Experience with the Conjugate Gradient Method for Stress Analysis on a Data Parallel Supercomputer

S. Lennart Johnsson and Kapil Mathur

YALEU/DCS/TR-753
December 1989

Experience with the Conjugate Gradient Method for Stress Analysis on a Data Parallel Supercomputer

S. Lennart Johnsson*and Kapil K. Mathur
Thinking Machines Corporation
Cambridge, MA 02142
Johnsson@think.com, Mathur@think.com

Abstract

The storage requirements and performance consequences of a few different data parallel implementations of the finite element method for domains discretized by three-dimensional brick elements are reviewed. Letting a processor represent a nodal point per unassembled finite element yields a concurrency that may be one to two orders of magnitude higher for common elements than if a processor represents an unassembled nodal point. The former representation also allows for higher order elements with a limited amount of storage per processor. A totally parallel stiffness matrix generation algorithm is presented. The equilibrium equations are solved by a conjugate gradient method with diagonal scaling. The results from several simulations designed to show the dependence of the number of iterations to convergence upon the Poisson ratio, the finite element discretization, and the element order are reported. The domain was discretized by three dimensional Lagrange elements in all cases. The number of iterations to convergence increases with the Poisson ratio. Increasing the number of elements in one special dimension increases the number of iterations to converegnce, linearly. Increasing the element order p in one spatial dimension increases the number of iterations to convergence as p^{α}, where α is 1.4 - 1.5 for the model problems.

1 Introduction

Several applications for which the finite element method is a suitable numerical method are computationally very demanding. Supercomputer performance is desirable. The most critical resource with respect to performance in supercomputer architectures is the bandwidth to storage. The primary storage has for many years been partitioned into so called storage banks in order to provide the necessary bandwidth. Another important characteristic of supercomputer architectures is pipelining of instruction and arithmetic units. As

*Also affiliated with the Department of Computer Science, Yale University

the physical dimensions of devices in MOS and bipolar technologies have been approaching fundamental physical limits, an increasing number of processors has appeared in supercomputer systems. With current technologies, future supercomputers with a performance of a trillion floating-point operations or more must have a large number of processing units and a large number of storage modules. The distributed memory architectures that have appeared during the last several years have many of the characteristics that are expected to be common in supercomputers in the next decade.

Architectures with a large number of processing units require a network with a high bandwidth to support the data motion required by the computation. High communications bandwidth is often the perfomance limiting factor in state-of-the-art technologies. Data placement and data motion are key factors in choosing data structures and algorithms for distributed memory architectures. With a large number of processing units, load balance is another important consideration. The desire to maintain locality of reference and a high degree of concurrency througout the computations has a significant impact upon the choice of algorithms.

In this paper, we first review some of the technological factors affecting supercomputer architectures. Then, we use the Connection Machine® system as a model architecture and discuss some alternative implementations of the finite element method on a massively parallel architecture. We present a totally parallel algorithm for stiffness matrix generation (no communication required). A conjugate gradient method is used for the solution of the equilibrium equations. The communication is local and the load balance is perfect. The rate of convergence is critical for the competitiveness of the conjugate gradient method with direct methods. We present some simulation results showing how the number of iterations to convergence is influenced by the Poisson ratio, the order of the elements for three dimensional Lagrange elements and the discretization of the domain. A diagonal preconditioner was used for all simulations.

2 Trends in Supercomputer Architecture

In the next decade, supercomputers are expected to have a performance of at least one trillion instructions per second, and a primary storage of tens to hundreds of Gbytes [6]. At this rate of computation and memory size, the operation code, the operand addresses, and the operands require 300–400 bits for a single instruction. The storage (including registers, or caches) must deliver 300–400 trillion bits per second, or about 16 million bits per cycle at a 25 MHz clock rate. This clock rate is somewhat conservative for MOS technologies, but it cannot be expected to become higher by more than a small constant factor. The width of the storage needs to be several million bits. Assuming each processor can deliver 50 Mflops/sec [27], 40,000 processors will have a nominal peak capacity of two trillion floating-point instructions per second. A system of this complexity is entirely feasible to build. In half micron technology, 40,000 chips with on-chip floating-point units and memory are projected to have a total of about 64 Gbytes of primary storage. With the required storage bandwidth and with tens of thousands of processing units, a network is the only feasible

alternative for passing data between processors and storage units. Using a technology that is an order of magnitude faster than MOS technologies, such as bipolar GaAs technology (used for the CRAY-3), would still require thousands of processing units for an architecture with a performance of a trillion floating-point operations per second.

In state-of-the art MOS technologies, 10^3 - 10^4 wires fit across a chip. The total data motion capacity of 40,000 chips is 100 - 1,000 TBytes/sec at 25 MHz clock rate without sharing of on-chip channels between different data paths. Assuming current standard packaging technologies of 100 - 300 pins per chip, the data motion capacity at the chip boundary is about 10 TBytes/sec. The data transfer rate on a chip is one to two orders of magnitude higher than the transfer rate at the chip boundary. At the board boundary, assuming connectors with 500 pins, the data motion capacity for a 200 board system is about 0.16 TBytes/sec. The transfer rate at the chip boundary is one to two orders of magnitude higher than the rate at the board boundary. The transfer rate at the board boundary is two to three orders of magnitude below the required rate for a system with a performance in the Tflop/sec range. A sustained performance of this magnitude is not possible with current packaging technologies without locality of reference. This point is clear from Table 2.

A suitable metric for measuring locality of reference is determined either by the topology of the data set, or the communications network. In solving partial differential equation, common distance measures are of the form $(\sum_{i=0}^{d} |x_i - y_i|^p)^{\frac{1}{p}}$, where d is the dimensionality of the problem domain and p the type of *norm*. The 2-norm (Euclidean distance) is often used in the physical domain. The 1-norm measures the distance between two points corresponding to traversals along coordinate axes. This measure is particularly interesting for Boolean cube networks. In such a network of n dimensions with x and y being processor addresses, and x_i and $y_i, 0 \leq i < n$ being the distances (0 or 1) along the coordinate axes, the 1-norm is equal to the *Hamming* distance between the two points. The Hamming distance is equal to the minimum number of communication links a data item must traverse to move from processor x to processor y in a Boolean cube network. The 1-norm is not ideal for all networks. In a completely interconnected network, all points are at unit distance from each other, and the 0-norm is a relevant distance measure.

In Table 2, three frequently used operations are used to illustrate the potential benefits from exploiting locality of reference. The operations are: matrix multiplication, a 7-point symmetric difference stencil applied at each node in a three dimensional grid, and butterfly based computations (FFT, bitonic sort). Applying a symmetric, 7-point difference stencil at every point in a three dimensional grid with k variables per grid point and 2 operations per variable, the number of operations per remote reference is $r = \frac{1}{2d}(\frac{M}{k})^{\frac{1}{d}}$. For $d = 3$ $r = \frac{1}{6}(\frac{M}{k})^{\frac{1}{3}}$. Table 1 gives some values of r for different sizes of the local memories. In Table 1 (and 2), $k = 8$. If the local variables form matrices and the local operations imply matrix multiplications, then the number of arithmetic operations per variable is higher. Several linear algebra operations have a ratio of operations to remote references that can be modeled by the same expression as was given for the difference molecules, i.e. $\frac{1}{\alpha}(\frac{M}{\beta})^{\frac{1}{\gamma}}$ for suitable values of α, β and γ. In the stress analysis case described later, the local state vectors are of length 3, and the local matrices of size 3×24 [23]. For butterfly based

Computation	Registers only	4 Mbit chips	256 4 Mbit chips (board)	256 Boards
Mtx mpy	0.5	104	1600	26000
3-d Relaxation	0.17	4.27	26.7	170.7
FFT	1	18.8	28.8	38.8

Table 1: Number of operations per remote reference of a single variable.

Computation	4 Mbit 1 proc. 1 chip	256 Procs. = Board	256 boards = Machine
Mtx mpy	1	10	160
3-d relaxation	32	480	24600
FFT	3	1140	160000
no locality	300	76800	19660800

Table 2: Number of bits across the chip/board/system boundary per cycle.

algorithms, such as the Fast Fourier Transform (FFT) and sorting, the dependence is of the form $\alpha \log(\frac{M}{\beta})$. For the FFT the ratio is $1.25 log_2(M/2)$ real operations per remote reference using a radix-M algorithm, which is optimum [15]. Exploiting locality reduces the required communication bandwidth by a factor of 8–100 at the chip boundary for these computations, a factor of 80–5000 at the board level, and at least a factor of 125 at the I/O interface. A sustained performance in the Tflops/s range is possible with state-of-the-art technology only if locality is properly exploited. Table 2 gives the number of bits that have to cross the chip, board, and system boundaries during a single cycle, assuming the optimum locality or no locality of reference. It is assumed that each chip has one processing unit, that a board has 256 processing units, and that all variables are in single precision.

3 Data Parallel Supercomputing

The key characteristics of the architectural model presented in the previous section are a large number of processing units with local memories and a network interconnecting these units. Architectures with a large number of processing units are often referred to as *data parallel* architectures, as opposed to *control parallel* architectures. The parallelism in execution in the former architectures is often determined by the size of the data set, or the number of processing units, whichever is smaller. In control parallel architectures, the parallelism is determined by function, rather than by the data set. With thousands, or tens of thousands of processing units, the detailed management of each individual unit, as in most traditional programming languages like for instance Fortran 77, becomes unpractical.

A higher level of abstraction becomes necessary. In this section, we use constructs in the proposed Fotran8X standard to illustrate the value of array constructs. We describe the Connection Machine architecture as an example of a data parallel architecture. The Connection Machine was used for the experiments reported in section five.

3.1 Programming model

Objects in data parallel languages are represented by higher level data types such as arrays in Fortran 8X [25]. In a language with an array syntax, a number of nested loops (often equal to the number of axes in the array) disappear from the code, compared to a language without the array syntax. We illustrate this property by two examples. The first example is the implementation of a 7-point stencil in three dimensions. The second example is taken from a finite element code for stress analysis.

In the example below, which defines the computation of a 7-point stencil at every point in a three dimensional grid, the operation CSHIFT defines a circular shift. The first argument is the variable to which the shift is applied, the second defines the axis along which the shift takes place, and the third argument defines the length and direction of the shift. Since there is no conditional statement in the code below, it implements periodic boundary conditions. (Note that there are no explicit loops for the array axes.)

```
      subroutine psolve(phi, omega, inside, n, iter)
      real phi(n, n, n), omega(n, n, n), factor
      factor = 1.0/6.0
      do 100 i=1,iter,1
        phi = factor * (
1         CSHIFT(phi, dim=1, shift=-1) +
2         CSHIFT(phi, dim=2, shift=-1) +
3         CSHIFT(phi, dim=3, shift=-1) +
4         CSHIFT(phi, dim=1, shift=+1) +
5         CSHIFT(phi, dim=2, shift=+1) +
6         CSHIFT(phi, dim=3, shift=+1) ) +
7         omega
  100 continue
      return
      end
```

In the finite element example below, the elements are brick elements of first order. There is one nodal point in each corner of an element. The state is represented by three displacements, $x = (u, v, w)$. The local interaction matrix, the elemental stiffness matrix, is a 3×24 matrix, with one row for each of the three components of the local displacement vector. The code segment also contains one compiler directive, SERIAL, which affects the data layout. The meaning will be explained later. The code fragment is from the iterative solver which requires the computation of a matrix vector product. In the particular finite

element code from which the code segment is selected, the elemental stiffness matrices are not assembled into a global stiffness matrix. Instead, a matrix vector product is performed for each element, and a total product vector assembled.

```
CMF$LAYOUT K(:SERIAL, :SERIAL, , , ), R(:SERIAL, , , ), X(:SERIAL, , , )
REAL K(3,24, 32, 32, 32), R(3,32,32,32), U(3,32,32,32), V(3,32,32,32), W(3,32,32,32), X(24,32,32,32)
CALL ALL-TO-ALL-ELEMENT-BROADCAST(U,V,W,X)
R = 0.0
DO I=1,24
  DO J=1,3
    R(J,:,:,:)=R(J,:,:,:)+K(J,I, :, :, :) * X(I, :, :, :)
  END DO J
END DO I
(WHERE (.NOT. I-RIGHT-BOUNDARY)) R=R + EOSHIFT(R, 1, 1)
(WHERE (.NOT. I-LEFT-BOUNDARY)) R= EOSHIFT(R, 1, -1)
(WHERE (.NOT. J-RIGHT-BOUNDARY)) R=R + EOSHIFT(R, 2, 1)
(WHERE (.NOT. J-LEFT-BOUNDARY)) R= EOSHIFT(R, 2, -1)
(WHERE (.NOT. K-RIGHT-BOUNDARY)) R=R + EOSHIFT(R, 3, 1)
(WHERE (.NOT. K-LEFT-BOUNDARY)) R= EOSHIFT(R, 3, -1)
```

In the above code segment, I-RIGHT-BOUNDARY, I-LEFT-BOUNDARY, etc. are boolean arrays which define the right–hand and left–hand boundaries of the finite element mesh.

3.2 The Connection Machine Architecture

The Connection Machine [11] is a data parallel architecture. It has a total primary storage of 512 Mbytes using 256 kbit memory chips, and 2 Gbytes with 1 Mbit memory chips. The data transfer rate to storage is approximately 45 Gbytes/s at a clock rate of 7 MHz. The primary storage has 64k ports and a simple 1-bit processor for each port. The storage per processor is 8 kbytes for a total storage of 512 Mbytes and 64k bytes with 1 Mbit memory chips. The Connection Machine model CM–2 can be equipped with hardware for floating-point arithmetic. With the floating-point option, 32 Connection Machine processors share a floating-point unit, which is an industry standard, single chip floating-point multiplier and adder with a few registers. The peak performance available from the standard instruction set and the higher level languages is in the range 1.5 Gflops/s - 2.2 Gflops/s. The higher level languages do not at the present time make efficient use of the registers in the floating-point unit for operations that vectorize. With optimum use of the registers, a performance that is one order of magnitude higher is possible. For instance, for large local matrices, a peak performance in excess of 25 Gflops/s has been measured [20].

The Connection Machine needs a host computer. Currently, three families of host architectures are supported: the VAX family with the BI-bus, SUN 4, and the Symbolics 3600 series. The Connection Machine memory is mapped into the address space of the host. The program code resides in the storage of the host. It fetches the instructions, does

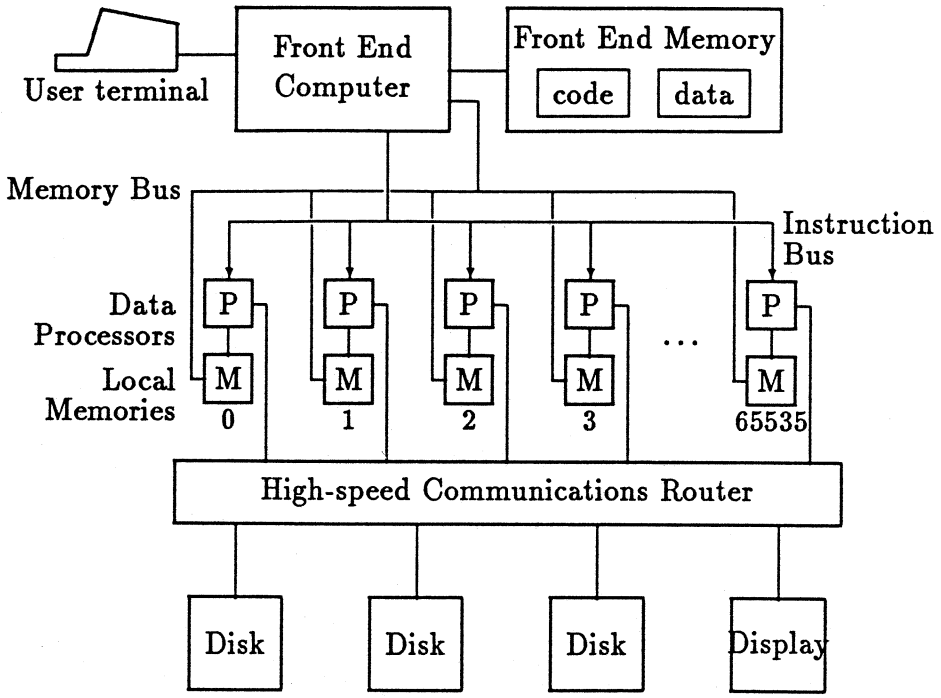

Figure 1: The Connection Machine System

the complete decoding of scalar instructions, and executes them. Instructions to be applied to variables in the Connection Machine are sent to a microcontroller, which decodes and executes instructions for the Connection Machine. Variables defined by array constructs are allocated to the Connection Machine, unless allocation on the front-end is requested. The architecture is depicted in Figure 1. The Connection Machine can also be equipped with a secondary storage system known as the data vault. There exist 8 I/O channels, each with a block transfer rate of up to approximately 30 Mbytes/s. The size of the secondary storage system is in the range 5 Gbytes to 640 Gbytes. The Connection Machine can also be equipped with a frame buffer for fast high resolution graphics. An update rate of about 15 frames per second can be achieved.

The Connection Machine processors are organized with 16 processors to a chip, and the chips interconnected as a 12-dimensional Boolean cube. The communication is bit-serial and pipelined. Concurrent communication on all ports is possible. Through the bit-serial pipelined operation of the communication system, remote processor references require no more time than nearest neighbor references provided there is no contention for communication channels. For communication in arbitrary patterns, the Connection Machine is equipped with a router which selects one of the shortest paths between source and destination, unless all of these paths are occupied. The router has several options for resolving contention for communication channels.

Array elements in the Connection Machine programming languages are often referred to as *virtual processors* [11, 30]. In general, several virtual processors (array elements)

are mapped to the storage of each physical processor. The number of virtual processors per physical processor is called the *virtual processor ratio* [30]. The storage of a physical processor is divided between as many virtual processors as is given by the virtual processor ratio. That many virtual processors time-share a physical processor.

3.3 Performance issues

In highly concurrent network architectures, the nominal processing capability is determined by the processing speed of a single processor and the number of processors. The real processing capability is determined by how well the individual processing units can be utilized, load balance, and how well the network supports the data motion required by the computation. The capacity available for the data motion is determined by technological constraints, and the requirements determined by data placement, computational algorithms, and routing algorithms. Of the various technological constraints that determine the performance characteristics of an architecture, the ones related to data motion are often the most unforgiving with respect to performance. The characteristics with respect to load balance and communication depend both on the problem and the numerical technique used to solve it.

3.3.1 Data motion and load balance.

Mathematical models such as (partial) differential equations are derived from laws of physics applied locally. Discrete approximations of continuous operators, such as difference stencils used to approximate (partial) derivatives in finite difference techniques are also local approximations. Finite elements provide a different local approximation. The difference stencils in finite difference techniques and the elements in finite element techniques completely define the spatial data interactions in one step of an *explicit method* for the solution of the discretized equations. The data interaction is local in the physical domain. The classical iterative methods for the solution of linear systems of equations only require local data interaction in the index space used for the solution variables. The conjugate gradient method requires a global reduction operation for the computation of scaling factors, and a global copy, or broadcasting, operation for the distribution of these factors in addition to the same local communication as required by Jacobi's method. Though each step in the iterative methods only involves local communication in the physical domain, most problems require global communication to attain a correct solution. Elliptic problems are of this type [7]. In explicit methods for solving partial differential equations, and iterative methods for systems of equations such as Jacobi and the conjugate gradient method, the entire data set is typically involved in each step, or iteration. The load balance with an even distribution of data elements to processors is ideal. If the computational domain can be mapped into processor network preserving locality, then good performance is possible for explicit techniques for solving partial differential equations, and iterative methods for linear systems of equations.

Direct methods requires non-local communication in the problem domain during some computation steps (potentially all steps). Factoring matrices by Gaussian elimination, or

Householder transformations, using rank-1 updates implies distribution of the pivot row (selected in Gaussian elimination, computed in Householder transformations) to the rows of the remaining submatrix with non-zero entries in the pivot column, and a dsitribution of the pivot column to the remaining columns with a non-zero entry in the pivot row. Depending on the topology of the graph that the (sparse) matrix represents, and the mapping of the graph to the processor network, the elimination process may only require local communication, even for networks of bounded degree. In the factorization process a reduction of the active data set takes place. This property leads to poor load balance. Similarly, the sequential dependence in the forward and backsubstitution process may lead to poor load balance. In the case of Gaussian elimination the average processor utilization may be reduced by a factor of 2 - 3 [17]. Hence, iterative methods often yield a better load balance than direct methods, though methods such as parallel cyclic reduction [14], or balanced cyclic reduction [18], may achieve good load balance. Likewise, even though the communication may be non-local in the computational domain, it may be local in the processor network.

3.4 Data Allocation

The goal of a good data allocation scheme is to map data into the local memories of the processors such that the need for communication is minimized. Any interconnection network into which the discretized physical domain can be embedded preserving locality has the potential for communication efficient implementation of explicit methods for partial differential equations, or iterative methods for systems of equations. High degree networks have the potential to offer local communication even when the communication in the physical domain is non-local. Hence, divide-and-conquer methods for solving linear systems of equations, such as odd-even cyclic reduction [3], parallel cyclic reduction [14], balanced cyclic reduction [18], nested dissection [8], and multi-grid methods [2], may still only require local communication in the processor network. For instance, any regular lattice can be embedded in a lattice of higher dimensionality preserving locality, but the converse is not true [29].

3.4.1 Data aggregation

For any data set with more data elements than processors several data elements need to be assigned to each processor. In *consecutive* [17] allocation successive data points along an axis are allocated to the same processor. In cyclic allocation [17] successive data points are allocated to adjacent processors. A significant difference in performance may result, since the communication between a processor and its memory is often considerably faster than communication between processors. For computations in which the interaction between data elements is equally frequent in all directions, the total amount of communication is minimized if the data elements assigned to a processor chip forms a single subdomain with an aspect ratio as close to one as possible [19]. The consecutive allocation scheme reduces the amount of inter-processor communication for explicit methods, but the cyclic allocation

scheme yields better load balance for direct methods [17, 19].

3.4.2 Encoding of array axes

In the common binary, encoding successive integers may differ in an arbitrary number of bits. For instance, 63 and 64 differs in 6 bits, and hence are at a *Hamming* distance of 6 in the Boolean cube. A *Gray* code by definition has the property that successive integers differ in precisely one bit. The most frequently used Gray code for the embedding of arrays in Boolean cubes is a *binary-reflected* Gray code [17, 22, 28]. This Gray code is periodic. The code preserves adjacency for any loop (periodic one-dimensional lattice) of even length, and for loops of odd length one edge in the loop is mapped into a path of length two [17]. For the embedding of multi-dimensional arrays, each axis may be encoded by the *binary-reflected* Gray code. The embedding of an $N_1 \times N_2 \times \ldots \times N_d$ array requires $\sum_{i=1}^{d} \lceil \log_2 N_i \rceil$ bits. The *expansion*, i.e., the ratio between the consumed address space and the actual array size, is $2^{\sum_{i=1}^{d} \lceil \log_2 N_i \rceil} / \Pi_{i=1}^{d} N_i$, which may be as high as $\sim 2^d$ [10, 12]. The expansion can be reduced by allowing some successive array indices to be encoded at a Hamming distance of two. The *dilation* is the maximum Hamming distance between any pair of adjacent array indices. Every two-dimensional array can be embedded with minimum expansion and dilation 2 [4]. Minimum expansion dilation 2 embeddings for a large class of two-dimensional arrays are given in [12], which also provides a technique for reducing the expansion of higher dimensional arrays. Minimal expansion dilation 7 embeddings are possible for all three dimensional arrays [5]. Embeddings with dilation 2 for many three dimensional arrays are given in [13].

3.4.3 The Connection Machine

The address field of the Connection Machine is divided into three parts: (off-chip|on-chip|memory). The off-chip field consists of 12 bits that encode the Connection Machine processor chips, the on-chip field encodes the 16 processors on each Connection Machine processor chip, and the lower order bits encode the memory addresses local to a processor. The lowest order off-chip bit encodes pairs of processor chips sharing a floating-point unit. The default data allocation scheme on the Connection Machine first determines how many data elements need to be stored in each processor for an equal number of elements per processor, then stores that many successive elements in each processor, *consecutive storage* [17].

Current implementations of the Connection Machine languages encode each axis of a multi-dimensional array separately. Each axis is extended to a length that is equal to some power of two. For an axis length P, $\lceil log_2 P \rceil$ address bits are assigned to the encoding of the elements along that axis. The consecutive allocation scheme is used for each axis. The encoding of the axes in the total address space attempts to configure each part of the address space (off-chip, on-chip, and memory) to conform with the array. To the extent possible, all axes have a segment of each address field, and the ratio of the lengths of segments for different axes is the same as that of the length of the axes.

The default allocation of axes to off-chip, on-chip, and memory bits may not always be the preferred allocation. The different Connection Machine languages provide different means for user controlled data allocation. In CM-Fortran compiler directives allow a user to specify an axis as SERIAL, which implies that the axis is allocated to a single processor. In PARIS (PARallel Instruction Set), the Connection Machine native language, a user has full control over what dimensions of the address space an axis occupies. But, only consecutive allocation of data to processors is supported.

If an array has fewer elements than the number of real processors in the configuration, then the array is extended such that there is one element per real processor. In CM-Fortran an axis is added to the array with a length equal to the number of instances of the specified array that matches the number of real processors.

The lattice emulation by a binary-reflected Gray code embedding is part of the standard programming environment on the Connection Machine system. In CM-Fortran, array axes are by default encoded in a binary-reflected Gray code for the off-chip segment of the address field. In the other Connection Machine languages, the Gray code encoding is invoked by configuring the Connection Machine as a lattice of the appropriate number of dimensions. The benefit of the lattice emulation feature is twofold: the virtual processors are assigned to physical processors such that the communication requirements are minimized, and lattice organized computations are often easier to express using programming constructs corresponding directly to the operations in the problem domain.

4 The Finite Element Method

4.1 Mathematical model

In stress analysis, the finite element discretization [26, 31] is formulated from a variational principle representing the statement of virtual work

$$0 = -\int_V \delta \mathrm{Tr}(\epsilon\sigma) dV + \int_{S_\sigma} \delta \mathbf{u}^T \mathbf{T} dS, \tag{1}$$

where $\delta\mathbf{u}$ is the virtual displacement field compatible with the virtual strain $\delta\epsilon$ and σ is the Cauchy stress in equilibrium with the applied traction field $\mathbf{T}$. After discretizing the domain into finite elements, and introducing piecewise interpolation functions, which are non–zero only in the domain of one finite element, approximations for the displacement field and the corresponding strain field are obtained as

$$\{u\} = [N]\{U\}, \tag{2}$$

and

$$\{\epsilon\} = [N']\{U\}. \tag{3}$$

In the above equations, the matrix $[N]$ comprises of a set of interpolation functions and $[N']$ is a matrix containing the spatial derivatives of these interpolation functions. The final

system of equations that results from the above approximations is of the form

$$[K]\{U\} = \{F\}, \tag{4}$$

where the global stiffness matrix $[K]$ is really a collection of elemental stiffness matrices

$$[K] = \sum_{\mathrm{i}} \left[K^{(\mathrm{i})}\right]. \tag{5}$$

The evaluation of the elemental stiffness matrices involves computation of interpolation functions which are local over the domain of a single finite element. The inherent parallelism in the concurrent generation of the elemental matrices is clear. But, there is also a significant parallelism in the generation of individual elemental stiffness matrices.

4.2 Data structures

In a data parallel implementation an operation on an array implies that all processors to which the array is allocated perform the operation (unless the statement is a conditional statement). For example, if a statement of the form

$$\mathbf{x} = 1.0, \tag{6}$$

is encountered, then the computing system sets the value of $\mathbf{x}$ to 1.0 in all processors. In the context of the Connection Machine this property is true regardless of the configuration of the address space, and its encoding. Also, conditional operations apply to the selected processors regardless of the configuration of the address space and its encoding.

For the finite element method, it is natural to have a processor represent either

- a finite element from the finite element mesh, or
- an unassembled nodal point of the finite element mesh.

For a mesh composed of identical finite elements, the latter choice has several advantages [23]. The two main advantages are:

1. With n nodal points per element, the degree of concurrency is a factor of n greater than the degree of concurrency obtained when finite elements are chosen as logic units of data.
2. the local storage requirement for the elemental stiffness matrix is a factor of $\frac{1}{6}(3n+1)$ less than the storage requirements for the first choice.

The influence of the amount of local storage and total storage available on the number of three dimensional Lagrange elements that can be accommodated on a Connection Machine System with 512 Mbytes of total storage is shown in Table (3) [23, 24]. The choice of the

	order	nodes per elem.	virtual processor ratio	maximum deg. of freedom (approx.)	maximum number of elements (approx.)
Processor per unassembled node	$1 \times 1 \times 1$	8	8	780,000	260,000
	$2 \times 2 \times 2$	27	4	295,000	32,000
	$3 \times 3 \times 3$	64	1	148,000	9,000
	$4 \times 4 \times 4$	125	1	62,000	4,000
Processor per element, unsym.	$1 \times 1 \times 1$	8	2	390,000	130,000
Processor per element, sym.	$1 \times 1 \times 1$	8	4	780,000	260,000

Table 3: The maximum number of degrees of freedom for three dimensional Lagrange elements that can be accommodated in 512 Mbytes of storage partitioned into 8 Kbytes per physical processor.

Virtual processor	Type of communication	Element transfers	Arithmetic
Processor per unassem. node	Intra–element (all to all)	$2(n-1)u$	$(2nu-1)u$
	Inter–element (assembly)	$6u$	$3u$
Processor per element	Intra–element	-	$(2nu-1)nu$
	Inter–element (assembly)	$6n^{\frac{2}{3}}u$	$3n^{\frac{2}{3}}u$

Table 4: Data element transfers and arithmetic operations per logical unit for Lagrange elements in three dimensions.

logical unit is clearly dependent on the available local and total storage. Analytic expressions for the storage requirements of two and three dimensional Lagrange and Serendipity elements can be found in [23]. Table (4) [23, 24] summarizes the communication and arithmetic requirements for the two choices of the logical unit of data. The ratio of the number of data element transfers to arithmetic operations per logical unit of data is approximately the same for the two choices.

The data allocation with a virtual processor representing an unssembled nodal point of a Lagrange element is illustrated in Figure 2 for Lagrange elements in two dimensions. The bilinear elements are labeled A, B, C, and D. The mesh has nine nodes labeled one through nine. Nodal points that are shared between elements are replicated on separate processors. Only the information about the geometry (the global coordinates) needs to be replicated on the processors representing the same nodal point. The 3×3 layout of nodes is mapped on to a 4×4 lattice of processors. Nodal point labeled five is shared by all four elements and is consequently placed on four separate processors. In general, every internal grid line in two dimensions, and every internal surface in three dimensions, is duplicated.

In the implementation from which experimental results are reported below three dimensional Lagrange elements are used, and a virtual processor represents an unassembled nodal point. A detailed evaluation of benefits and drawbacks can be found in [23]. Each processor

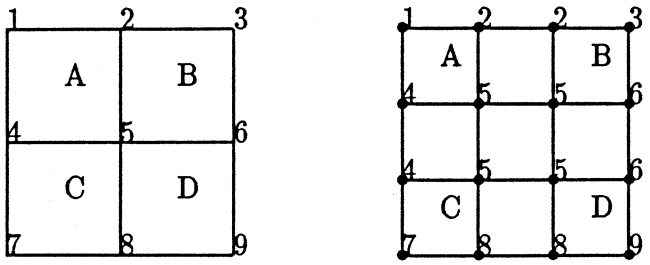

Figure 2: Mapping a physical domain composed of rectangular elements to processors. The processors are represented by dots.

stores the rows of the elemental stiffness matrix corresponding to the unassembled nodal point represented by the processor, i.e., a $u \times nu$ matrix per processor, where u is the number of degrees of freedom per node and n is the number of nodes per element.

For regular two or three dimensional finite element meshes the lattice emulation feature of the programming systems on the CM–2 is used advantageously. It simplifies the programming, and allows for efficient communication.

4.3 Algorithms

4.3.1 Stiffness matrix generation

When one virtual processor of the Connection Machine system represents an unassembled nodal point, the generation of the elemental stiffness matrix for each element is shared by n processors. The generation of the entries of the elemental stiffness matrices requires numerical integration, which is performed by Gauss quadrature. This quadrature can be carried out without communication by performing it sequentially on each virtual processor [23]:

```
for all quadrature points, k
  evaluate jacobian and shape function derivatives
    at the quadrature point, k.
  add contribution of the quadrature point, k,
    to the rows of the elemental matrix stored
    on the virtual processor.
```

4.3.2 Solution of the equilibrium equations

The equilibrium equations are solved by a conjugate gradient method with diagonal scaling. The main computational and data communication effort is in the sparse matrix–vector product of the form

$$\{r\} = \{b\} - [A]\{x\}, \tag{7}$$

where the coefficient matrix $[A]$ is not explicitly assembled, but is stored as

$$[A] = \sum_{i}^{n} \left[A^{(i)}\right]. \tag{8}$$

The sparse matrix–vector product involves:

1. Accumulation of the local displacements from the processors representing the unassembled nodes. All processors forming the element require the local displacements from every other processor in this subset. This communication is a segmented "all-to-all" broadcast [21], that can be implemented efficiently by nearest neighbor communication, if the processors are configured as a lattice. After the segmented "all-to-all" broadcast every processor stores the local displacements for every node on the element in a vector of length nu.

2. A local matrix–vector product $[(u \times nu) \times (nu \times 1)]$ is then performed by every processor. After this multiplication, every processor contains the unassembled contribution of the nodal point to the product vector $(u \times 1)$.

3. Finally, the product vector is assembled by performing nearest neighbor communication among processors representing the same nodal point.

For the example two dimensional mesh shown in Figure 2, the three steps above for stress analysis are:

1. For all elements (A–D) accumulation of the eight displacement components associated with each element.

2. Multiplication of the two rows of the unassembled stiffness with the accumulated displacement vector.

3. Assembly over all processors representing replicated nodal points (nodes labeled 2, 4, 5, 6, and 8).

5 Computational experiments

The number of iterations to convergence for the conjugate gradient method depends upon the condition number of the global stiffness matrix, or the stiffness matrix modified by the

N_0, number of elements	Conjugate gradient iterations				
	$p_0 = 1$	$p_0 = 2$	$p_0 = 3$	$p_0 = 4$	$p_0 = 5$
100	100	215	436	652	1083
200	200	442	869	1331	2166
300	300	670	1301	2003	3251
400	400	899	1773	2673	4319
500	500	1129	2167	3345	5400

Table 5: The number of conjugate gradient iterations for a $10 \times 1 \times 1$ domain discretized by $N_0 \times 1 \times 1$ elements of order $p_0 \times 1 \times 1$. Poisson ratio $\nu = 0$. Convergence criteria: a normalized global residual of 5.0×10^{-8}.

preconditioning matrix for a preconditioned conjugate gradient method. The number of iterations to convergence for the standard conjugate gradient method is proportional to the logarithm of the error used as a convergence criteria, and the square root of the condition number [1]. The relationship between the condition number and the element order for a few types of planar elements are derived in [1].

In this section we give convergence characteristics for the conjugate gradient method with a diagonal preconditioner for a number of experiments designed to improve the understanding of the relationship between the condition number, the finite element discretization, the order of the elements, and the boundary conditions for rectangular domains discretized by three-dimensional Lagrange elements [31]. Two domains are considered: a beam of dimensions $10 \times 1 \times 1$ discretized by $N_0 \times 1 \times 1$ elements, and a $10 \times 1 \times 10$ plate discretized either by $N_0 \times 1 \times 16$ elements or by $16 \times 1 \times N_2$ elements. All elements for a given discretization are of the same order. The order of the elements are specified as $p_0 \times p_1 \times p_2$.

The results from different discretizations of the beam are reported first. Only one loading case was investigated. Next, the results from varying the discretization of the plate are reported. A pulling and a bending load was applied to the plate. For both domains a Poisson ratio of 0, and a ratio of 0.3 were investigated.

5.1 A beam discretized by a linear array of brick elements

The $10 \times 1 \times 1$ domain is discretized by $N_0 \times 1 \times 1$ elements of order $p_0 \times 1 \times 1$. The number of elements along the first axis is N_0, and the number of elements along each of the second and third axis is one. All elements are of Lagrange type, and of the same order. The order of the elements along the first axis is p_0, and the order along each of the second and third axes is one. The boundary at $x_0 = 0$ is fixed, and a pulling force applied at $x_0 = 10$, where x_0 is the coordiante value in dimension 0.

Simulation results for a Poisson ratio $\nu = 0$ are given in Table 5 for various discretizations and element order. The number of iterations is equal to N_0 for $p_0 = 1$. The number of degrees of freedom is $12(p_0 N_0 + 1)$. The number of iterations cannot be less than N_0 with

Node	t=30	t=60	t=90	t=120
$N_0 - t + 0$	0.931×10^{-04}	0.931×10^{-04}	0.931×10^{-04}	0
$N_0 - t + 1$	0.208×10^{-18}	0.318×10^{-18}	0.477×10^{-18}	0.477×10^{-18}
$N_0 - t + 2$	0.533×10^{-18}	0.105×10^{-17}	0.122×10^{-17}	0.130×10^{-17}
$N_0 - t + 3$	0.960×10^{-19}	0.199×10^{-18}	0.667×10^{-18}	0.934×10^{-18}
$N_0 - t + 4$	0.647×10^{-18}	0.694×10^{-18}	0.605×10^{-18}	0.383×10^{-18}
$N_0 - t + 5$	0.393×10^{-19}	0.429×10^{-18}	0.431×10^{-18}	0.763×10^{-18}
$N_0 - t + 6$	0.367×10^{-18}	0.218×10^{-18}	0.431×10^{-18}	0.470×10^{-18}
$N_0 - t + 7$	0.206×10^{-18}	0.556×10^{-18}	0.594×10^{-18}	0.928×10^{-18}
$N_0 - t + 8$	0.627×10^{-18}	0.213×10^{-18}	0.539×10^{-18}	0.493×10^{-18}
$N_0 - t + 9$	0.369×10^{-18}	0.445×10^{-18}	0.666×10^{-18}	0.782×10^{-18}

Table 6: Evolution of local residuals for a force applied at time $t = 0$ at location $x_0 = 10$ for the $10 \times 1 \times 1$ domain discretized by $N_0 \times 1 \times 1$ elements of order $1 \times 1 \times 1$. Poisson ratio $\nu = 0$.

Node	t=30	t=60	t=90	t=120
$N_0 - t + 0$	0.537×10^{-04}	0.365×10^{-04}	0.248×10^{-04}	0
$N_0 - t + 1$	0.382×10^{-04}	0.489×10^{-04}	0.507×10^{-04}	0.466×10^{-04}
$N_0 - t + 2$	0.146×10^{-04}	0.118×10^{-04}	0.173×10^{-04}	0.294×10^{-04}
$N_0 - t + 3$	0.807×10^{-05}	0.168×10^{-04}	0.228×10^{-04}	0.210×10^{-04}
$N_0 - t + 4$	0.134×10^{-04}	0.174×10^{-04}	0.138×10^{-04}	0.985×10^{-05}
$N_0 - t + 5$	0.158×10^{-04}	0.145×10^{-04}	0.102×10^{-04}	0.145×10^{-04}
$N_0 - t + 6$	0.159×10^{-04}	0.126×10^{-04}	0.108×10^{-04}	0.117×10^{-05}
$N_0 - t + 7$	0.150×10^{-04}	0.109×10^{-04}	0.832×10^{-05}	0.743×10^{-05}
$N_0 - t + 8$	0.134×10^{-04}	0.758×10^{-05}	0.574×10^{-05}	0.892×10^{-05}
$N_0 - t + 9$	0.109×10^{-04}	0.268×10^{-05}	0.721×10^{-05}	0.104×10^{-04}

Table 7: Evolution of local residuals for a force applied at time $t = 0$ at location $x_0 = 10$ for the $10 \times 1 \times 1$ domain disretized by $N_0 \times 1 \times 1$ elements of order $1 \times 1 \times 1$. Poisson ratio $\nu = 0.3$.

p_0, order of interpolation	Total degrees of freedom	Conjugate gradient iterations
1	1452	120
2	2892	260
3	4332	522
4	5772	787
5	7212	1309

Table 8: The number of iterations to convergence for a $120 \times 1 \times 1$ mesh of elements of order $p_0 \times 1 \times 1$. Poisson ratio $\nu = 0$. A Pulling load is applied at $x_0 = 10$, whereas $x_0 = 0$ is fixed. Convergence tolerance: a normalized global residual of $\leq 5.0 \times 10^{-8}$.

the loading applied at one end of the bar discretized with N_0 elements along the axis along which the force is applied. The iterative method requires N_0 steps for the force to propagate through the structure. The convergence is very rapid once the force has propagated to a node. The convergence behavior is apparent from considering the local residuals, Table 6. The influence of the value of the Poisson ratio on the convergence behavior is quite significant. The values of the local residuals for a Poisson ratio of $\nu = 0.3$ is shown in Table 7. The decay of the local residuals is quite slow. Essentially the force propagates through the structure while a very moderate decay in the error takes place. The error then decreases slowly and uniformly thoughout the beam, as shown in Figure 3. The number of iterations to convergence for a convergence criteria of 5.0×10^{-8} increases by about 50%.

The global stiffness matrix is block tridiagonal with the nodes shared between two elements numbered consecutively, and nodes ordered from one end to the other along axis zero. For first order elements the blocks are 4×4 block matrices, with each such block being a 3×3 matrix. The number of blocks is N_0, and the matrix size is $12N_0 \times 12N_0$. A block Gaussian elimination solver (12×12 blocks) would need N_0 (block) forward and backsubstitution steps. With the force applied at one end only backsubstitution is required. A parallel direct solver, like a nested dissection solver [8, 16], would compute the solution in $\log N_0$ (block) steps.

For higher order elements the block tridiagonal matrix changes shape such that there are $4p_0 \times 4p_0$ diagonal blocks of 3×3 matrices with 4×4 3×3 blocks coupling between the diagonal blocks. The number of diagonal blocks is N_0. As p_0 increases the dynamic behavior becomes more important. Table 8 gives the results from some simulations on a beam discretized by $120 \times 1 \times 1$ elements. Figure 4 shows the same data graphically.

The number of iterations to convergence depends on the interpolation order p_0 according to the formulas

$$N_{cg} = N_0 p_0^{1.46} \qquad \text{for } \nu = 0.$$

$$N_{cg} = 1.5 N_0 p_0^{1.46} \qquad \text{for } \nu = 0.3.$$

The exponent is independent of ν. A set of simulations was also carried out in which both N_0 and p_0 were varied such that $N_0 p_0 = const.$ The dependence on N_0 and p_0 was the

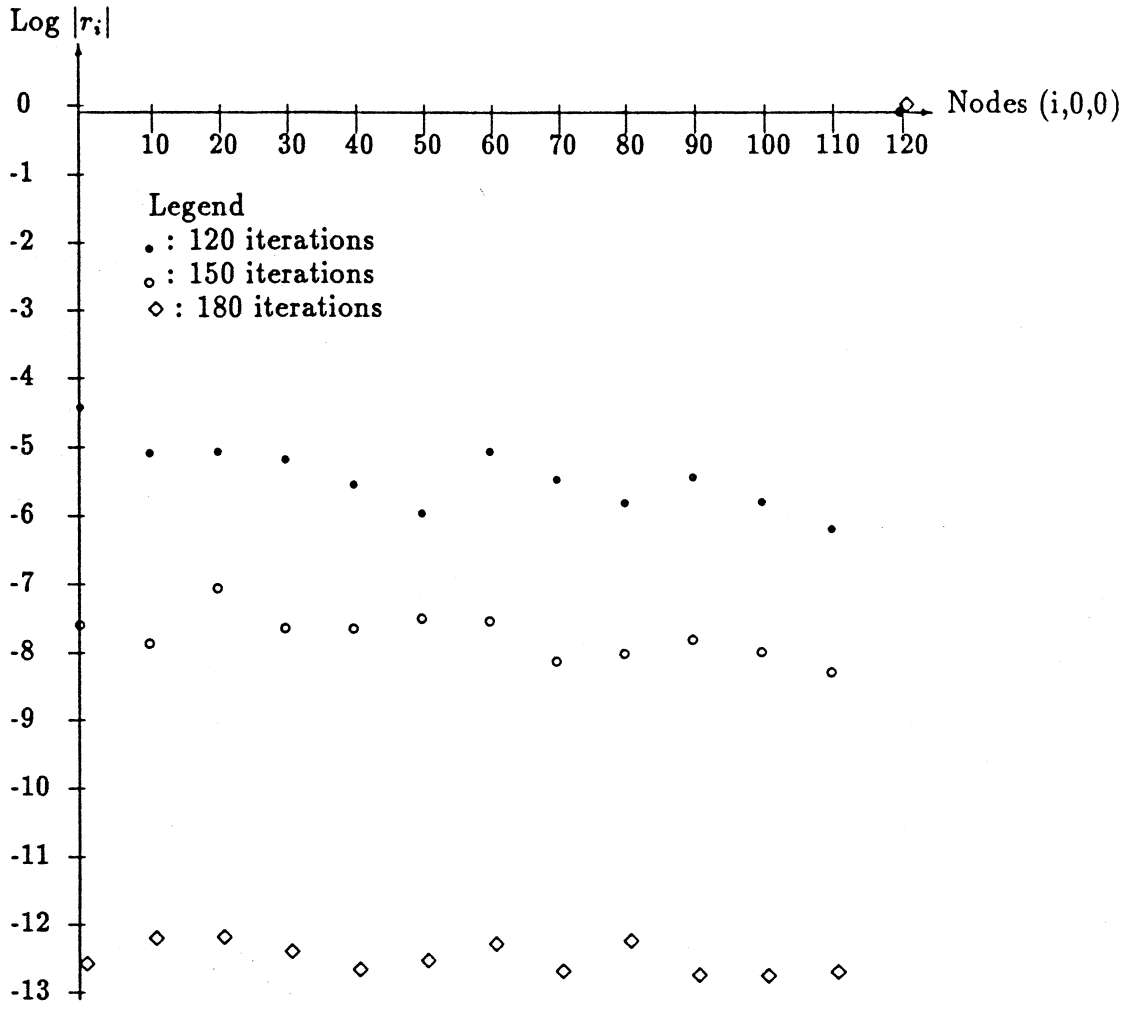

Figure 3: Local residuals for pulling a beam discretized by $120 \times 1 \times 1$ elements of order $1 \times 1 \times 1$. Poisson ratio $\nu = 0.3$

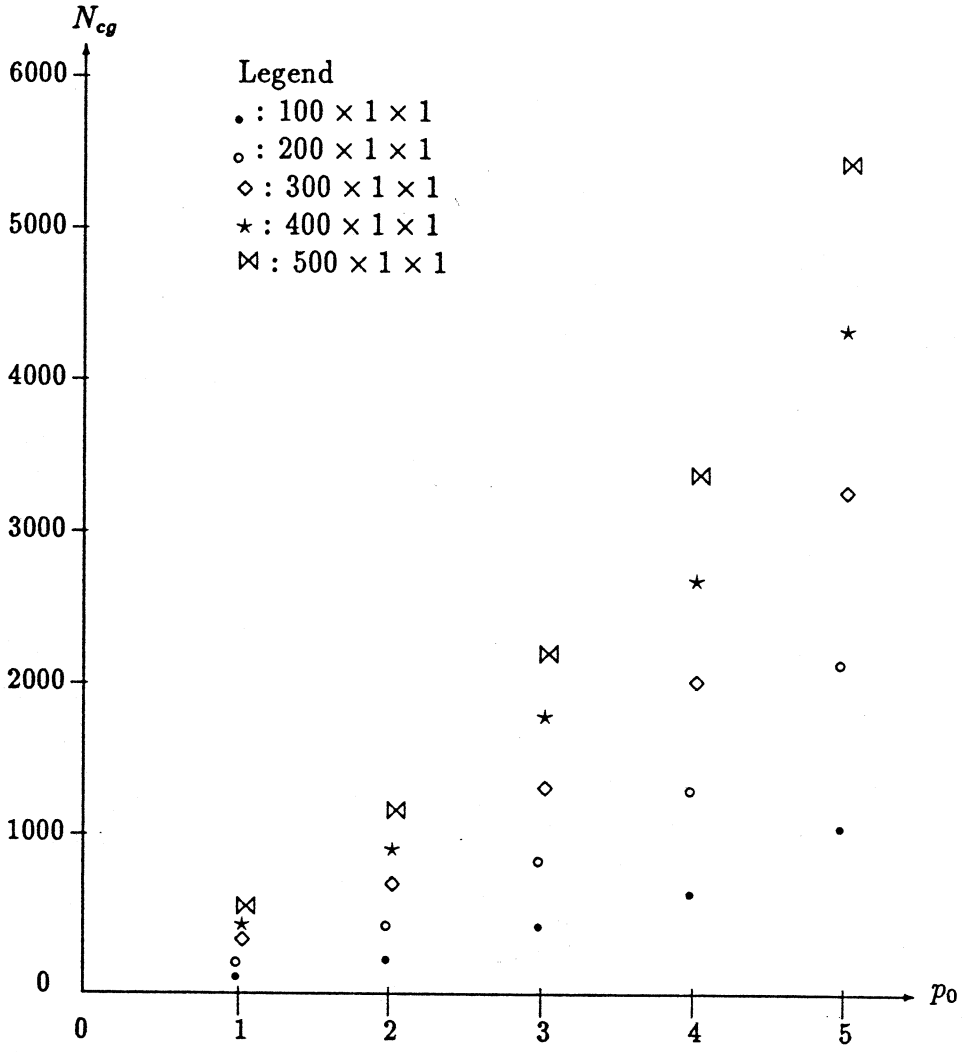

Figure 4: The number of iterations to convergence as a function of the order of $p_0 \times 1 \times 1$ elements for a domain discretized by $N_0 \times 1 \times 1$ elements. Poisson ratio $\nu = 0$

N_0	p_0	Number of conjugate gradient iterations			
		1.0×10^{-3}	1.0×10^{-4}	1.0×10^{-5}	5.0×10^{-8}
120	1	120	120	120	120
60	2	122	122	123	125
40	3	155	158	161	175
30	4	167	169	171	176
24	5	227	235	239	258

Table 9: Influence of the interpolation order on the convergence behavior of the conjugate gradient method for $N_0 \times 1 \times 1$ elements of order $p_0 \times 1 \times 1$, $N_0 p_0 = const.$ Poisson ratio $\nu = 0$. Convergence criteria: a normalized global residual of 5.0×10^{-8}.

same, Table 9. Changing N_0 with p_0 such that $N_0 p_0$ is constant yields a matrix of constant size, but it becomes increasingly dense as p_0 increases and N_0 decreases.

5.2 Multiple elements along two axes.

The $10 \times 1 \times 10$ domain is discretized either by $N_0 \times 1 \times 16$ elements, or by $16 \times 1 \times N_2$ elements. The order of each Lagrange element for the first type of discretization is $p_0 \times 1 \times 1$, and for the second type of discretization it is $1 \times 1 \times p_2$. With the crossection consisting of several elements the propagation of the force through the structure accounts only for an insignificant number of iterations of the total. The dynamic properties in the transverse directions become significant. We first consider pulling of the plate fixed at $x_0 = 0$ by a distributed force applied at $x_0 = 10$, then consider the case of a bending force applied at the same surface.

5.2.1 Pulling

The number of iterations to convergence for discretizations by $N_0 \times 1 \times 16$ elements of order $1 \times 1 \times 1$ is shown in Table 10 The number of iterations for discretizations $16 \times 1 \times N_2$ is shown in Table 11. Figure 5 shows the convergence behavior for 128 elements along axis zero or two, and a Poisson ratio of 0. The convergence for the same discretizations, but a Poisson ratio of 0.3 is shown in Figure 6.

The required number of iterations to convergence depends linearly upon the mesh resolution for the axis with the smallest grid point spacing. The side of the brick elements is the shortest along this axis. The linear dependence holds regardless of which axis has the smallest grid point spacing, and for both Poisson ratios. The linear dependence upon the number of elements N_i is expected for a conjugate gradient method with diagonal preconditioner [1, 9]. The constant of proportionality depends upon the accuracy, the axis of smallest grid point spacing, and the Poisson ratio, as seen from Table 12. The constants of proportionality in this table are computed for the range $16 \leq N_i \leq 128$.

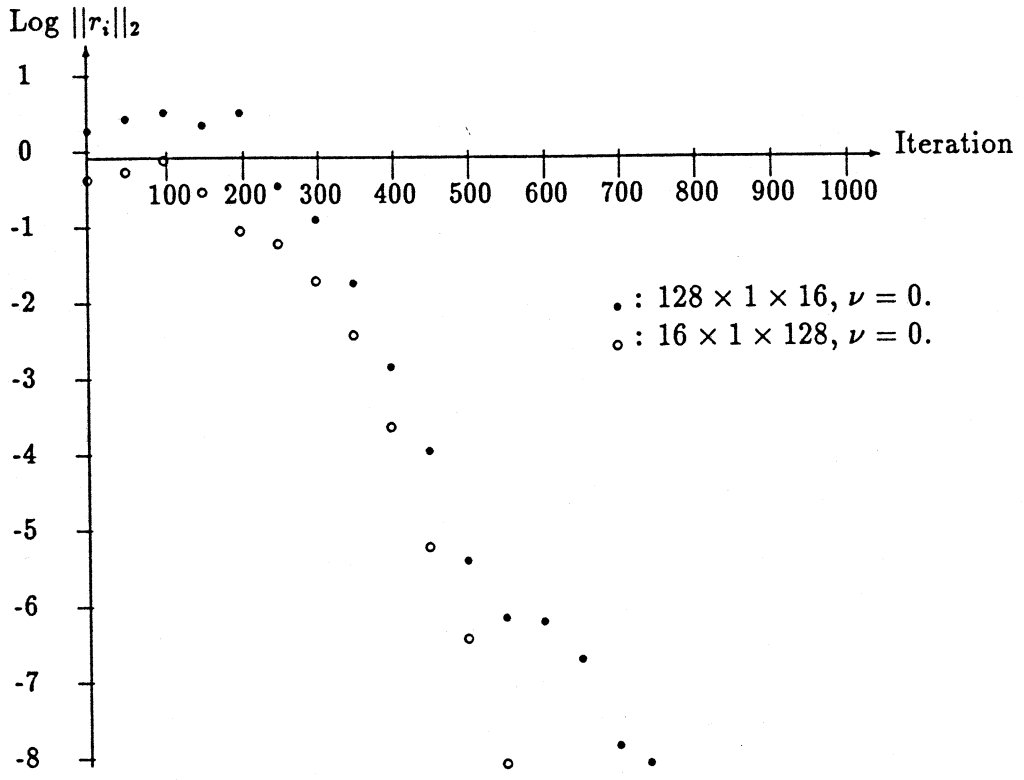

Figure 5: Evolution of the the normalized global residuals for pulling on a $10 \times 1 \times 10$ plate discretized by elements of order $1 \times 1 \times 1$. The face at $x_0 = 0$ is fixed and the load applied at the face $x_0 = 10$. Two different discretizations are shown.

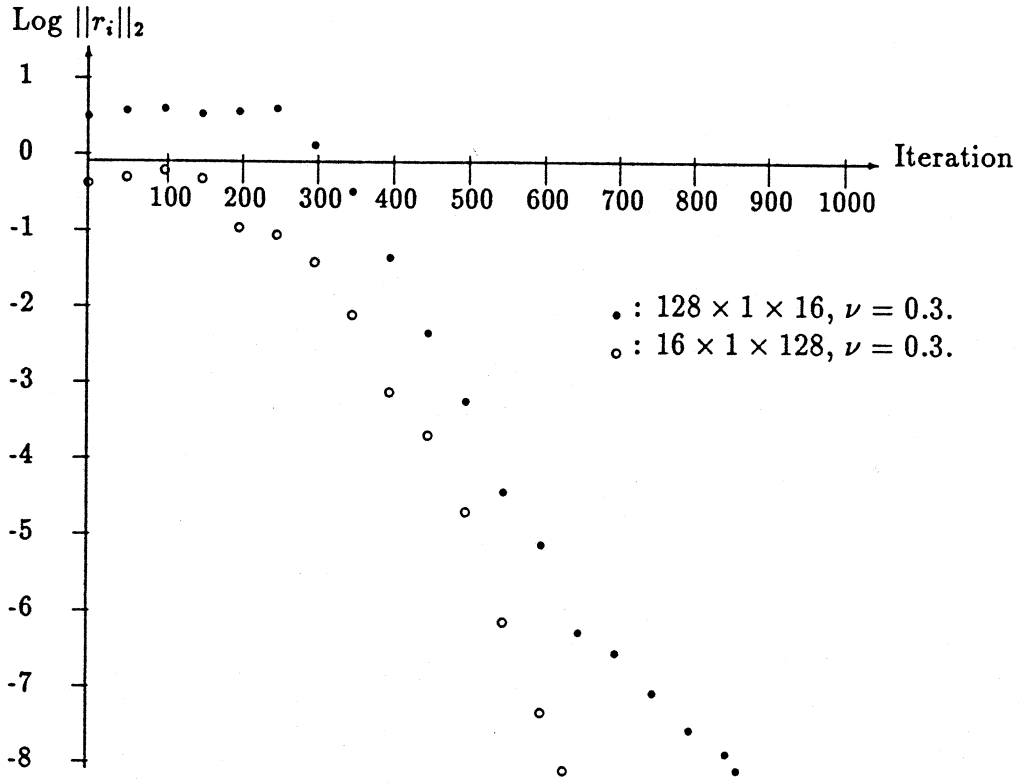

Figure 6: Evolution of the the normalized global residuals for pulling on a $10 \times 1 \times 10$ plate discretized by elements of order $1 \times 1 \times 1$. The face at $x_0 = 0$ is fixed and the load applied at the face $x_0 = 10$. Two different discretizations are shown.

ν	N_0	$\|\|\bar{r}\|\|_2 > 1.0$	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$
0	1		37	39
	2		36	43
	4		58	71
	8		60	72
	16		68	91
	32	57	124	177
	64	107	244	351
	128	233	492	742
0.3	1		45	52
	2		54	65
	4		67	78
	8	19	71	86
	16	33	86	115
	32	67	151	212
	64	150	299	408
	128	313	600	864

Table 10: The number of conjugate gradient iterations required for the three normalized global residuals. The mesh discretization is $N_0 \times 1 \times 16$ and the order of the elements is $1 \times 1 \times 1$. The interface at $x_0 = 0$ is fixed and a pulling load applied at $x_0 = 10$.

The constant of proportionality for the number of iterations to convergence for varying the discretization along axis 0 increases almost in direct proportion to the logarithm of the norm of the global residual. However, changing the discretization along axis 2, which is orthogonal to the axis of the pulling force, increases the number of iterations to convergence considerably less. Increasing the value of ν from 0 to 0.3 increases the number of iterations to convergence. The increase is approximately 20% for axis zero and 15% along axis two.

In order to investigate the influence of the interpolation order of the elements experiments as reported in Table 13 were performed. In both cases the order and the discretization were varied such that $N_i p_i = 60$.

The dependence of the number of iterations upon the element order can be expressed as

$$N_{cg} = c(i, error, \nu) N_i p_i^{1.5}$$

The structure of the global stiffness matrix for the discretizations $N_0 \times 1 \times 16$ and $16 \times 1 \times N_2$ for the same values of N_0 and N_2 can be made the same with the exception of the boundary conditions. But, the values in the corresponding matrix positions are not the same. The number of iterations required for convergence of the conjugate gradient method indicates that the condition number for the global stiffness matrix only depends on the shape of the discretization by a constant factor. Note that for a given value of N_i, the aspect ratios of the elements are the same for the two discretizations, but their orientation

ν	N_2	$\|\|\bar{r}\|\|_2 > 1.0$	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$
0	1		16	16
	2		40	61
	4		57	75
	8		61	87
	16		68	91
	32		120	147
	64		226	284
	128		447	550
0.3	1	11	20	23
	2	25	46	91
	4	33	69	85
	8	34	74	99
	16	22	79	115
	32	38	142	170
	64		262	321
	128		515	631

Table 11: The number of conjugate gradient iterations required for the three normalized global residuals. The mesh discretization is $16 \times 1 \times N_2$ and the order of the elements is $1 \times 1 \times 1$. The interface at $x_0 = 0$ is fixed and a pulling load applied at $x_0 = 10$.

Direction	$\nu = 0$			$\nu = 0.3$		
	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$	$\frac{\|\|\bar{r}\|\|_2=10^{-8}}{\|\|\bar{r}\|\|_2=10^{-5}}$	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$	$\frac{\|\|\bar{r}\|\|_2=10^{-8}}{\|\|\bar{r}\|\|_2=10^{-5}}$
0	3.8	5.8	1.5	4.6	6.7	1.5
2	3.4	4.1	1.2	3.9	4.6	1.2
Ratio $\frac{\text{Dir. 0}}{\text{Dir. 2}}$	1.12	1.41	1.25	1.18	1.46	1.25

Table 12: Constants of proportionality for the number of conjugate gradient iterations for the range $16 \leq N_i \leq 128$ of element discretizations.

Discretization Interpolation	$N_0 \times 1 \times 16$ $p_0 \times 1 \times 16$	$16 \times 1 \times N_2$ $16 \times 1 \times p_2$
$p_i = 1$	382	302
$p_i = 2$	413	348
$p_i = 3$	555	495
$p_i = 4$	662	515
$p_i = 5$	872	726

Table 13: The dependence of the the number of conjugate gradient iterations upon the element order and discretization for pulling of a plate. Convergence criteria: a normalized global residual less than 10^{-8}. Poisson ratio $\nu = 0.3$.

of the elements with respect to the force field is different.

5.2.2 Bending

As in the case with a pulling load the $10 \times 1 \times 10$ domain is discretized by $N_0 \times 1 \times 16$ elements, or by $16 \times 1 \times N_2$ elements of order $p_0 \times 1 \times 1$ and $1 \times 1 \times p_2$, respectively. For the study of the dependence of the number of iterations to convergence upon the direction of discretization the element order was $1 \times 1 \times 1$. The results are given in Tables 14 and 15. In both cases the Poisson ratio $\nu = 0.3$.

Unlike with a pulling load, no directional dependence of the number of iterations to convergence was observed in the bending case. The improvement of the global residual in the first several iterations is considerably less with the bending load than with the pulling load. However, the rate of convergence improves as the computations proceed such that the convergence rate for a bending load may actually be higher than for the pulling load. Hence, comparing Tables 10 and 15, and Tables 11 and 15, twice as many iterations may be required before a reduction in the global residual starts to take place, about 50% more iterations required for a global residual of at most 10^{-5}, but only about 10% more iterations required for a global residual of at most 10^{-8}. The evolution of the normalized global residual is shown in Figure 7.

For a given value of the global residual the number of iterations depends approximately linearly upon the number of elements. The constants of proportionality for two different residuals are given in Table 16. As in the pulling case the constants are computed for $16 \leq N_i \leq 128$.

A few experiments were also made to study the influence of the order of the elements upon the number of iterations to convergence. Table 17 shows some of these results. The number of iterations to convergence depends upon the order of the elements as

$$N_{cg} \sim N_i p_i^{1.5}.$$

N_0	$\|\|\bar{r}\|\|_2 > 1.0$	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$
1	36	46	51
2	58	103	111
4	69	120	139
8	87	124	139
16	103	140	155
32	170	231	259
64	315	431	489
128	621	862	969

Table 14: The number of conjugate gradient iterations required for the three normalized global residuals. The mesh discretization is $N_0 \times 1 \times 16$ and the order of the elements is $1 \times 1 \times 1$. The interface at $x_0 = 0$ is fixed and a bending load applied at $x_0 = 10$. Poisson ratio $\nu = 0.3$.

N_2	$\|\|\bar{r}\|\|_2 > 1.0$	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$
1		23	26
2	14	55	62
4	60	100	110
8	83	119	135
16	103	140	155
32	170	232	258
64	313	418	484
128	617	857	960

Table 15: The number of conjugate gradient iterations required for the three normalized global residuals. The mesh discretization is $16 \times 1 \times N_2$ and the order of the elements is $1 \times 1 \times 1$. The interface at $x_0 = 0$ is fixed and a bending load applied at $x_0 = 10$. Poisson ratio $\nu = 0.3$.

Direction	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$
0/2	6.5	7.2

Table 16: Constants of proportionality for the number of conjugate gradient iterations as a function of direction of discretization. Poisson ratio $\nu = 0.3$.

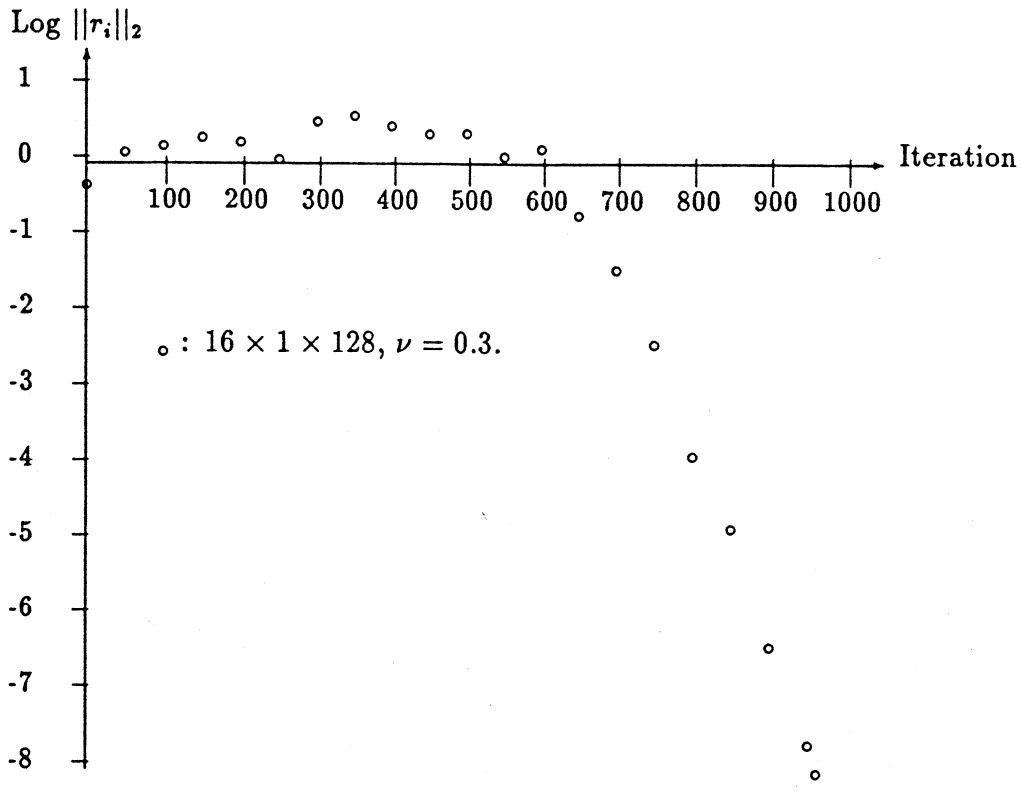

Figure 7: Evolution of the the normalized global residuals for bending a $10 \times 1 \times 10$ plate discretized by elements of order $1 \times 1 \times 1$. The face at $x_0 = 0$ is fixed and the load applied at the face $x_0 = 10$. Poisson ratio $\nu = 0.3$

Discretization Interpolation	$16 \times 1 \times N_0$ $1 \times 1 \times p_0$	$16 \times 1 \times N_2$ $1 \times 1 \times p_2$
$p_i = 1$	671	455
$p_i = 2$	784	531
$p_i = 3$	965	919
$p_i = 4$	1153	1104
$p_i = 5$	1525	1195

Table 17: The number of conjugate gradient iterations to convergence for a $N_0 \times 1 \times 16$ and $16 \times 1 \times N_2$ element discretization of order $p_0 \times 1 \times 1$ and $1 \times 1 \times p_2$ respectively ($N_0 p_0 = 60$ and $N_2 p_2 = 60$). A bending load is applied at $x_0 = 10$, with $x_0 = 0$ fixed. Convergence criteria: a normalized global residual less than 10^{-8}.

6 Summary

In implementing the finite element method on a data parallel computer the choice of logical unit affects both the concurrency and the storage requirement per processor in a significant way. By representing an unassembled nodal point per element as the logical unit the concurrency is higher than if a processor represents an unassembled element by a factor equal to the number of nodes per element. The required storage per processor is less by the same factor. The data parallel implementation used for the experiments use the nodal point per processor representation.

The results from the simulations with a pulling force applied to a beam, and a pulling and a bending force applied to a plate showed that the number of iterations to convergence for a diagonally scaled conjugate gradient method increases with an increased Poisson ratio. The number of iterations also increased linearly with the number of elements along the axis with the smallest grid point spacing. The constant of proportionality depended upon the element orientation with respect to the force field for pulling of the plate, but was independent of the orientation for a bending force. The number of iterations also increased in proportion to the logarithm of the inverse of the normalised global residual. The dependence of the number of iterations to convergence upon the element order approximately followed the relationship

$$N_{cg} = c(\nu, dir)(-log(||r||_2))Np^{\alpha}.$$

where α was in the range $1.4 - 1.5$. These results leads to the conjecture that the condition number of the global stiffness matrix depends upon the element order p of Lagrange elements as p^d, where d is the number of spatial dimensions.

References

[1] Owe Axelsson and V.A. Barker. *Finite Element Solutions of Boundary Value Problems.* Academic Press, 1984.

[2] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31:333–390, 1977.

[3] Billy L. Buzbee, Gene H. Golub, and C W. Nielson. On direct methods for solving Poisson's equations. *SIAM J. Numer. Anal.*, 7(4):627–656, December 1970.

[4] M.Y. Chan. Dilation-2 embeddings of grids into hypercubes. Technical Report UTDCS 1-88, Computer Science Dept., University of Texas at Dallas, 1988.

[5] M.Y. Chan. Embeddings of 3-dimensional grids into optimal hypercubes. Technical report, Computer Science Dept., University of Texas at Dallas, 1988. To appear in the Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, March, 1989.

[6] Monty M. Denneau, Peter H. Hochschild, and Gideon Shichman. The switching network of the TF-1 parallel supercomputer. *Supercomputing Magazine*, 2(4):7–10, 1988.

[7] W. Morven Gentleman. Some complexity results for matrix computations on parallel processors. *J. ACM*, 25(1):112–115, January 1978.

[8] A. George. Nested dissection of a regular finite element mesh. *SIAM J. on Numer. Anal.*, 10:345–363, 1973.

[9] Anne Greenbaum, Congming Li, and Han Zheng Chao. Parallelizing preconditioned conjugate gradient algorithms. Technical report, Courant Institute of Mathematical Sciences, New York University, November 1988.

[10] I. Havel and J. Móravek. B-valuations of graphs. *Czech. Math. J.*, 22:338–351, 1972.

[11] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[12] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *1987 International Conf. on Parallel Processing*, pages 188–191. IEEE Computer Society, 1987.

[13] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in Boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computing*, 7(3), December 1989. Technical Report YALEU/DCS/RR-746, Department of Computer Science Yale University, September 1989. This is a revision of Technical Report YALEU/DCS/RR-689 March 1989, Technical Report DA89-1, Thinking Machines Corp., September 1989.

[14] Roger W. Hockney and C.R. Jesshope. *Parallel Computers.* Adam Hilger, 1981.

[15] 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.

[16] S. Lennart Johnsson. Solving narrow banded systems on ensemble architectures. *ACM TOMS*, 11(3):271–288, November 1985.

[17] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987.

[18] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Statist. Comput.*, 8(3):354–392, May 1987.

[19] S. Lennart Johnsson. *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures.* Morgan Kaufman, 1989.

[20] S. Lennart Johnsson, Tim Harris, and Kapil K. Mathur. Matrix multiplication on the Connection Machine. In *Supercomputing 89.* ACM, November 1989. Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-736, Technical Report NA89-3, Thinking Machines Corp., September 1989.

[21] 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.

[22] S. Lennart Johnsson and Peggy Li. Solutionset for AMA/CS 146. Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[23] 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*, 1989. Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-743, Technical Report CS89-1, Thinking Machines Corp., December, 1988.

[24] Kapil K. Mathur and S. Lennart Johnsson. The finite element method on a data parallel computing system. *Int. J. of High-Speed Computing*, 1(1):29–44, May 1989. Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-742, Thinking Machines Corp., Technical Report CS89-2.

[25] Michael Metcalf and John Reid. *Fortran 8X Explained.* Oxford Scientific Publications, 1987.

[26] J. Tinsley Oden and Graham F. Carey. *Finite Elements: Mathematical Aspects*, volume IV. Prentice-Hall, 1983.

[27] Tekla S. Perry. Intel's secret is out. *IEEE Spectrum*, 26(4):22–28, 1989.

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

[29] Arnold L. Rosenberg. Preserving proximity in arrays. *SIAM J Computing*, 4:443–460, 1975.

[30] Thinking Machines Corp. **Lisp Release Notes*, 1987.

[31] O.C. Zienkiewicz. *The Finite Element Method.* McGraw-Hill, 1967.