

rithm to move the indices along with the data than to compute the indices locally for both all-to-all broadcast and all-to-all reduction.

We have incorporated the Hamiltonian cycle-based all-to-all communication routines in the matrix–vector and vector–matrix multiplication and rank-1 update routines of the Connection Machine Scientific Software Library, CMSSL [23], Version 3.0. A summary of the performance of the matrix–vector and vector-matrix routines are given in Table 19 and in Figure 21.

ACKNOWLEDGMENTS

This work was carried out with partial support from the U.S. Air Force Office of Scientific Research under grant F49620-93-1-0480 with Harvard University and the U.S. Office of Naval Research under grant N00014-93-1-0192 also with Harvard University. The majority of this effort was supported by Thinking Machines Corp. as part of its Connection Machine Scientific Software Library (CMSSL) development. The software reported in this paper is available as part of the CMSSL.

REFERENCES

[1] J.-P. Brunet and S. L. Johnsson, "All-to-all broadcast with applications on the Connection Machine," *Int. J. Supercomput. Appl.*, vol. 6, pp. 241–256, 1992.

[2] L. E. Cannon, "A cellular computer to implement the kalman filter algorithm," PhD thesis, Montana State University, 1969.

[3] M. Y. Chan, "Embeddings of 3-dimensional grids into optimal hypercubes," in *Proc. of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, 1990, p. 297.

[4] M. Y. Chan, "Embedding of grids into optimal hypercubes," *SIAM J. Comput.* vol. 20, pp. 834–864, 1991.

[5] B. Chapman, P. Mehrotra, and H. Zima, "Programming in vienna Fortran," *Sci. Programming*, vol. 1, pp. 31–50, 1992.

[6] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," *SIAM J. Comput.*, vol. 10, pp. 657–673, 1981.

[7] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu, "Fortran D language specification," Department of Computer Science, Rice University, Tech. Rep. TR90-141, Dec. 1990.

[8] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[9] E. N. Gilbert, "Gray codes and paths on the *n*-cube," *Bell Systems Tech. J.*, vol. 37, pp. 815–826, 1958.

[10] "High Performance Fortran Language Specification, Version 1," *Sci. Programming*, vol. 2, nos. 1–2, pp. 1–170, 1993.

[11] C.-T. Ho and S. L. Johnsson, "Embedding meshes in Boolean cubes by graph decomposition," *J. Parallel Distrib. Comput.*, vol. 8, pp. 325–339, April 1990.

[12] S. L. Johnsson, "Communication efficient basic linear algebra computations on hypercube architectures," *J. Parallel Distrib. Comput.*, vol. 4, pp. 133–172, April 1987.

[13] S. L. Johnsson and C.-T. Ho, "Matrix transposition on Boolean n-cube configured ensemble architectures," *SIAM J. Matrix Anal. Appl.*, vol. 9, pp. 419–454, July 1988.

[14] S. L. Johnsson and C.-T. Ho, "Matrix multiplication on Boolean cubes using generic communication primitives," in *Parallel Processing and Medium Scale Multiprocessors*. SIAM, 1989, pp. 108–156.

[15] S. L. Johnsson and C.-T. Ho, "Spanning graphs for optimum broadcasting and personalized communication in hypercubes," *IEEE Trans. Comput.*, vol. 38, pp. 1249–1268, September 1989.

[16] S. L. Johnsson, C.-T. Ho, M. Jacquemin, and A. Ruttenberg, "Computing fast Fourier transforms on Boolean cubes and related networks," in *Advanced Algorithms and Architectures for Signal Processing II*, vol. 826. Society of Photo-Optical Instrumentation Engineers, 1987, pp. 223–231.

[17] S. L. Johnsson and K. K. Mathur, "Data struc-

tures and algorithms for the finite element method on a data parallel supercomputer," *Int. J. Numerical Methods Eng.*, vol. 29, pp. 881–908, 1990.

[18] K. K. Mathur and S. L. Johnsson, "Multiplication of matrices of arbitrary shape on a data parallel computer," *Parallel Comput.*, vol. 20, pp. 919–951, July 1994.

[19] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

[20] Q. F. Stout and B. Wagar, "Intensive hypercube communication I: Prearranged communication in link-bound machines." Computing Research Laboratory, University of Michigan, Ann Arbor, MI, Tech. Rep. CRL-TR-9-87, 1987.

[21] Q. F. Stout and B. Wagar, "Passing messages in link-bound hypercubes," in *Hypercube Multiprocessors 1987*, Michael T. Heath, Ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1987.

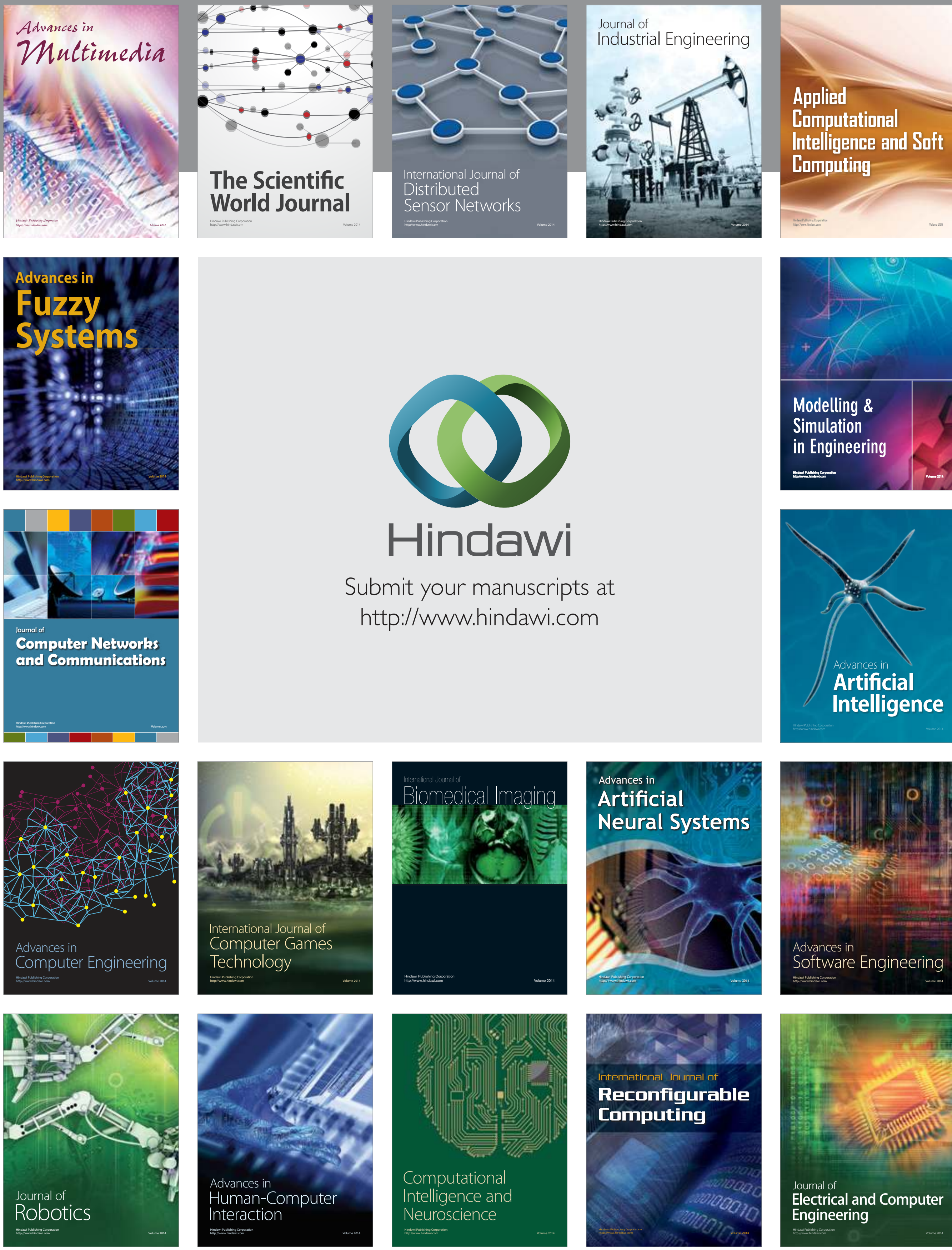
[22] Thinking Machines Corp., *CM-200 Technical Summary*. Thinking Machines, 1991.

[23] Thinking Machines Corp., *CMSSL for CM Fortran, Version 3.1*. Thinking Machines, 1993.

[24] C. Tong and P. N. Swarztrauber, "Ordered Fast Fourier transforms on a massively parallel hypercube multiprocessor," *J. Parallel Distrib. Comput.*, vol. 12, pp. 50–59, May 1991.

[25] X. Zhang, "An efficient implementation of the backpropagation algorithm on the Connection Machine CM-2," *Adv. Neural Information Processing Systems*, vol. 2, pp. 801–809, 1989.

[26] H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald," Vienna Fortran—A language specification version 1.1.," Tech. Rep. ICASE, Interim Report 21, March 1992.


Advances in Multimedia
The Scientific World Journal
International Journal of Distributed Sensor Networks
Journal of Industrial Engineering
Applied Computational Intelligence and Soft Computing
Advances in Fuzzy Systems
Modelling & Simulation in Engineering
Hindawi
Submit your manuscripts at
http://www.hindawi.com
Journal of Computer Networks and Communications
Advances in Artificial Intelligence
Advances in Computer Engineering
International Journal of Computer Games Technology
International Journal of Biomedical Imaging
Advances in Artificial Neural Systems
Advances in Software Engineering
Journal of Robotics
Advances in Human-Computer Interaction
Computational Intelligence and Neuroscience
International Journal of Reconfigurable Computing
Journal of Electrical and Computer Engineering