Element Order and Convergence Rate of the Conjugate Gradient Method for Data Parallel Stress Analysis

Kapil K. Mathur and S. Lennart Johnsson

YALEU/DCS/TR-733
September 1989

To appear in the Proceedings of Supercomputing 1989

Element Order and Convergence Rate of the Conjugate Gradient Method for Data Parallel Stress Analysis

Kapil K. Mathur and S. Lennart Johnsson*
Thinking Machines Corp.
245 First Street,
Cambridge, MA 02142
Mathur@think.com, Johnsson@think.com

Abstract

A data parallel formulation of the finite element method is described. The data structures and the algorithms for stiffness matrix generation and the solution of the equilibrium equations are presented briefly. The generation of the elemental stiffness matrices requires no communication, even though each finite element is distributed over several processors. The conjugate gradient method with a diagonal preconditioner has been used for the solution of the resulting sparse linear system. This formulation has been implemented on the Connection Machine® model CM–2. The simulations reported in this article investigate the influence of the mesh discretization and the interpolation order on the convergence behavior of the conjugate gradient method. A linear dependence of the convergence behavior on the mesh discretization parameter is observed. In addition, the convergence rate depends on the interpolation order p as $O(p^{1.6})$. The peak floating point rate (single–precision) for the evaluation of the stiffness matrix is approximately 2.4 Gflops s^{-1}. The iterative solver peaks at nearly 850 Mflops s^{-1}.

Keywords: finite element method, data parallel algorithms, preconditioned conjugate gradient method.

*Also affiliated with Department of Computer Science, Yale University, New Haven CT 06520

1 Introduction

This article summarizes some numerical experiments with a data parallel implementation of the conjugate gradient method in the context of sparse linear systems. The application used in this study is three dimensional stress analysis of domains discretized by brick elements. The conjugate gradient method is used for the solution of the equilibrium equations. The resulting linear system is sparse. The structure of the sparse system is dependent on the discretization of the domain and the interpolation order of the elements used to construct the mesh. An iterative solver was chosen for the solution of the resulting sparse system because it offers a high degree of concurrency with good load balance. Moreover, the data structure used for the evaluation of the elemental stiffness matrices can also be used by the iterative solver. There is no fill–in. For the regular domain discretizations considered here, the sparse matrix–vector product required by the iterative solver involves local interactions only.

The data parallel implementation of the finite element method on the Connection Machine CM–2 is described briefly. A performance analysis of the implementation is reported. Next, several different sets of simulations are presented. The primary intent of these simulations is to investigate the influence of the discretization parameter, h, and the interpolation order, p, on the convergence behavior of the conjugate gradient method. The approximation error decreases with

the element order and the mesh resolution. However, since the condition number of the global stiffness matrix (κ) increases with the mesh resolution ($O(h^{-2})$ [1]) and the interpolation order [2] the convergence rate of the iterative method is also affected ($O(\sqrt{\kappa})$).

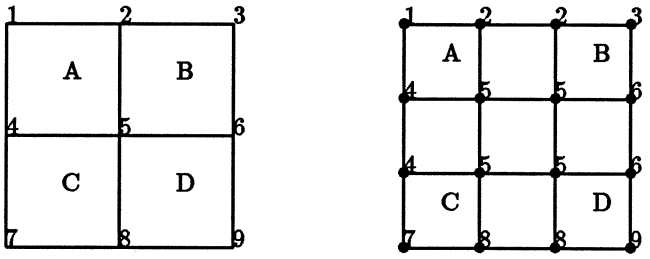
2 The Model Architecture

The Connection Machine Model CM–2 is a data parallel computing system [5,3]. The primary elements of the computing system include a front–end computer, a parallel processing unit of 64K bit serial processors and a high–performance data parallel I/O system. The front–end computer provides the development and execution environment. The data parallel operations are executed on the bit serial processors. Each bit serial processor has its own memory (32 Kbytes per processor which corresponds to a total memory of 2 Gbytes). When the number of processors available are less than the number of data elements required by the parallel data structures, the computing system operates in a virtual processor mode. In this mode, the application program is presented with a larger number of virtual processors, each with a correspondingly smaller memory. Each physical processor is made to simulate an appropriate number of virtual processors. These virtual processors time share the physical processor. The ratio of the number of virtual processors to the physical processors is referred to as the *virtual processor ratio.*

The processors are interconnected by a network. General patterns of communication between the processors is supported by a router. More regular patterns of communications is supported by special communication software, which takes advantage of the lattice emulation capability of the network. For example, array data structures configure the processors of the Connection Machine as a lattice with the same number of dimensions as the array. The peak rate for communication between adjacent processors in the lattice is approximately 15 Gbytes s^{-1}. This rate of communication depends upon the dimensionality of the lattice and the virtual processor ratio.

3 A Data Parallel Implementation

The physical domain of the applications discussed here have been discretized by Lagrange brick elements. The data parallel implementation of the finite element method maps an unassembled nodal point of the mesh on to a processor. 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. Figure (1) shows this mapping for a two dimensional finite element mesh with four bilinear elements labeled A, B, C, and D. The mesh has nine nodes labeled one through nine. 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.



Finite Element Mesh

The Unassembled Nodal Point Representation

Figure 1: Mapping the physical domain composed of brick/rectangular elements on to the data parallel architecture. In the example shown above, the finite element mesh comprises of four bilinear elements labeled A–D. The nodes are labeled one to nine. The processors of the data parallel architecture are represented by dots.

In this mapping scheme, each element has a subset of processors working towards the generation of the elemental stiffness matrices and in the evaluation of the sparse matrix–vector product during the solution phase. Moreover, if the elemental stiffness matrices are not explicitly assembled into a global stiffness matrix, the storage required to store these matrices is shared evenly by the subset of processors. Each processor stores the rows of the elemental stiffness matrix corresponding to the unassembled nodal point represented by the processor. This corresponds to 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. This is especially advantageous for three dimensional discretizations because of limited local storage per processor of the model data parallel architecture. When the elemental stiffness matrices are distributed evenly over a subset of processors, higher order elements can

be used in the construction of the mesh [7].

The data parallel implementation of the finite element method can be divided into two distinct sections – the evaluation of the elemental stiffness matrices and the solution of the resulting sparse linear system. For the mapping described here, the evaluation of the elemental stiffness matrices requires no communication if the numerical quadrature for each matrix element is performed sequentially. However, several matrix elements can be computed in parallel. This is in addition to the concurrency already present between different finite elements. Further details are available in [7]. The resulting sparse linear system has been solved by a conjugate gradient method with diagonal scaling. In the data parallel implementation of the conjugate gradient method, the main computational and data communication effort is in the sparse matrix–vector product of the form

$$\{r\} = \{b\} - [A]\{x\}, \tag{1}$$

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

$$[A] = \sum_{i}^{n} \left[A^{(i)}\right]. \tag{2}$$

For the mapping scheme used in the implementation, this sparse matrix–vector product involves:

1. Accumulation of the unknowns from the processors representing the unassembled nodes. All processors in the subset of processors forming the element require the unknowns from every other processor in this subset. This type of communication is often termed as a segmented "all to all" broadcast [6] and can be implemented very efficiently by the use of nearest neighbor communication when the processors of the data parallel architecture are configured as a lattice. In addition to the matrix containing the unassembled rows of the nodal point represented by the processor, after a segmented all to all broadcast every processor also stores a vector of length nu containing the accumulated unknown vector.

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.

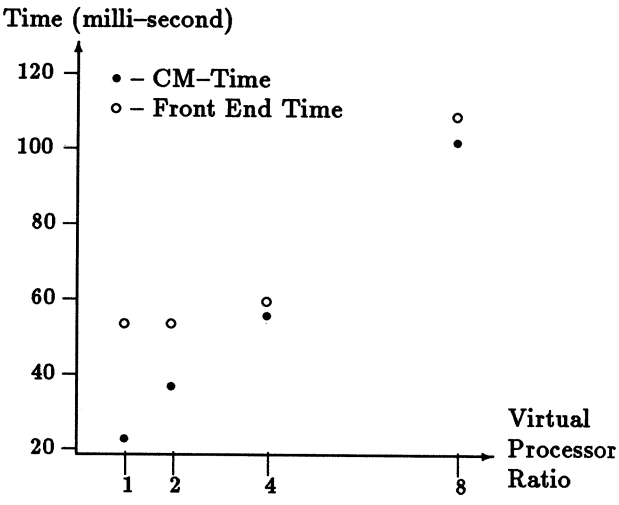


Figure 2: The front–end and the CM time for the evaluation of the elemental stiffness matrices as a function of the virtual processor ratio. All reported times are in milli–seconds for single precision floating point operations.

For the example two dimensional mesh shown in Figure (1) and in simulations involving stress analysis, the three sections described above 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).

Performance: To evaluate the performance of the data parallel implementation, several simulations involving three dimensional stress fields were performed. Single precision floating point arithmetic was used in this timing analysis. The finite element meshes were constructed using three dimensional Lagrange elements. For these simulations, $u = 3$ and $n = (p+1)^3$ where p is the interpolation order of the Lagrange elements. Figure (2) shows the CM time and the front–end time for the generation of the elemental stiffness matrices for trilinear brick elements as a function of the virtual processor ratio. A peak performance corresponding to 2.4 Gflops s^{-1} is obtained at a virtual processor ratio of eight with a CM utilization of over

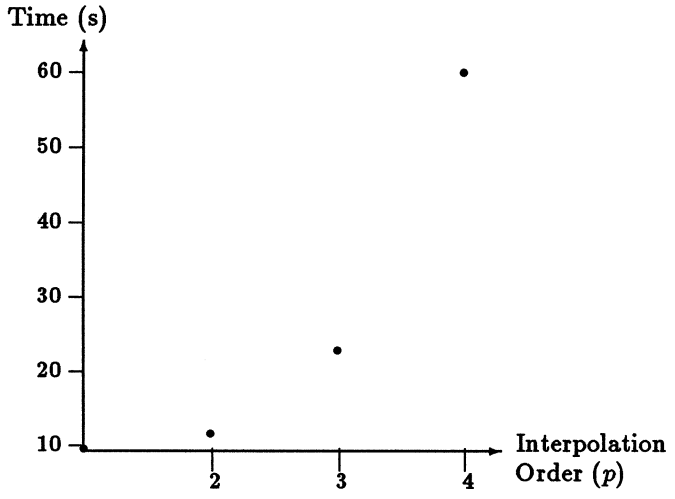


Figure 3: The influence of interpolation order, p, on the time (CM) required to generate the elemental stiffness matrices for three–dimensional Lagrange elements. All reported times are in seconds and correspond to single precision floating point operations.

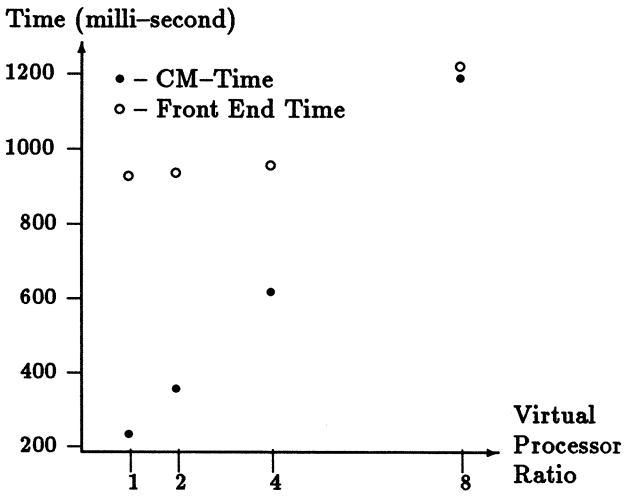


Figure 4: The time per conjugate gradient iteration (with diagonal scaling) as a function of the virtual processor ratio. All reported times are in milli–second and correspond to single precision floating point operations. The finite element meshes used in this analysis comprised of trilinear brick elements.

96%. After a virtual processor ratio of four, the difference between the front–end time and the CM time is approximately a constant. Most of this difference is the time used to evaluate the shape functions at the quadrature points in the local coordinate system. These functions are the same for *all* processors and are therefore evaluated on the front–end computer. The timings reported here were obtained using a Symbolics front–end. The CM utilization is expected to improve with a faster front–end computer. Figure (3) shows the influence of the interpolation order of the finite element on the CM time required to generate the elemental stiffness matrices. The plot clearly shows the $O(n^2)$ parallel floating point arithmetic complexity for the evaluation of the elemental stiffness matrices.

The performance of the conjugate gradient solver with a diagonal preconditioner is shown in Figure (4) as a function of the virtual processor ratio. The peak performance measured is approximately 850 Mflops s^{-1} at a virtual processor ratio of eight, where the CM utilization is approximately 95%. As before, as the virtual processor ratio increases, the front–end overhead reduces and consequently the CM utilization improves significantly. For most of the segments of the implementation of the iterative solver, the processor utilization is optimal. With the data representation of an unassembled nodal point per processor, the only segments of the implementation where some processors are inactive are during the assembly[1] and during the evaluation of the inner products[2].

4 Numerical Experiments

This section describes the numerical experiments performed to investigate the convergence behavior of the conjugate gradient method with a diagonal preconditioner. The first set of experiments simulated the three dimensional deformation field that results from the action of bending loads on a thin plate. The geometry of the physical domain corresponds to a square plate ten units long, one unit thick and ten units wide. One face in the length–width plane, i.e., one of the 10×10 faces, was fixed. Traction boundary conditions corresponding to a uniform unit load were applied to the opposite face. The domain was discretized by finite element meshes made up of trilinear bricks. These meshes had sixteen elements in the length dimension, one element in the thickness dimension and N elements in the width dimension, where N was varied from one to 128. Table (1) and Figure (5) shows the number of iterations required for the magnitude of the normalized

[1] only processors representing replicated nodal points take part in the assembly.

[2] only one of the replicated nodes takes part in the evaluation of the inner product.

N	$\|\|\bar{r}\|\|_2 > 1.0$	$\|\|\bar{r}\|\|_2 = 10^{-5}$	$\|\|\bar{r}\|\|_2 = 10^{-8}$
1	2	23	27
2	14	55	62
4	61	101	111
8	84	120	135
16	104	140	155
32	170	233	258
64	313	436	484
128	617	857	961

Table 1: The number of conjugate gradient iterations required for the magnitude of the normalized global residual to reach a value of 1×10^{-5} and 1×10^{-8}. Also shown in the table are the number of iterations required for the magnitude of the normalized global residual to become less than one. The mesh discretization for this set of simulations was $16 \times 1 \times N$, where $1 \leq N \leq 128$.

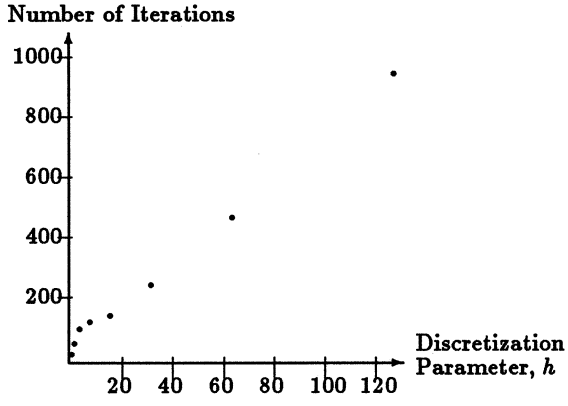


Figure 5: The convergence rate of the iterative solver as a function of the mesh discretization parameter, h. Convergence criterion : $||\bar{r}||_2 \leq 1.0 \times 10^{-8}$.

global residual to reach a value of 1×10^{-5} and 1×10^{-8} respectively. Clearly, with diagonal scaling, the number of iterations required for convergence grow almost linearly with N. This behavior agrees with analytical results [1,4]. Figure (6) shows the evolution of the normalized global residual during the iteration process for $N = 128$. The energy norm of the residual vector defined as

$$e^2 = x^T A x, \quad x \in R^n \tag{3}$$

is shown in the Figure (7).

The normalized global residual remains greater than one for a significant fraction of the iteration process. For the particular example of $N = 128$, more than 600 iterations are required for the normalized global residual to achieve a value of less than one. This segment of the iteration process can further be divided into two parts. In the initial phase of the iteration process, the

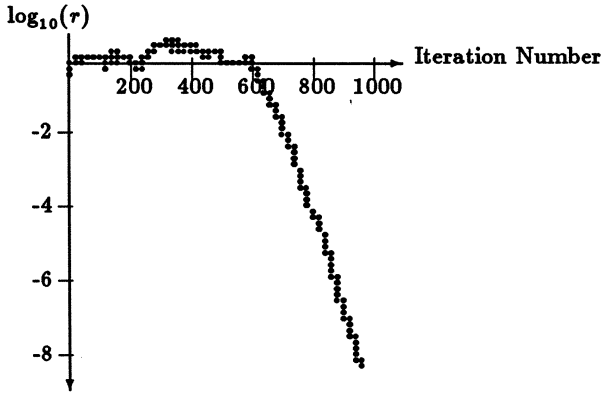


Figure 6: The evolution of the magnitude of the normalized global residual ($\log_{10}$) during the iteration process. The finite element mesh for this simulation was composed of trilinear brick elements. The mesh discretization in the length, thickness and width directions was $16 \times 1 \times 128$ respectively.

rate of decrease in the energy norm is small. This is observed as an increase in the magnitude of the normalized global residual. In the second part of the segment of the iteration process where the magnitude of the normalized global residual remains greater than one, the rate of change of the energy norm of the residual vector increases rapidly. Finally, after the normalized global residual becomes less than one, the rate of change in the energy norm of the residual vector is marginal even though the magnitude of the normalized global residual is decreasing steadily. The number of conjugate gradient iterations required for the magnitude of the normalized global residual to achieve a value of less than one for the above set of simulations are summarized in Table (1). Again, an almost linear dependence on N is observed. This linear dependence, along with the sluggish rate of decrease of the energy norm of the residual vector during the initial iteration phase shows the limitations of the diagonal preconditioner.

The influence of the interpolation order on the convergence behavior of the conjugate gradient method for the same physical domain and the same boundary conditions was also investigated by varying N, the number of elements in the width dimension and p, the interpolation order in the width direction such that the product $N \times p$ was a constant. This ensured that the total number of degrees of freedom of the discretized domains and consequently the sizes of the resulting linear system were fixed. Table (2) shows the evolution of the magnitude of the normalized global residual for the various values of N and p. Figure (8) shows the num-

p	$\|\|\bar{r}\|\|_2 \geq 1.0$	$\|\|\bar{r}\|\|_2 = 10^{-4}$	$\|\|\bar{r}\|\|_2 = 10^{-6}$
1	294	392	472
2	341	453	493
3	407	545	592
4	498	668	728
5	546	876	957
6	745	1196	1313

Table 2: The magnitude of the normalized global residual as a function of the iteration process. The finite element meshes used in the above simulations comprised of brick elements with $1 \times 1 \times p$ interpolation. The meshes discretizations were $16 \times 1 \times N$, such that $N \times p = 60$.

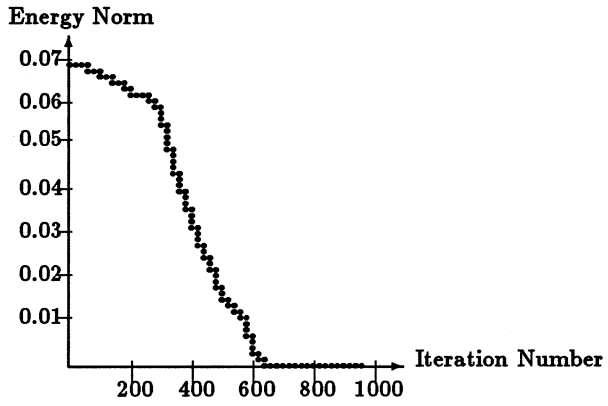


Figure 7: The evolution of the energy norm of the residual vector during the iteration process. The finite element mesh for this simulation was composed of trilinear brick elements. The mesh discretization in the length, thickness and width directions was $16 \times 1 \times 128$ respectively.

ber of conjugate gradient iterations required for convergence as a function of the interpolation order in the width direction. For the same size of the linear system, the number of conjugate gradient iterations required for convergence increase with the interpolation order. The dependence of the number of iterations on the interpolation order is more than linear. A least square analysis of Figure (9) suggests that

$$N_{\text{cg}} \sim O\left(p^{1.6}\right). \tag{4}$$

This is consistent with theoretical results and previous numerical experiments [8], where the convergence behavior of the conjugate gradient method with diagonal scaling was investigated for three dimensional Lagrange elements for the case where the loading was primarily of "pulling" type.

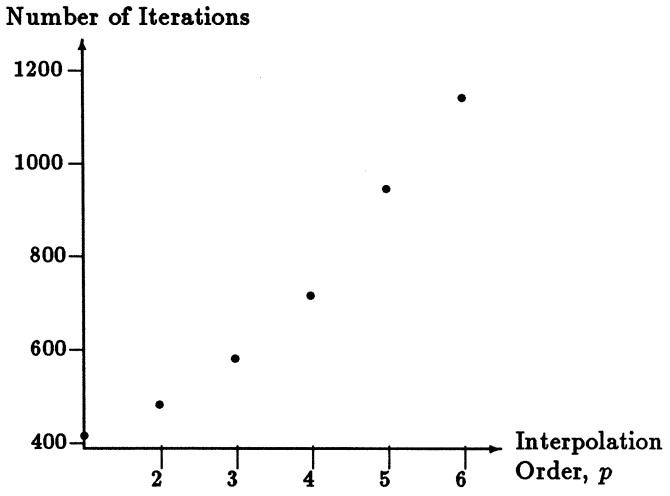


Figure 8: The number of iterations required for convergence as a function of the interpolation order in the width direction. Convergence criterion : $||\bar{r}||_2 \leq 1.0 \times 10^{-6}$. The finite element meshes used in the above simulations comprised of brick elements with $1 \times 1 \times p$ interpolation. The meshes discretizations were $16 \times 1 \times N$, such that $N \times p = 60$.

5 Summary

A data parallel implementation of the finite element method is described using the Connection Machine system, CM–2 as the model architecture. The mapping between the processors of the data parallel architecture and the logical units of data on which all the processors operate concurrently takes advantage of nearest neighbor communication when the processors are configured as a lattice. This mapping ensures uniform processor utilization and an efficient utilization of the storage. The mapping described here works very well for meshes comprising of brick elements. A more complex mapping is necessary for meshes composed of different types of elements and for domains with arbitrary geometries.

The convergence rate of the conjugate gradient method with diagonal scaling in the numerical experiments is inversely proportional to the mesh discretization, h. The influence of the interpolation order on the convergence rate also verifies analytical results.

References

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

[2] Ivo Babuska and H. C. Elman. *Some Aspects of Parallel Implementation of the Finite Element Method on Message Passing Architectures.* Technical Report UMIACS–TR–88–35; CS–TR–2030, University of Maryland, May 1988.

[3] Thinking Machines Corp. *Connection Machine Model CM–2 Technical Summary.* Technical Report HA87-4, Thinking Machines Corp., 1987.

[4] 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.

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

[6] 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.

[7] 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. Technical Report CS– 89/1, Thinking Machines Corp., December, 1988.

[8] 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. Thinking Machines Corp., Technical Report CS–89/2.