Data Motion and High Performance Computing

Citation

Johnsson, S. Lennart. 1994. Data Motion and High Performance Computing. Harvard Computer Science Group Technical Report TR-09-94.

Permanent link

<http://nrs.harvard.edu/urn-3:HUL.InstRepos:24829623>

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA>

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. [Submit a story](#).

[Accessibility](#)

Data Motion and High Performance Computing

S. Lennart Johnsson

TR-09-94

April 1994

Parallel Computing Research Group
Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

To appear in the First International Workshop on *Massively Parallel Processing Using
Optical Interconnections*, April 26–27, 1994.

Data Motion and High Performance Computing

S. Lennart Johnsson
Thinking Machines Corp.
and
Harvard University
Cambridge, MA

Abstract

Efficient data motion has been key in high performance computing almost since the first electronic computers were built. Providing sufficient memory bandwidth to balance the capacity of processors led to memory hierarchies, banked and interleaved memories. With the rapid evolution of MOS technologies, microprocessor and memory designs, it is realistic to build systems with thousands of processors and a sustained performance of a trillion operations per second or more. Such systems require tens of thousands of memory banks, even when locality of reference is exploited. Using conventional technologies, interconnecting several thousand processors with tens of thousands of memory banks can feasibly only be made by some form of sparse interconnection network. Efficient use of locality of reference and network bandwidth is critical. We review these issues in this paper.

1 Introduction

Preserving locality of reference has been critical in achieving high performance almost as long as electronic computers have been in existence. Even in the early days of electronic computers, memory was relatively slow compared to the ALU. Over time the reasons for the difference in speed has varied. For the first thirty or so years, much of the difference was due to the fact that different technologies were used for processors and memory. For the last 10 - 15 years, memories and processors are largely produced in the same technology. However, architectural innovations for both processors and memory systems, together with the characteristics of the MOS technology that is dominating in the computer industry, cause processor speeds to exceed the speed of the primary memory. In fact, the speed of microprocessors in MOS technology improves over time at a much higher rate than MOS memories in the form of DRAMs. Currently, the speed of microprocessors is doubling about every 18 months, while

Figure 1: The memory system for distributed memory architectures.

the DRAM speed has been improving at a steady rate of about 7%/yr. Thus, even for high performance microprocessor based systems, much more sophisticated memory systems are required in the future for a balanced design. In the near future, the memory system must account for a speed difference by a factor of about 10.

High performance computers have had multiple processors, an interconnection network, and a relatively large number of memory banks amounting to several banks per processor for a balance between processor speed and memory speed. In massively parallel processors, memory is distributed among the processors as shown in Figure 1, thereby reducing the demands on the communication network when there is locality of reference in the computations.

In most existing massively parallel processors the local memory bandwidth in each node is significantly higher than the bandwidth by which a node can send and receive data. This difference may be aggravated by contention in the communication system. Thus, for massively parallel processors, achieving a high utilization of the computational resources requires a data mapping that exploits locality of reference with respect to a distributed memory hierarchy, and a data routing such that the impact of remote references are minimized.

For coarse grained computations routing for mini-

mum contention tends to be more critical than routing for minimum latency. Coarse grained computations often have sufficient excess parallelism that latency can be hidden with the proper software techniques and hardware support. Both latency and contention are dependent upon the relative placement of data sets, the interconnection network, and the reference pattern. Much work has been expended in devising efficient routing algorithms for different routing patterns and networks, as well as for oblivious algorithms.

In this paper we will discuss some of the benefits of exploiting locality of reference and some techniques for exploiting locality of reference on distributed memory systems. We will also briefly discuss some routing techniques.

2 Locality of reference

In this section we give a few examples of the memory references required by a few frequent computations. In the next section we comment on how these benefits can be attained, and in particular how the benefits relates to the mapping of data to memory.

It is well known that matrix-vector multiplication can be organized such that it is sufficient to load one operand from memory for each multiply-add operation. Thus, in 64-bit precision, 8 bytes of memory bandwidth per floating-point operation suffice. Matrix-matrix multiplication, $C \leftarrow A \times B + C$, allows for a further reduction in memory bandwidth requirement. It may be performed as a sequence of operations on b by b sub-blocks. If the blocks fit into the registers, then $2b^3$ floating-point operations may be computed using $3b^2$ input elements (b^2 elements per operand), producing b^2 results. If all contributions to a block of C are accumulated in registers, then it suffices to load $2b^2$ inputs for each set of $2b^3$ floating-point operations. All stores are delayed until all computations for a $b \times b$ block of C are completed. Therefore, $16b^2/2b^3$ bytes of memory bandwidth are required per floating-point operation in 64-bit precision, or $8/b$ bytes/flop. Table 1 gives the memory bandwidth requirements for a few common operations in linear algebra.

The idea of blocking can be used recursively for memory hierarchies with several levels. Blocking at several levels is indeed used in some scientific subroutine libraries, such as the CMSSL [48]. The innermost blocking is based on the size of the register file, the second level is based on DRAM pages, and the third level on minimizing thrashing in Translation Lookahead Buffers, TLB.

Another important class of computations are explicit finite difference methods. The dominating part

Operation	Mem. bandw. Bytes/flop
$y \leftarrow \alpha x$	16
$y \leftarrow \alpha x + y$	12
$dot \leftarrow x^T y$	8
$A \leftarrow \alpha x y^T + A$	8
$y \leftarrow \alpha A x + \beta y$	4
$C \leftarrow \alpha AB + \beta C$	$8/b$

Table 1: Memory bandwidth requirement in 64-bit-precision for full utilization of a floating-point unit with one adder and one multiplier.

Figure 2: The common five-point stencil in two dimensions.

in such computations is often the evaluation of difference stencils at each of the grid points. Grid point values are used as many times as there are points in a stencil. For instance, in case of the common five-point stencil in two dimensions, Figure 2, a grid point value is used in five stencil evaluations.

For two-dimensional stencils, the reduction in memory bandwidth requirements can be accomplished by combining stencils along one axis, often referred to as the *pipeline* axis, into *multistencils*, and by combining multistencils along the other axis, the *sweep* axis, into *swaths* [1, 2]. Figure 3 shows the subgrid that corresponds to a swath of 10 multistencils of width eight each.

For the five-point stencil, the benefit of a multistencil is that, for example, the rightmost point of the leftmost stencil require loading exactly the same source array element as the center point of the next stencil to the right. Computing multistencils along the sweep axis successively can be made by loading only the *leading edge* of the multistencil as it sweeps through the source array along the sweep axis. The set of grid points on the leading edge is denoted by empty circles in Figure 3. The reuse of data increases with the width of the stencil along the sweep axis. In the extreme, almost all source array values are reused as many times as there are points in the stencil. The technique can be extended to stencils in more than two dimensions. In such cases multistencils are constructed along all but

Figure 3: The subgrid corresponding to a swath of 10 multistencils of width eight each for the common five-point stencil in two dimensions.

one axis, which is the sweep axis.

For the FFT using a radix proportional to the size of the local memory allows $5M \log_2 M$ floating-point operations to be performed for the loading of $16M$ bytes of data in 64-bit precision. After the computation this many bytes must be stored as well. A radix- M algorithm yields a byte/flop requirement of $32/(5 \log_2 M)$. Precomputed twiddle factors increase the memory bandwidth requirements by less than a factor of two [23]. The idea of using a radix equal to the size of the available memory was first proposed by Gentleman [9]. Sorting exhibits a behavior similar to the FFT.

The potential benefits of exploiting locality of reference for three typical computations are quantified in Table 2 for relatively large size memories. Table 2 [16] shows the number of operations (and references within a memory of a size given by the column header) per load/store operation to the next level in the memory hierarchy holding larger amounts of data. For matrix multiplication and a block of size 100×100 (local storage about 32k words), the reduction in the need for memory or communications bandwidth is a factor of 100 compared to no locality of reference. For the FFT, the reduction in required interprocessor communication bandwidth is a factor of 14 for a radix-16k algorithm. The higher demands on memory bandwidth of the FFT is very clear. Data for the finite difference

Operation	Local storage (words)					
	Reg.	1k	32k	1M	32M	1G
Matrix Mult.	1	16	90	512	2896	16384
3D-Relaxation	2	19	32	77	307	1024
FFT	1	20	32	45	57	70

Table 2: Number of local operations per remote reference. 3D-Relaxation: 7-point stencil, vector-length 8 ($\alpha = 8$, $\beta = 96$).

operator in Table 2 assumes that the ratio of operations to remote references follow the rule $\frac{1}{\alpha}(\frac{M}{\beta})^{\frac{1}{\gamma}}$ for suitable values of α , β , and γ . With current memory sizes per node in distributed memory systems, exploiting locality reduces the required communication bandwidth by a factor of 8–100 at the node level, and a factor of 80–5000 at the board level.

Given the importance of conserving memory bandwidth it is of interest to determine whether the approaches discussed above indeed are the best possible, or if there may exist some other method that may offer even greater reductions. The block algorithms for matrix multiplication and the high radix FFT are indeed optimal for with respect to memory bandwidth use, as shown by Hong and Kung [12].

In distributed memory systems, memory accesses require a different amount of time depending upon location and the mapping of the index space to memory addresses have significant performance implications. In fact, the mapping of the index space to memory addresses is also important in high performance single processor systems, since memory is not truly random access. In order to discuss locality of reference, the data reference pattern must be considered. Thus, before discussing data mapping issues, we will review a few common data reference patterns.

2.1 Data reference patterns

The data reference pattern clearly depends upon the algorithm that determines how data are combined, and the order in which the data is combined. Explicit methods for solving partial differential equations using finite difference methods on regular grids result in stencil operations, or convolution of grid point values, at all of the grid points. Many stencils (convolution kernels) are naturally expressed through shift operations along the coordinate axes of the grid. Efficient nearest neighbor references along the three coordinate axes is important for performance. For higher order stencils, data references beyond nearest neighbors are required. But, the references are still in some local neighbor in Cartesian space.

Figure 4: Index dependence for odd-even cyclic reduction.

Figure 5: Index dependence for parallel cyclic reduction.

One-dimensional discretizations encountered in, for instance, Alternating Direction Methods, often results in the need for tridiagonal matrix-vector multiplication, or the solution of tridiagonal systems of equations. Parallel methods for the solution of such systems, such as odd-even cyclic reduction or parallel cyclic reduction, rely on a combination of short and long range communications. The data reference patterns are illustrated in Figures 4 and 5, respectively.

In addition to these methods for solving the tridiagonal systems of equations, advantage can also be taken of the fact that, in general, many independent systems shall be solved. Thus, even though the data for each system originally may be distributed, one approach is to move all the data for a system to the same node, with different nodes handling different systems in order to achieve load-balance. The communication pattern in this case may be similar to that of matrix transposition and have the form of *all-to-all personalized* communication [20]. Other algorithms may indeed use *all-to-all broadcast* for the solution of tridiagonal systems.

The finite element computations on unstructured grids require gather/scatter operations. Quadrature coefficients may be stored on one node and broadcast to all other nodes, i.e., *one-to-all broadcast*. In unassembled form, computations on elemental stiffness matrices consists of matrix-vector multiplication, which for high order elements may require both distributed broadcast and reduction. Reduction takes

place along rows of the matrix, while broadcast takes place along the columns. As we will see later, for many data distributions, matrix-vector multiplication implies *all-to-all broadcast* and *all-to-all reduction* among subsets of nodes. Solving the system of equations by a conjugate gradient method involves a global reduction, or *all-to-one reduction*.

It is well known that the FFT requires communication in the form of a butterfly network. However, in the case of performing FFT on distributed memory processors, the communication may actually be better performed as *all-to-all personalized* communication [23]. Other data reference patterns associated with the FFT are bit-reversal, for ordered FFT, and vector-reversal, for real-to-complex FFT.

Many divide-and-conquer methods in higher dimensions require communication in the form of pyramid networks in one or several dimensions. Finally, we mention that naive $N - \text{body}$ algorithms also rely on all-to-all broadcast.

In summary, common algorithms for scientific problems involve data references in the form of

- One-to-all reduction/copy
- All-to-all reduction/copy
- Gather/scatter
- One-to-all personalized communication
- All-to-all personalized communication
- Dimension(index) permutation
- Generalized shuffle permutations
- Scan/parallel prefix
- Lattice emulation
- Butterfly emulation
- Data manipulator network emulation (PM2I network emulation)
- Pyramid network emulation
- Bit-inversion
- Vector reversal ($i \leftarrow N - i$).

Knowledge of the aggregate (global) data reference pattern allows not only for optimal data allocation, but also for optimal scheduling of the data motion between processors in a distributed memory environment. Of the data reference patterns above, we have mentioned all but bit-inversion. This communication is required in certain matrix multiplication algorithms [19] and implies that elements are permuted by complementing all of their address bits.

2.2 Data mapping

Key issues in determining the mapping of data, in particular arrays, are the communication and load-balance it implies for the computations that takes place using it. Preserving locality of reference may yield a substantial reduction in the demands for data references outside of the memory closest to the processor. In the matrix multiplication case, spatial locality is defined by the elements within blocks of shape $b \times b$. In a memory hierarchy, it is clearly desirable that the collection of elements within a block are close also in the address space. In terms of the array index space, spatial locality for most matrix computations are based on distance in a Cartesian space. Many measures of distance are cast in the form $(\sum_{i=0}^d |x_i - y_i|^p)^{\frac{1}{p}}$, where d is the dimensionality of the problem space, and p defines the type of the *norm*. The most common norm in physical space is the 2-norm, which is equal to the Euclidean distance between the two points x and y . Other common norms are the 1-norm, measuring the so-called Manhattan distance, and the ∞ -norm, measuring the maximum distance along any of the coordinate axis. The Manhattan distance is the sum of the absolute values of the coordinate differences.

For matrix computations proximity of indices in a two or three dimensional index space is desirable for many algorithms. For the FFT on the other, it is also true that adjacent memory references suffice, if the index space is viewed as having $n = \log_2 N$ dimensions, where N is the number of data points in the transform. Indeed, all data references are nearest neighbor in such an n -dimensional space.

The examples of matrix multiplication and FFT stress the relevant metric must be defined in discussing locality of reference.

Note that though maximizing locality of reference is highly desirable to reduce the bandwidth demands to the next further removed level of the memory hierarchy, it is exactly opposite to what is required in a banked or interleaved memory. In such a memory, in order to maximize the use of the (local) memory system, successive references should go to *different* banks. In order to limit the data transfer needs between memory systems in different nodes of a distributed memory system, blocks of data shall be mapped to different nodes.

Most programming languages linearize a multidimensional address space through either row or column major ordering. The data mapping used for the linearized address space to a banked or interleaved memory system is optimized for accesses with a stride of one in the linearized address space. The mapping is *cyclic* in that successive indices are mapped to differ-

ent memory banks. Thus, a cyclic mapping applied to a parallel memory system indeed maximizes the need for communication for data accesses with unit stride. On the other hand, a *block* or *consecutive* mapping, maps successive indices to the same bank, thus conserving communication, but resulting in bank conflicts in a banked or interleaved memory system. Formally, the cyclic mapping of N elements to B memory units is

$$\text{Memory unit no. : } j \bmod B, \quad 0 \leq j < N$$

$$\text{Local address : } \lfloor \frac{j}{B} \rfloor, \quad 0 \leq j < N$$

If B is a power-of-two, i.e., $B = 2^k$ for some $k > 0$, then the memory unit address is simply the k least significant bits. For the consecutive mapping, typically the maximum size of the local data set is computed first, then this many elements are assigned to as many memory units as possible, followed by one memory unit that receives less data and a few units that receive no data, unless $N \bmod B = 0$. Thus, for the consecutive allocation of one-dimensional arrays we have:

$$\text{Memory unit no. : } \lfloor j / \lceil \frac{N}{B} \rceil \rfloor, \quad 0 \leq j < N$$

$$\text{Local address : } j \bmod \lceil \frac{N}{B} \rceil, \quad 0 \leq j < N$$

We have already remarked that cyclic assignment is a very poor choice with respect to locality of reference when nearest neighbor communication is desired. In fact, for an operation that requires communication between adjacent nodes of a one-dimensional array, such as tridiagonal matrix-vector multiplication, cyclic allocation maximizes the number of nonlocal references. This is apparent from Figure 6. For m elements per memory unit $2m$ remote references are required. On the other hand, in a consecutive allocation only two nonlocal references are required, see Figure 7. The two allocation schemes are shown in Figure 8, where the numbers indicate memory unit number.

Though the cyclic mapping is undesirable for nearest neighbor references on one-dimensional arrays, a cyclic mapping of a two-dimensional array linearized either in row or column major order allow the references along one axis to be entirely local. Evaluating five-point stencils for all elements of an 80×80 array mapped to eight memory banks through a linearized address space mapped cyclicly to memory addresses, results in a total of $2 \times 80 \times \frac{80}{8} = 1600$ nonlocal references for data associated with each memory unit (assuming periodic boundary conditions). For a consecutive partition of both axes of the data array, the set of

Memory unit							
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26					

Figure 6: Cyclic allocation of a one-dimensional array of 27 elements to eight memory units.

Memory unit							
0	1	2	3	4	5	6	7
0	4	8	12	16	20	24	
1	5	9	13	17	21	25	
2	6	10	14	18	22	26	
3	7	11	15	19	23		

Figure 7: Consecutive allocation of a one-dimensional array of 27 elements to eight memory units.

memory units can be viewed as configured as a 2×4 array, resulting in subarrays of shape 40×20 assigned to each memory unit. With such a partitioning the number of nonlocal references for the data in each memory units is $2 \times (20 + 40) = 120$, a reduction by a factor of $40/3$. In general, for an equal number of nearest neighbor references in both directions of all coordinate axes, the number of remote references are minimized when the surface area for a given subset of the index space is minimized. This is known as the “surface-to-volume” effect.

Some computations, such as LU or QR factorization of dense matrices, and triangular system solution does not involve nearest neighbor communication in a Cartesian space. The only required communication is one-to-all broadcast and all-to-one reduction. These communications are not effected by a cyclic or consecutive mapping of array data to memory units. However, the load-balance is affected, since the set of indices of elements subject to updates decreases during the course of the computations. A cyclic allocation guarantees good load-balance. But, a good load-balance can be achieved also for consecutive mapping by ad-

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
Consecutive																											
0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	
Cyclic																											

Figure 8: Consecutive and cyclic assignment of a one-dimensional array to memory units.

justing the elimination order accordingly [28].

A multi-dimensional address space is highly desirable for parallel processing, in order to reduce the demands on the communication system for many common algorithms and data structures. High Performance Fortran [7] addresses this issue by allowing the programmer to specify consecutive, cyclic, or any combination of these allocation schemes independently for each of the axis. Thus, HPF supports a multidimensional address space, as well as a means for trading load-balance and locality of reference against each other.

Partitioning a large unstructured grid into subgrids such that locality is maximized and remote references minimized is a very difficult problem. Two general partitioning techniques of significant recent interest are the recursive spectral bisection technique proposed by Pothen et. al. [36] and the geometric approach proposed by Miller et. al. [33, 32, 46]. The recursive spectral bisection technique has been used successfully by Simon [41] for partitioning of finite volume and finite element meshes. A parallel implementation of this technique has been made by Johan [13].

The spectral partitioning technique is based on the eigenvector corresponding to the smallest nonzero eigenvalue of the Laplacian matrix associated with the graph to be partitioned. The Laplacian matrix is constructed such that the smallest eigenvalue is zero and its corresponding eigenvector consists of all ones. The eigenvector associated with the smallest nonzero eigenvalue is called the *Fiedler vector* [4, 5, 6]. Grid partitioning for finite volume and finite element methods is often based on a dual mesh representing finite volumes or elements and their adjacencies (or some approximation thereof) rather than the graph of nodal points. The reason for using a volume or element based graph is that the computations are naturally organized as volume or elementwise computations. These computations exhibit locality of reference within the volumes or elements and can often be performed as a (large) collection of dense matrix operations. Communication is required when passing data between the global representation, and the representation of the collection of local elements [25, 31]. The purpose of the partitioning is to minimize this communication.

For finite element computations, the adjacency in applying the spectral bisection method has been approximated by elements that share faces. This adjacency accurately represents the communication requirements for finite volume methods. However, in finite element methods, communication is also required between elements sharing edges and corners.

Thus, for finite element and finite volume computations the graph to be partitioned have nodes rep-

No. of partitions	No. of shared edges	% of total	No. of shared nodes	% of total
8	188	0.8	195	2.4
16	381	1.6	396	4.8
32	752	3.1	773	9.3
64	1483	6.0	1479	17.8
128	2154	8.8	2101	25.3

Table 3: Partitioning of a planar mesh with inner boundary in the form of a double ellipse.

representing individual elements or volumes, and edges representing shared faces. Based on this partitioning, a partitioning of the set of nodal values is carried out. Nodal points internal to a partition are mapped to the processing node to which the partition is assigned. Boundary nodes must be assigned to one of the partitions among which they are shared, or replicated among the partitions among which they are shared. Only boundary nodes require communication.

One advantage of the spectral bisection technique is that it is based on the topology of the graph underlying the sparse matrix. It requires no geometric information. However, it is computationally quite demanding. The geometric partitioning technique by Miller et. al. holds promise to be computationally less demanding than the spectral decomposition technique, but relies on geometric information and geometric properties of the graph [34, 35]. Geometric information is typically available for meshes generated for the solution of partial differential equations, but may not be present in other applications. Since we do not yet have much experience with the geometric partitioning technique we only report the results from the spectral bisection technique.

The results of applying the spectral bisection technique to two model problems are reported in [13, 14] and shown in Tables 3 and 4. One of the model problems consists of a planar triangular mesh between an outer ellipse and an inner double ellipse. The other problem is a grid of tetrahedra between concentric cylinders. The planar grid has 8,307 nodes, 16,231 triangles, and 24,537 edges. The numbers of shared nodes and edges as a function of the number of partitions are given in Table 3. The grid for the concentric spheres consists of 20,374 nodes, 107,416 tetrahedra, and 218,807 faces.

The results of applying the spectral bisection technique in a more realistic finite element application [15] are summarized in Table 5. The spectral bisection technique in this example offered a reduction in the number of remote references by a factor of 13.2. The

No. of partitions	No. of shared edges	% of total	No. of shared nodes	% of total
8	5186	2.4	2735	13.4
16	8005	3.7	4095	20.1
32	11553	5.3	5747	28.2
64	16055	7.3	7721	37.9
128	21502	9.8	9827	48.2

Table 4: Partitioning of a tetrahedral mesh between concentric spheres.

Operation	Standard allocation	Spectral bisection
Partitioning	—	66
Gather	298	23
Scatter	445	46
Computation	180	181
Total time	923	316

Table 5: Gather and scatter times in seconds on a 32-node CM-5 for 50 time steps with a 1-point integration rule for finite element computations on 19,417 nodes and 109,914 elements.

speedup for the gather operation was a factor of 13 and of the scatter operation the speedup was a factor of 9.6 (the scatter operation includes the time required for addition which is unaffected by the partitioning).

Another important aspect of computations with arbitrary sparse matrices is that unlike for dense and grid sparse matrices, address computations cannot be performed by incrementing addresses using fixed strides. For arbitrary sparse matrices, indirect addressing is required. The address computation may be fairly complex. It frequently is the most time consuming part on uniprocessors. On a distributed memory machine, the address computations do not only involve the computation of local addresses, but routing information as well. In an iterative (explicit) method, the underlying grid may be fixed for several or all iterations. For such computations it is important with respect to performance to amortize the cost of computing the addresses over as many iterations as possible. Cacheing this information and reusing it later is important for performance [48].

In an arbitrary sparse matrix, there is no simple way of encoding the global structure. Yet, arbitrary sparse matrices may still have some local structure resulting in a block sparse matrix. Taking advantage of such a block structure for both economy in data representation, data storage and efficiency of operations,

is significantly simplified by explicitly representing the blocks [48].

2.3 Allocation of aggregates

The consecutive and cyclic distribution schemes and the dimensionality of the address space for regular and grid sparse arrays, and spectral or geometric partitioning for arbitrary sparse matrices and irregular grids, determines which elements are grouped together for a node. The data reference pattern determines the communication needs of each node.

The total demand on the communication system is not only affected by how the data is grouped together, but also how the groups are assigned to nodes. Ideally, the groups of data are allocated to the nodes such that the contention is minimized. To accomplish this task, both the *data reference pattern* and the *network topology* must be taken into account, as well as the *routing* scheme. Optimal allocation of data is a hard problem. Moreover, the allocation may need to be dynamic to efficiently accommodate different phases of a computation.

2.3.1 Allocation of regular arrays

For nearest neighbor references along data array axes, such as for typical difference stencils on regular grids, proximity preserving data array embeddings of array partitions to nodes are ideal, if such embeddings exist. Such embeddings do exist in nodal arrays of a dimensionality that is at least as high as that of the data array [39]. Thus, two-dimensional mesh configured machines can embed meshes of at most two dimensions preserving adjacency, while three-dimensional meshes of processing nodes allows adjacency preserving embeddings for grids of up to three dimensions. Binary cube networks of nodes allow for adjacency preserving embeddings of meshes with a dimensionality up to the number of cube dimensions. On the Connection Machine system CM-200 [47] which have processing nodes interconnected by a binary cube network, the default embedding of data arrays preserve adjacency.

Though data arrays can be embedded preserving adjacency in meshes with sufficient dimensionality, such embeddings are not possible in some other common networks, such as butterfly networks, and binary tree networks [40].

2.3.2 Allocation of irregular arrays

For irregular grids, mapping of groups of data to processing nodes such that proximity is preserved, is a much more difficult problem than the mapping of regular arrays. Instead of attempting to find the best

Problem	Gather		Scatter	
	standard alloc.	random alloc.	standard alloc.	random alloc.
3200 20-node brick elements	75	50	124	55
864 8-node brick elements	5.6	3.7	7.2	3.4

Table 6: The effect of randomization on gather and scatter performance. Times in msec on an 8K CM-200.

possible map, it may be more profitable to search for a map that is guaranteed to have an acceptable worst case behavior. A randomized data placement [38, 37] reduces the risk for bottlenecks in the routing system. The randomized placement of data achieves the same communication load characteristics in a single (deterministic) routing phase as randomized routing achieves in two phases [49, 50]. Randomized placement is an option for some functions on the Connection Machine systems [48].

The Connection Machine system CM-200 does not support randomized routing. The routing on the CM-5 randomizes the path selection on the message path towards the root in its fat-tree interconnection network [26, 27], followed by a deterministic path from the point of turnaround to the destination.

Figures 9 and 10 give examples of the performance improvements achieved on the CM-2 through the use of randomized data allocation in a finite element computation on an unstructured grid. The horizontal axis shows the number of degrees of freedom and elements, while the vertical axis denotes the execution time. Each element has 24 degrees of freedom. The performance improvement for the gather instruction due to randomization is in the range 2.1 - 2.4. The improvement is increasing with the problem size. Figure 10 shows the execution times for two methods of accumulating the product vector: using the combining features of the router, and accumulation after the routing operation. Randomization of the addresses improved the router combining time by about a factor of two, but performing the routing without combining is even more effective. Table 6 gives the gather scatter times with and without randomization for a solid mechanics application [31] on the CM-200. The performance enhancement is a factor of 1.5 - 2.2, which in our experience is typical. It is rarely the case that randomization has caused a performance degradation.

Figure 9: Gather with binary and randomized addresses. 8K CM-2.

Figure 10: Accumulation of vector elements. Binary and randomized addresses. 8K CM-2.

2.4 A summary of data allocation issues

The assignment of data to memory units affects load-balance, communication requirements, network contention, and the performance of the computations in each node (by affecting the ability to carry out local blocking and pipelining of operations).

- Consecutive distribution preserves locality of reference along data array axes, and is suitable for stencil-like reference patterns. It also offers the possibility to improve the efficiency of the operations in each node by increasing the chance for good cache behavior through optimal blocking, and through long vectors for pipelined processors.
- Cyclic distribution significantly increases the communication requirements for relaxation methods and explicit methods for the solution of partial differential equations, and shall be avoided. Cyclic distribution may offer a reduction in the communication requirements for the FFT by a factor of two [23]. Cyclic distribution is *not* required for load-balance in LU and QR factorization, or for the solution of triangular systems of equations.
- Grid sparse problems can be represented as dense matrix problems. The partitioning techniques for such problems yield good locality of reference for typical operations on regular grids and grid sparse matrices
- Arbitrary sparse matrices and irregular grids can be successfully partitioned into subdomains using recursive spectral bisection and geometric partitioning.
- An optimum distribution of partitions requires a data dependence analysis and an understanding of optimum embeddings and routing algorithms for the network at hand. For irregular computations, and when proximity preserving embeddings may not be possible, minimizing the contention through randomized distribution has shown to be effective.

Finally, we remark that data distributions cannot be determined until run-time, not only because the index sets may not be known until run-time, but it is highly desirable not to constrain the number of processors used for execution at compile time. Library routines must always yield a correct result. Library routines must also offer as good a performance as possible, given the specified data distributions of the input and output objects. The decisions of what algorithm to choose for a given function, and whether or not a

redistribution shall be performed before and after executing the function, must be made at run-time.

Though mapping of data to memory units such that proximity is preserved often is a good idea, it does not necessarily guarantee that the communication time is minimized. The reason is that in some interconnection networks the available bandwidth between a pair of nodes may increase faster than the required bandwidth due to increased distance. A greater distance may also reduce the likelihood of network contention. Thus, though minimizing distance is often the best with respect to communication time, in particular under light load, it may not be the best under heavy load when bandwidth rather than distance governs the performance of the interconnection network. Randomization of the routing, or the address map, may be used to reduce the likelihood for severe congestion in the communication system.

3 Data Motion

Below we will discuss some of the techniques used to achieve high utilization of the communications bandwidth in binary cube networks. Full bandwidth utilization is achieved by using multiple embeddings.

3.1 Broadcast

Broadcast and reduction from a single source to subsets of nodes, holding an entire row or column, are critical for the efficiency of computations such as LU and QR-factorization. In fact, many concurrent broadcast (and reduction) operations are desired in these computations as illustrated in Figure 11. Whether or not these broadcast operations imply communication that interfere, depends upon the network topology and how the index space is mapped to the nodes.

On a binary cube network, entire subcubes are often assigned to a data array axis. In such a case, the broadcasts within the different columns in the index space do not interfere with each other, and the concurrent broadcast operation degenerates to a number of broadcasts within disjoint subsets of nodes. However, in other networks, such as the fat-tree, such a data mapping may not be feasible, and the simultaneous broadcast from several sources to distinct subsets of nodes may require a more complex routing for optimal bandwidth utilization. For instance, with a two-dimensional address space linearized with respect to processor addresses, implies that broadcast within rows can take place without interference between rows (for row major ordering), on the CM-5 fat-tree, since they are mapped to separate subtrees. However, for

Figure 11: Broadcasting of a pivot row in LU-decomposition.

broadcast within columns contention occurs since the CM-5 fat-tree exhibits a reduction in bi-section width for the first two levels.

Global broadcast and reduction is used, for instance, in the conjugate gradient method. Even for the conjugate gradient method, several simultaneous broadcast operations may be required due to multiple-instance computation.

In implementing a broadcast algorithm it is important to exploit the bandwidth. A simple spanning tree may not accomplish this task. Consider a binary cube network (used in the CM-2 and CM-200). On an n -cube, using a binomial tree to broadcast M elements from a node to all other nodes requires a time of nM with the communication restricted to one channel at a time. The time is proportional to M with concurrent communication on all channels of every node, *all-port* communication. However, the lower bounds for the two cases are M and $\frac{M}{n}$, respectively [20]. Thus, the binomial tree algorithm is nonoptimal by a factor of n in both cases.

Multiple spanning trees rooted at the same node can be used to create lower bound algorithms [20]. The basic idea is that the source node splits its data set into $\frac{M}{n}$ disjoint subsets and sends each subset to a distinct neighbor. Then, each of these neighbor nodes broadcasts the data set it received to all other nodes (except the original source node) using spanning binomial trees. By a suitable construction of the trees, the n binomial trees are edge disjoint, and the full bandwidth of the binary n -cube is used effectively.

The multiple spanning binomial tree algorithm is used for broadcasting on the Connection Machine systems CM-2 and CM-200. This broadcast function is part of the Connection Machine Run-Time System. The performance is illustrated in Figure 12 [17]. As expected, the time to broadcast a given size data set

Figure 12: Time in msec for broadcast of 16K 32-bit data elements on Connection Machine system CM-200 as a function of the number of cube dimensions.

decreases with the number of nodes to which the set is broadcast. Thus, broadcast exhibits a logarithmic speedup with respect to the number of nodes.

The CM-5 communication system is of the node limited variety and the optimizations are different.

3.2 All-to-all broadcast/reduce

Another important communication primitive is the simultaneous broadcast from each node in a set to every other node in the same set, *all-to-all broadcast* [20]. This communication is typical for so called direct N -body algorithms, but it is also required in many dense matrix algorithms. 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. Here we will illustrate their use in matrix-vector multiplication.

With the processing nodes configured as a two-dimensional nodal array for the matrix, and 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 13 illustrates the data allocation for both row major and column major ordering of the matrix. The data allocation shown in Figure 13 is typical on Connection Machine systems.

For a matrix of shape $P \times Q$, allocated to a two-dimensional nodal array in column major order, an all-to-all broadcast [8, 20, 42, 43] 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, resulting in a row major ordering of y . But, the node labeling is in column major order, and a reordering from row to column major ordering is required in order to establish the final allocation of y . Thus, for a column major order of the matrix elements, matrix-vector multiplication can be expressed as:

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

All-to-all broadcast or reduction is required also when a one-dimensional nodal array configuration is used for the matrix [29].

As in the case of broadcast from a single source, all-to-all broadcast on an n -cube can be performed in a time proportional to the lower bound. With each node initially holding M elements, a time of $M(N-1)$ is required for communication restricted to a single port at a time, and a time of $\frac{M(N-1)}{n}$ is required for all-port communication [20]. A simple, yet optimal, all-port algorithm for all-to-all broadcast uses n Hamiltonian paths for each node. For all-to-all broadcast, the Hamiltonian paths need not be edge-disjoint [20, 3].

3.3 Gather/Scatter

Gather and scatter operations on regular grid data represented as one or multidimensional arrays, as well as irregular grid data, is critical for the performance of many scientific and engineering applications. On the Connection Machine Systems, gather and scatter operations on regular grids are supported through PSHIFT (for polyshift) [10, 48], which allows the programmer to specify concurrent shift operations in one or both directions of one or multiple axes. On the CM-2 and CM-200 with Gray-coded axes, PSHIFT concurrently performs all the data exchanges requiring communication between nodes. In effect, PSHIFT provides an effective means of emulating lattices on binary cubes. (A further level of optimization is provided by the so called stencil compiler [2], which in addition to

Figure 13: Data allocation on a rectangular nodal array.

Problem	Gather		Scatter	
	standard alloc.	random alloc.	standard alloc.	random alloc.
3200 20-node brick elements	75	50	124	55
864 8-node brick elements	5.6	3.7	7.2	3.4

Table 7: The effect of randomization on gather and scatter performance. Times in msec on an 8K CM-200.

maximizing the concurrency in internode communication (using PSHIFT), avoids unnecessary local memory moves and uses a highly optimized register allocation in order to minimize the number of load and store operations between local memory and the register file in the floating-point unit.)

For gather and scatter operations on unstructured grid computations, the general router on the Connection Machine systems is used. However, two means of improving the performance are provided; randomization of the data allocation, and savings of the routing information for repetitive gather/scatter operations. Table 7 [30] summarizes the effect of randomization of the data allocation for a few meshes.

The performance enhancement of 1.5 – 2.25 is typical.

3.4 Personalized communication

In *one-to-all personalized communication*, a node sends a unique piece of data to every other node. An example is matrix computations where a node holds an entire column, which may need to be redistributed

evenly over all the nodes as in some algorithms for matrix-vector multiplication [29]. In all-to-all personalized communication, each node sends unique information to all other nodes. Personalized communication is not limited to matrix transposition, but encompasses operations such as bit-reversal, transposition or bit-reversal combined with a code change (such as the conversion between binary code and binary-reflected Gray code), and bit-inversion. We now illustrate the significance of personalized communication in computing the FFT on a multiprocessor.

In computing the FFT on distributed data, one possibility is to exchange data between nodes and have one of the nodes in a pair compute the “top” and the other compute the “bottom” of the butterfly requiring data from the two nodes. This type of algorithm is used on the Connection Machine systems CM-2 and CM-200 [24]. When there are two or more elements per node, then an alternative is to perform an exchange of data between nodes, such that each node in a pair computes one complete butterfly. The sequence of exchanges required for the FFT amounts to a shuffle, as illustrated below, where the | separates node address bits to the left and local memory address bits to the right:

Example 1.

Address	Index
(54321 0)	(54321 0) = (54321 x ₀)
(04321 5)	(x ₀ a ₄ a ₃ a ₂ a ₁ a ₅)
(05321 4)	(a ₄ x ₀ a ₃ a ₂ a ₁ a ₅)
(05421 3)	(a ₄ a ₃ x ₀ a ₂ a ₁ a ₅)
(05431 2)	(a ₄ a ₃ a ₂ x ₀ a ₁ a ₅)
(05432 1)	(a ₄ a ₃ a ₂ a ₁ x ₀ a ₅)

$$(15432|0) \quad (a_4 a_3 a_2 a_1 a_5 | x_0)$$

Thus, the end result of the sequence of exchanges is a shuffle on the node address field. Each step is equivalent to the transposition of a collection of 2×2 matrices.

In practice, for a one-dimensional transform, there are typically several local memory bits. For performance, under many models for the communication system, minimizing the number of exchange steps is desirable, i.e., instead of performing bisections it is desirable to perform multisections including all local memory bits. Thus, for instance, with two local memory bits, four-sectioning should be carried out, as shown in Example 2. Multisectioning is used for the FFT on the CM-5.

Example 2.

Address	Index
$(65432 10)$	$(65432 10) = (65432 \underbrace{x_1}_{j_1} \underbrace{x_0}_{j_0})$
$(10432 65)$	$(\underbrace{x_1 x_0}_m a_4 a_3 a_2 \underbrace{a_6 a_5}_m)$
$(10652 43)$	$(\underbrace{a_4 a_3}_m \underbrace{x_1 x_0}_m \underbrace{a_2}_{n-k \cdot m} \underbrace{a_6 a_5}_m)$
$(10654 23)$	$(\underbrace{a_4 a_3 a_2}_{n-m} \underbrace{x_0}_{(k+1)m-n} \underbrace{x_1}_{n-k \cdot m} \underbrace{a_6 a_5}_m)$
$(23654 10)$	$(\underbrace{a_4 a_3 a_2}_{n-m} \underbrace{a_6}_{(k+1)m-n} \underbrace{a_5}_{n-k \cdot m} \underbrace{x_0 x_1}_m)$
$(23654 10)$	$(\underbrace{a_4 a_3 a_2}_{n-m} \underbrace{a_6}_{(k+1)m-n} \underbrace{a_5}_{n-k \cdot m} \underbrace{x_1 x_0}_m)$

Example 2 was deliberately chosen such that the exchanges cannot simply be treated as digit exchanges with increased radix for the digit, but must indeed be treated as exchanges with digits of different radices. Moreover, the last few exchange steps were made such that the final order represents an m -step shuffle on the nodal address bits, where m is the number of bits used to encode the first exchange. This node address ordering requires a local memory shuffle to restore the original local memory ordering. (In practice, it may, in fact, be preferable to avoid the local memory reordering by performing the last exchange such that local memory is normally ordered, which would leave the node addresses in an order corresponding to two shuffles: one m -step shuffle on all n node address bits, one $n \bmod m$ shuffle on the last m bits.)

In multidimensional FFT, all of local memory should be considered in performing the multisectioning, see [18].

The FFT produces the results in bit-reversed order with respect to the indices. Thus, establishing a normal index map in the output domain requires an unshuffle with bit-reversal. Figure 14 [18] shows an example.

The lower bounds for all-to-all personalized communication depends upon the network and communication system. For a binary n -cube with M data elements per node, the lower bound is $\frac{nMN}{2}$ for communication restricted to a single port per node and $\frac{MN}{2}$ for all-port communication [20]. Balanced spanning trees [11] provide for optimal all-to-all personalized communication with all-port communication. A balanced spanning tree has N/n nodes in each of the n subtrees of the root. The use of n rotated spanning binomial trees rooted in each node also yields the desired complexity [20]. The CM-5 network is node limited, and the communication time is simply limited by the bandwidth at the node. However, the order in which the data is sent and retrieved from the network has a measurable impact on the performance.

In our FFT example above, several all-to-all personalized communications were performed in succession. In such a case, it may be of interest to minimize the time elements spend in transition from source to destination in order to minimize pipeline delays. Algorithms with a minimal transition time are presented in [22].

Bit-reversal with an equal number of dimensions assigned to the node address field and the local memory address field constitutes one form of all-to-all personalized communication. The performance on various sizes of the CM-2 is shown in Figure 15 [17]. As expected, the execution time is almost independent of the machine size for a fixed size data set MN . The increase in the execution time is largely due to the fact that local memory operations cannot be performed in parallel. Thus, there is a term proportional to n in addition to the constant term.

3.5 Index reversal – bit-inversion

Index reversal is another important permutation used for instance in the computation of real-to-complex FFT. For this computation, the standard algorithm requires that data with indices i and $N - i$, $0 \leq i < N$ be operated upon in a preprocessing or postprocessing step for the FFT [44, 45]. In binary-coded data, the index reversal required for the FFT corresponds to a two's-complement subtraction (bit complement plus one).

Figure 14: Two step reordering after 4-section based radix-2 FFT. First step, bit-exchange between nodes; second step, bit-exchange between local memory and node addresses. $p \leq 2n - m$.

Figure 15: All-to-all personalized communication on the Connection Machine.

However, in the case of the real-to-complex FFT on a one-dimensional array with binary-coded data, the first step in one of the most common algorithms is to perform a complex-to-complex FFT on the array viewed as a half-size, one-dimensional array of complex data points. The result is shown in Figure 16. The Figure also shows that the postprocessing matching indices i and $N - i$ correspond to bit-inversion in subcubes of the form $00 \dots 01xx \dots x$, with the inversion being performed on the bits denoted by x .

If there are more than one complex data point per node, then the communication requirements depend upon how the indices are aggregated to the nodes. In consecutive data allocation, the communication pattern between nodes is the same as if there was only one element per node. In a cyclic data allocation, the communication for the first complex local memory location across all nodes is the same as if there was a single element per node. The communication for the second and all subsequent complex local memory locations is bit-complement on the entire node address.

Bit-inversion also occurs in the alignment of the operands in matrix-matrix multiplication on three-dimensional nodal array configurations [19].

Concurrent communication for bit-inversion on binary cubes is straightforward. For instance, multiple exchange sequences starting in different dimensions and progressing through the dimensions in increasing (or decreasing) order cyclicly can be used.

Figure 16: Postprocessing for real-to-complex FFT. Bit-inversion in subcubes.

3.6 Shuffle operations on binary cube networks

Computing the FFT through multisectioning results in an m -step shuffle on the index space, where m is the number of bits encoding the first digit exchange. Restoring the original index order corresponds to an unshuffle (except for the FFT which in itself implements a bit-reversal). Reshaping the nodal array for a given data array also represents a general shuffle operation. For instance, changing the allocation

$$(y_2^{a_3} y_1^{a_2} y_0^{m_2} x_3^{a_1} x_2^{a_0} x_1^{m_1} x_0^{m_0})$$

to the allocation

$$(y_2^{m_2} y_1^{m_1} y_0^{m_0} x_3^{a_3} x_2^{a_2} x_1^{a_1} x_0^{a_0}),$$

where x_i and y_i denote bits encoding an x -axis and y -axis, respectively, and a_i denotes nodal address bits and m_i local memory address bits, constitutes a generalized shuffle, or dimension permutation. The dimension permutation is: $a_3 \leftarrow a_1 \leftarrow m_1 \leftarrow a_2 \leftarrow a_0 \leftarrow m_0 \leftarrow m_2 \leftarrow a_3$. In this example, the reshaping resulted in a single cycle on the dimensions. In general, the reshaping may result in several cycles, just as the m -step shuffle in general can be decomposed into several cycles.

A shuffle can be implemented as a sequence of successive pairwise dimension exchanges starting in any position. In a binary cube, such exchanges imply communication in two cube dimensions for each step, when

both dimensions in an exchange are nodal address dimensions. However, it is also possible to use a fixed memory dimension for each exchange. If the first exchange is repeated as a last exchange, then the result is a shuffle on all bits but the fixed exchange dimension. For a shuffle on n bits, the first alternative requires $n - 1$ exchanges while the last requires $n + 1$ exchanges. Thus, at the expense of two additional exchanges, each exchange only involves one nodal address dimension. In [21] we present algorithms that are nonoptimal by two exchanges at most, regardless of the number of cube dimensions in the shuffle and the data elements per node. The algorithms use multiple exchange sequences (embeddings), exploiting the fact that a shuffle can be performed as a sequence of exchanges starting at any bit and proceeding in order of decreasing dimensions cyclicly.

4 Summary

We have shown that for current memory sizes in a node in distributed memory architectures preserving locality of reference may reduce the demand for network bandwidth by one to two orders of magnitude for common computations. Mapping data arrays for matrix computations, relaxation methods, explicit finite difference methods, and convolution by decomposition into blocks with sides of approximately equal length is indeed optimal or close to optimal for many of the common operations in matrix algebra and explicit methods for partial differential equations. Such allocations are easy to determine, but require a multidimensional address space. Conventional languages linearize the address space, but the recently proposed High Performance Fortran provides multidimensional addresses. Partitioning of irregular grids to preserve locality of reference is much more difficult. We have used recursive spectral bisection with excellent results.

For relatively coarse grained computations, which is the typical case for current massively parallel computers, using routing techniques that maximize bandwidth utilization is often more critical than minimizing latency. In some networks, such as binary cubes, there exists multiple paths between any source/destination pair. We have discussed some algorithms that exploit this fact by routing messages along multiple paths without congestion through proper scheduling. In fact, the bandwidth is used optimally for several of the algorithms.

Acknowledgement

Many people have participated in the work discussed here. Many thanks go to Ching-Tien Ho, Abhiram Ranade, Michel Jacquemin, Kapil K Mathur, Zdenek Johan, Doug MacDonald, Jean-Philippe Brunet, Robert Krawitz, and Palle Pedersen.

The work discussed in this paper directly or indirectly has been supported by the Office of Naval Research under contracts N00014-84-K-0043 and N00014-86-K-0564, and by the Air Force Office for Scientific Research under contract AFOSR-89-0382, and the National Science Foundation and the Defence Research Project Agency (now ARPA) under contract CCR-8908285.

References

- [1] Ralph G. Brickner, William George, S. Lennart Johnsson, and Alan Ruttenberg. A stencil compiler for the Connection Machine models CM-2/200. In H.J. Sips, editor, *Fourth International Workshop on Compilers for Parallel Computers*, pages 68–78. Delft University of Technology, Faculty of Applied Physics, 1993.
- [2] M. Bromley, Steve Heller, Tim McNerny, and Guy Steele. Fortran at ten Gigafllops: The Connection Machine convolution compiler. In *Proceedings of ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*. ACM Press, June 1991.
- [3] Jean-Philippe Brunet and S. Lennart Johnsson. All-to-all broadcast with applications on the Connection Machine. *International Journal of Supercomputer Applications*, 6(3):241–256, 1992.
- [4] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23:298–305, 1973.
- [5] M. Fiedler. Eigenvectors of acyclic matrices. *Czechoslovak Mathematical Journal*, 25:607–618, 1975.
- [6] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25:619–633, 1975.
- [7] High Performance Fortran Forum. High performance fortran; language specification, version 1.0. *Scientific Programming*, 2(1 - 2):1–170, 1993.
- [8] Geoffrey C. Fox, Mark A. Johnson, Gregory A. Lyzenga, Steve W. Otto, John K. Salmon, and David W. Walker. *Solving Problems on Concurrent Processors*. Prentice-Hall, 1988.
- [9] W. Morven Gentleman and G. Sande. Fast Fourier transforms – for fun and profit. In *Proceedings – Fall Joint Computer Conference, 1966*, pages 563–578. AFIPS, 1966.
- [10] William George, Ralph G. Brickner, and S. Lennart Johnsson. Polyshift communications software for the Connection Machine systems CM-2 and CM-200. *Scientific Programming*, 3(1):83–99, Spring 1994.
- [11] Ching-Tien Ho and S. Lennart Johnsson. Spanning balanced trees in Boolean cubes. *SIAM Journal on Sci. Stat. Comp.*, 10(4):607–630, July 1989.
- [12] J.W. Hong and H.T. Kung. I/O complexity: The red-blue pebble game. In *Proc. of the 13th ACM Symposium on the Theory of Computation*, pages 326–333. ACM, 1981.
- [13] Zdenek Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics*. PhD thesis, Department of Mechanical Engineering, Stanford University, 1992.
- [14] Zdenek Johan and Thomas J. R. Hughes. An efficient implementation of the spectral partitioning algorithm on the connection machine systems. In *International Conference on Computer Science and Control*. INRIA, 1992.
- [15] Zdenek Johan, Kapil K Mathur, S. Lennart Johnsson, and Thomas J.R. Hughes. An efficient communication strategy for Finite Element Methods on the Connection Machine CM-5 system. *Computer Methods in Applied Mechanics and Engineering*, 113:363–387, 1994.
- [16] S. Lennart Johnsson. *Models of Massively Parallel Computation*, chapter 4, pages 223–389. Morgan Kaufmann, 1990.
- [17] S. Lennart Johnsson. Performance modeling of distributed memory architectures. *J. Parallel and Distributed Computing*, 12(4):300–312, August 1991.
- [18] S. Lennart Johnsson. Data ordering in multisection FFT. Technical report, Thinking Machines Corp., 1992. In preparation.
- [19] S. Lennart Johnsson. Minimizing the communication time for matrix multiplication on multiprocessors. *Parallel Computing*, 19(11):1235–1257, 1993.

- [20] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.
- [21] S. Lennart Johnsson and Ching-Tien Ho. Generalized shuffle permutations on Boolean cubes. *J. Parallel and Distributed Computing*, 16(1):1–14, 1992.
- [22] S. Lennart Johnsson and Ching-Tien Ho. Optimal communication channel utilization for matrix transposition and related permutations on Boolean cubes. *Discrete Applied Mathematics*, 1992.
- [23] S. Lennart Johnsson, Michel Jacquemin, and Robert L. Krawitz. Communication efficient multi-processor FFT. *Journal of Computational Physics*, 102(2):381–397, October 1992.
- [24] S. Lennart Johnsson and Robert L. Krawitz. Cooley-Tukey FFT on the Connection Machine. *Parallel Computing*, 18(11):1201–1221, 1992.
- [25] S. Lennart Johnsson and Kapil K. Mathur. Data structures and algorithms for the finite element method on a data parallel supercomputer. *International Journal of Numerical Methods in Engineering*, 29(4):881–908, 1990.
- [26] Charles E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Computers*, 34:892–901, October 1985.
- [27] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Cral R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A St Pierre, David S. Wells, Monica C. Wong, Shaw-Wen Yang, and Robert Zak. The network architecture of the Connection Machine CM-5. In *SPAA '92*, pages 272–285. ACM Press, 1992.
- [28] Woody Lichtenstein and S. Lennart Johnsson. Block cyclic dense linear algebra. *SIAM Journal of Scientific Computing*, 14(6):1257–1286, 1993.
- [29] Kapil K. Mathur and S. Lennart Johnsson. All-to-all communication. Technical Report 243, Thinking Machines Corp., December 1992.
- [30] Kapil K. Mathur and S. Lennart Johnsson. Communication primitives for unstructured finite element simulations on data parallel architectures. *Computing Systems in Engineering*, 3(1 – 4):63–72, December 1992.
- [31] Kapil K. Mathur, Alan Needleman, and V. Tvergaard. Dynamic 3-d analysis of the charpy v-notch test. *Modelling and Simulation Materials Science and Engineering*, 1(4):467–484, July 1993.
- [32] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Automatic mesh partitioning. In *Sparse Matrix Computations: Graph Theory Issues and Algorithms*. The Institute of Mathematics and its Applications, 1992.
- [33] Gary L. Miller, Shang-Hua Teng, and Stephen A. Vavasis. A unified geometric approach to graph separators. In *32nd Annual Symposium on Foundations of Computer Science*, pages 538–547. IEEE Computer Society, 1991.
- [34] Gary L. Miller and William Thurston. Separators in two and three dimensions. In *Proceedings of the 22th Annual ACM Symposium on the Theory of Computing*, pages 300–309. ACM Press, 1990.
- [35] Gary L. Miller and Stephen A. Vavasis. Density graphs and separators. In *Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 331–336. ACM Press, 1991.
- [36] Alex Pothén, Horst D. Simon, and Kang-Pu Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.
- [37] Abhiram Ranade. How to emulate shared memory. In *Proceedings of the 28th Annual Symposium on the Foundations of Computer Science*, pages 185–194. IEEE Computer Society, October 1987.
- [38] Abhiram G. Ranade, Sandeep N. Bhatt, and S. Lennart Johnsson. The Fluent abstract machine. In *Advanced Research in VLSI, Proceedings of the fifth MIT VLSI Conference*, pages 71–93. MIT Press, 1988.
- [39] Arnold L. Rosenberg. Preserving proximity in arrays. *SIAM J. Computing*, 4:443–460, 1975.
- [40] Arnold L. Rosenberg and Larry Snyder. Bounds on the costs of data encodings. *Mathematical Systems Theory*, 12:9–39, 1978.
- [41] Horst D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2:135–148, 1991.
- [42] Quentin F. Stout and Bruce Wagar. Intensive hypercube communication I: prearranged communication in link-bound machines. Technical Report

CRL-TR-9-87, Computing Research Lab., Univ. of Michigan, Ann Arbor, MI, 1987.

- [43] Quentin F. Stout and Bruce Wagar. Passing messages in link-bound hypercubes. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.
- [44] Paul N. Swarztrauber. Symmetric FFTs. *Mathematics of Computation*, 47(175):323–346, July 1986.
- [45] Clive Temperton. On the FACR(l) algorithm for the discrete Poisson equation. *J. of Computational Physics*, 34:314–329, 1980.
- [46] Shang-Hua Teng. *Points, Spheres, and Separators: a unified geometric approach to graph partitioning*. PhD thesis, Carnegie-Mellon University, School of Computer Science, 1991.
- [47] Thinking Machines Corp. *CM-200 Technical Summary*, 1991.
- [48] Thinking Machines Corp. *CMSSL for CM Fortran, Version 3.1*, 1993.
- [49] Leslie Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11:350–361, 1982.
- [50] Leslie Valiant and G.J. Brebner. Universal schemes for parallel communication. In *Proc. of the 13th ACM Symposium on the Theory of Computation*, pages 263–277. ACM, 1981.