Data Parallel Algorithms for the Finite Element Method

Kapil K. Mathur and S. Lennart Johnsson

YALEU/DCS/TR-775
March 1990

To be published in the Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing
Chicago, 1989.

Data Parallel Algorithms for the Finite Element Method

Kapil K. Mathur and S. Lennart Johnsson
Thinking Machines Corporation
245 First Street
Cambridge, MA 02412

Abstract. A data parallel implementation of the finite element method is described. The focus of the presentation is on data mapping and data motion. The essential ideas of the data parallel implementation are developed for discretizations of regular domains by Lagrange elements of arbitrary order in two and three dimensions. A generalization to irregular domains is also presented. Implementations of the data mappings for both regular and irregular domains have been made on the Connection Machine® system model CM–2. Peak performances well in excess of 2 Gflops s^{-1} have been measured for the evaluation of the elemental stiffness matrices. The performance of the iterative solver is in the range of 600 – 850 Mflops s^{-1}.

1. Introduction. The most critical issue in high performance computing is data motion. On data parallel architectures a high storage bandwidth is accomplished by distributing the memory among thousands of units. Each such unit is often associated with a unique processor, as is the case for the Connection Machine system and all other currently available medium scale and data parallel architectures. An interconnection network is used on such architectures to provide the data interaction bandwidth necessary for the solution of elliptic problems. The effective utilization of this bandwidth is critical to obtain high performance. Effective use of the communication system requires that the locality inherent in many algorithms for elliptic problems should be preserved through an appropriate data mapping and data routing.

This article describes a data parallel algorithm of the finite element method for discretizations composed of isoparametric Lagrange elements of arbitrary order in two and three dimensions. Data layouts of the finite element discretization that allows for a highly concurrent computation of elemental stiffness matrices and solution of the equilibrium equations, as well as efficient communication for regular domains are presented. A data parallel algorithm that computes each elemental stiffness matrix concurrently, as well as different such matrices concurrently, with no communication is described. The resulting sparse linear system is solved by a conjugate gradient method. The algorithms have been implemented on the Connection Machine system, CM–2. The performance of this implementation is reported for the evaluation of the elemental stiffness matrices and for the solution of the sparse linear system. Finally, a brief description of one possible data parallel implementation of the finite element method for unstructured grids is presented with evaluation times for a representative mesh.

[0] *To be published in the Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing, Chicago, December 1989.*

2. The Model Data Parallel Architecture. The Connection Machine model CM–2 is a parallel processing unit with up to 64K processing elements (physical processors) and up to 2 Gbytes of primary storage. These processing elements execute data parallel operations. Front–end computers provide the development environment and execute scalar instructions. In addition, the Connection Machine system includes a high–performance parallel I/O system, and fast interactive graphics.

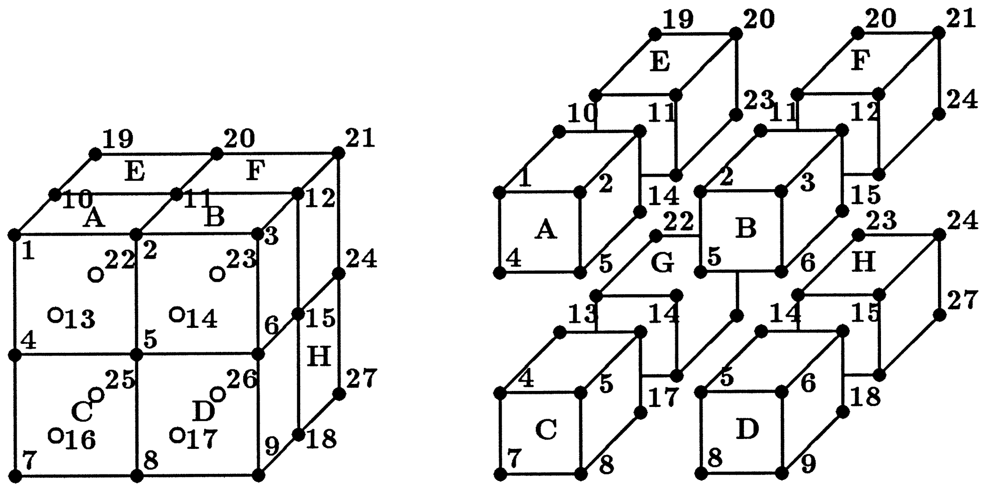
In the high level programming languages each processing element has its own local memory (up to 32K bytes). There is no shared memory, but the global address space is shared. When the number of objects in the parallel data structure exceed the number of *physical processors*, the Connection Machine system operates in a *virtual processor* mode. The number of objects in a parallel data structure is limited by the storage requirement per object and the total primary storage. In the virtual processor mode the application program is assigned as many virtual processors as there are objects, each virtual processor with a correspondingly smaller memory. This feature alleviates the applications program from the responsibility of conforming to the number of physical processors. Each physical processor is made to simulate the appropriate number of virtual processors. This number is referred to as the virtual processor ratio.

The virtual processor facility of the Connection Machine system supports the notion of virtual processor sets. All virtual processors associated with a data set is called a *virtual processor set.* The system software allows for the simultaneous existence of more than one virtual processor sets. In the context of high level programming languages, all arrays of the same shape reside in the same virtual processor set.

In the high level languages the processing elements of the Connection Machine computing system can communicate information among themselves by two mechanisms: lattice emulation and general routing. The lattice emulation makes use of the fact that lattices with axes lengths being powers of two are subgraphs of Boolean cubes, the interconnection network of the Connection Machine processor chips. The lattice emulation is a preprogrammed communication pattern explicitly managing the data transfer on the communication channels. The reader is referred to [1] for a general description of the Connection Machine computing system.

3. Data Layout. Three data representation schemes have been reviewed for the implementation of the finite element method on the Connection Machine system. The first two schemes make use of a single data layout for all operations: a processor is mapped to either an unassembled nodal point per finite element, or an unassembled finite element. The third scheme maps an unassembled element to a processor for the evaluation of the local data structures. Then, for solving the sparse linear system one processor represents an assembled nodal point.

The first two data layouts are very well suited for structured meshes and can use the lattice emulation features of the model architecture. The third scheme uses the general communications feature (router), which is typically slower than the lattice emulation due to contention for communication channels. The contention is caused by increased need for communication bandwidth due to a non–optimal data allocation,



A Finite Element Mesh

The Unassembled Nodal Point Representation

FIG. 1. *Mapping a physical domain composed of brick elements on to the data parallel architecture. In the example the finite element mesh consists of eight linear elements labeled A–H. The nodes are labeled 1–27. The processors of the data parallel architecture are configured as a* $4 \times 4 \times 4$ *array of processors.*

and potentially non–optimal routing and scheduling of messages.

3.1. An unassembled nodal point mapping. In this data representation, one processor of the data parallel architecture represents an unassembled nodal point. Nodal points shared between two or more elements are replicated on separate processors. Figure (1) illustrates this replication scheme for a finite element mesh composed of eight first order elements in three dimensions labeled A though H. This mesh has 27 nodes, labeled 1 to 27, and are mapped on to a $4 \times 4 \times 4$ array of processors. Nodal points labeled 2, 4, 6, 8, 10, 12, 16, 18, 20, 22, 24, and 26 are shared between two elements and are replicated twice, nodal points 5, 11, 13, 15, 17, and 23 are shared between four elements and are replicated four times, while nodal point 14 is shared by all elements and is replicated eight times.

This replication scheme extends naturally for isoparametric Lagrange elements of high order (Figure 2). Figure (2) shows a two-dimensional domain discretized by four biquadratic elements with 25 nodal points. The 5×5 grid of nodes is mapped on to a 6×6 array of processors. In general, nodal points lying on every line (surface in three dimensions) that is shared between elements are replicated. Note that the fraction of replicated nodes decreases rapidly with the order [2]. Mapping one processor to an unassembled nodal point has several advantages over the other mapping strategies. First, the processor utilization is optimal for the evaluation of the elemental stiffness matrices, and is very close to optimal for the solution of the sparse linear system. Second, if iterative methods are used to solve the sparse linear system, then the explicit assembly of the global stiffness matrix can be avoided by assembling matrix–vector products instead. Consequently, the storage requirements per processor are

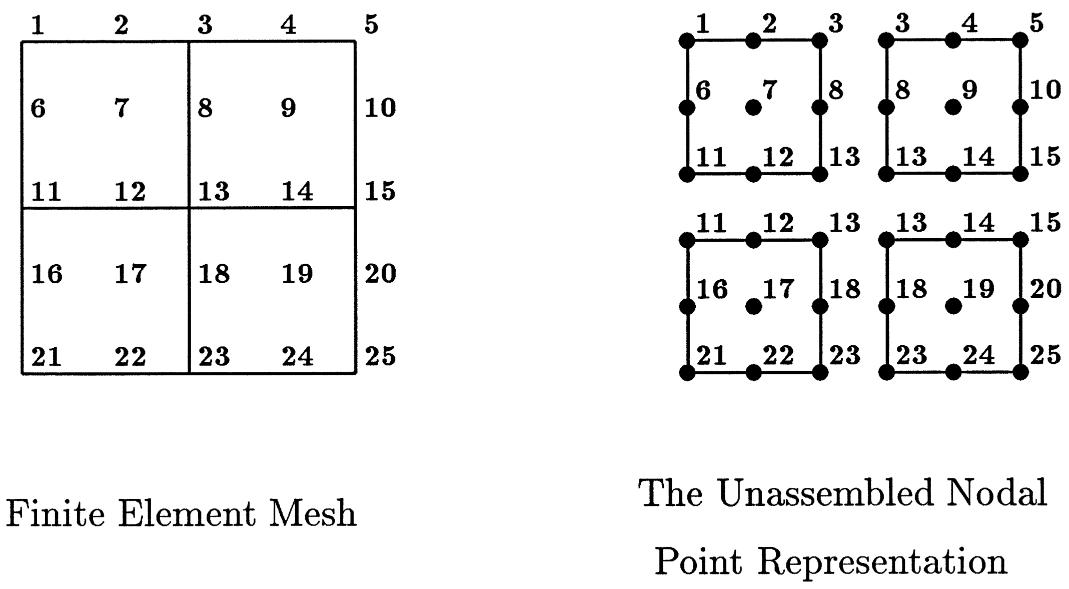


FIG. 2. *An unassembled nodal point data representation for a domain discretized by four biquadratic elements. The* 5×5 *grid of nodal points are mapped on a* 6×6 *processor layout.*

uniform. Third, the evaluation of the elemental data structures and the computation of the sparse matrix vector product for the iterative solver is now shared by a group of processors for each finite element. For example, eight processors evaluate the elemental stiffness matrix for the example illustrated in Figure (1). The concurrency of the computation is significantly higher than for an unassembled finite element per processor representation discussed in the next section. Consequently, the evaluation time is reduced proportionally. No inter–processor communication is required to evaluate the elemental data structures, even though the computation is being spread over a group of processors.

3.2. An unassembled finite element mapping. The main advantage of mapping an unassembled finite element to a processor is that it is easy to exploit the symmetry of the elemental stiffness matrices because the entire elemental stiffness matrix resides in the memory of a processor. However, there is a significant reduction in the concurrency. Reference [2] describes the storage requirements and the communication complexity of a data parallel implementation of the finite element method where one unassembled element is mapped on to a processor.

3.3. Unstructured discretizations. General finite element discretizations involve complicated geometries. The data parallel implementation of the finite element method involving unstructured grids is most conveniently modeled by two virtual processor sets, the first corresponding to the data set of all *unassembled* elements (one unassembled finite element per processor) and the second corresponding to the data set of all *assembled* nodes (one assembled node per processor). The analysis and the implementation presented in [2] is still valid for the evaluation of the elemental

```
loop over all finite elements
   loop over all quadrature points
      evaluate Jacobian and shape function
        derivatives for current quadrature point
      loop over rows in elemental stiffness matrix
         loop over columns in elemental stiffness matrix
            evaluate contribution to entry (row, column)
              of the elemental stiffness matrix
         end loop
      end loop
   end loop
end loop
```

FIG. 3. *Pseudo-code for evaluating the elemental stiffness matrices.*

```
do concurrently over all rows of the elemental
   stiffness matrix for all elements
   loop over all quadrature points
      evaluate Jacobian and shape function derivatives
         for the current quadrature point
      loop over columns in elemental stiffness matrix
         evaluate contribution to entry (row, column)
      end loop
   end loop
end do
```

FIG. 4. *The data parallel algorithm for evaluating the elemental stiffness matrices when one processor of the data parallel architecture represents an unassembled nodal point.*

stiffness matrices.

4. Evaluating the Elemental Stiffness Matrices. Figure (3) shows the sequential pseudo–code found in a typical finite element implementation. The loop over all finite elements vanishes naturally for a data parallel implementation. Moreover, in the data layout for which *a processor represents an unassembled nodal point*, n processors represent a finite element with n nodal points. These n processors share the evaluation of *one* elemental stiffness matrix. In the pseudo–code shown in Figure (3) one of the three inner nested loops can also be executed concurrently. Figure (4) shows the pseudo–code for the data parallel implementation for evaluating the elemental stiffness matrices. Notice that the loop over all the finite elements in the mesh is missing in the pseudo–code shown in Figure (4). Moreover, the loop that computes the rows of the unassembled elemental stiffness matrices is also being executed concurrently.

For three dimensional isoparametric brick elements, the unassembled nodal point

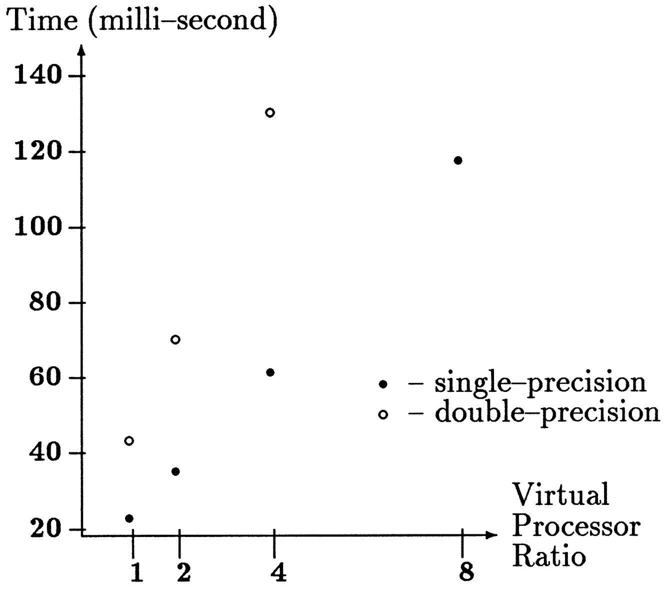


FIG. 5. *Time for the evaluation of the elemental stiffness matrices as a function of the virtual processor ratio. The peak performance corresponds to* 2.4 *Gflops* s^{-1}*, single precision, at a virtual processor ratio of eight.*

representation requires that each processor stores the u rows of the unassembled elemental stiffness matrix, where u is the number of degrees of freedom per nodal point. This results in a local matrix of size $u \times nu$ on every processor. In addition, each processor also stores the coordinates of all other nodal points forming the element in a local vector which is nu long. Once this "coordinate" vector is known the data parallel algorithm illustrated in Figure (4) requires no inter–processor communication.

Figure (5) shows the performance for evaluating the elemental stiffness matrices as a function of the virtual processor ratio, i.e., the number of unassembled nodal points residing on each physical processor. The peak performance is approximately 2.4 Gflops s^{-1}, single precision, at a virtual processor ratio of eight. For double precision floating-point operations, the measured peak performance is nearly 1.1 Gflops s^{-1}, at a virtual processor ratio of four.

Figure (6) shows the time taken to evaluate the elemental stiffness matrices when a processor represents an unassembled finite element. The numbers reported in Figure (6) correspond to the evaluation time for the symmetric half of the elemental stiffness matrices stored as a triangular matrix in packed form.

At a fixed virtual processor ratio, the time required to evaluate the elemental stiffness matrices is independent of the size of the finite element mesh. This feature is of particular significance for non–linear finite element simulations, where the evaluation of the elemental data structures dominate the total simulation time.

5. Data Parallel Implementation of the Conjugate Gradient Method. The sparse linear system representing the equilibrium equations has been solved by the method of conjugate gradients with diagonal scaling. The primary issues are an efficient evaluation of inner products and the sparse matrix–vector product. A de-

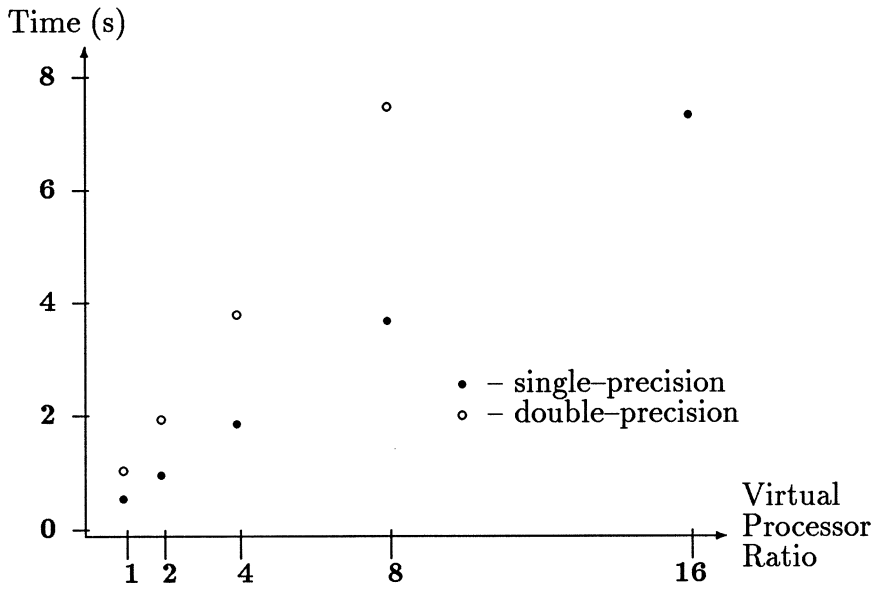


FIG. 6. *Time for the evaluation of the elemental stiffness matrices, when one processor represents an unassembled finite element, as a function of the virtual processor ratio.*

tailed description of the numerical behavior of the conjugate gradient method with diagonal scaling for sparse linear systems arising in stress analysis by the finite element method for beams, plates and bricks is reported in [3,4,5]. For the unassembled nodal point mapping diagonal scaling and the evaluation of inner products is straightforward. The sparse matrix–vector product is more involved and is described in detail for both regular and unstructured discretizations.

5.1. Sparse matrix–vector product. For the *unassembled nodal point* representation, apart from storing the u unassembled rows of the elemental stiffness matrix, each processor also stores the nodal loads (a vector of length u) corresponding to the nodal point represented by the processor. The sparse matrix–vector ($y = A \times x$) product then involves first accumulating the nodal values, x, from each processor in the group of n processors forming the element. This operation is a segmented "all–to–all" broadcast, which can be performed concurrently for all finite elements in the mesh. In addition, concurrency within the group of processors forming the finite element is also exploited [6]. The nodal values are accumulated in a local vector X, which is nu long. After the accumulation, a local matrix–vector product is performed, which yields the *unassembled* values for the vector y. Finally, an assembly of the local products results in the *assembled* values of the product vector at each processor. The accumulation phase and the assembly phase require inter–processor communication. For structured grids, the communication can be performed by using the lattice communication primitives. Figure (7) shows the time per conjugate gradient iteration as a function of the virtual processor ratio for finite element meshes composed of trilinear brick elements. As before, the virtual processor ratio represents the number of unassembled nodal points per physical processor. The time per conjugate gradient iteration in Figure (7) includes the arithmetic operations required for diagonal

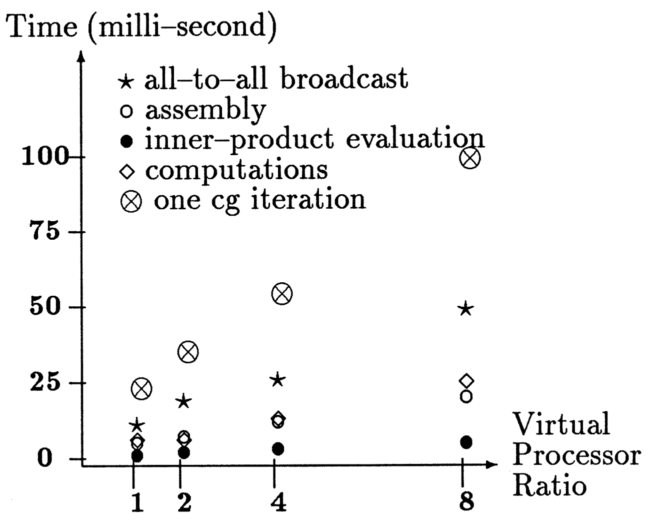


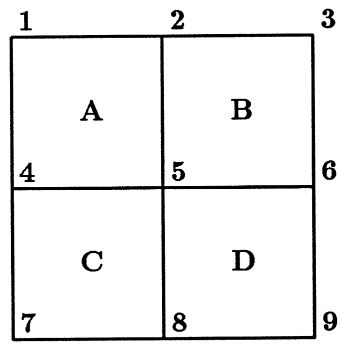
FIG. 7. *Time per conjugate gradient iteration (with diagonal scaling) as a function of the virtual processor ratio. The peak performance is approximately* 850 *Mflops* s^{-1}*, single precision, at a virtual processor ratio of eight.*

preconditioning. The peak performance of the solver corresponds to approximately 850 Mflops s^{-1}, single precision, at a virtual processor ratio of eight. More than half of the total time per iteration is spent in communication required for the segmented "all–to–all" broadcast, and the assembly procedure. Another timing study reported in [2] shows that on a Connection Machine system with 16K physical processors, the time per conjugate gradient iteration with diagonal scaling for an application with nearly 400,000 degrees of freedom is approximately 0.13 s for double precision floating-point operations.

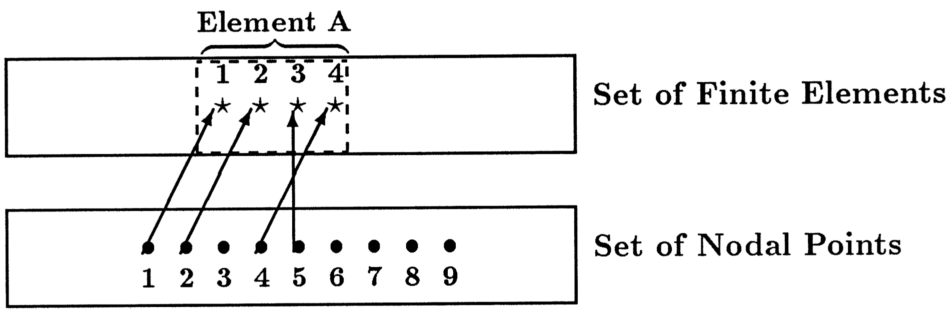
The two data mappings used to implement *unstructured discretizations* do not assemble the global stiffness matrix explicitly for evaluating the sparse matrix vector product. Instead a "gather" operation is performed to collect elemental data into local vectors. This operation involves data motion from the virtual processor set representing the assembled nodes to the virtual processor set representing the elements. Figure (8) illustrates the "gather" operation for a representative mesh composed of four elements and nine nodal points. After the elemental vectors have been collected, a local matrix–vector multiplication is performed and finally the local results are assembled into the global vector. Figure (9) shows the time taken per conjugate (with diagonal preconditioning) as a function of the number of assembled nodal points per physical processor. The timings reported in Figure (9) are for representative unstructured grids and have been measured on a Connection Machine system with 16384 physical processors. These timings include the time required to gather the elemental vectors and assemble the global vector after the product.

6. Summary.

Data parallel implementations of the finite element method for structured and unstructured grids in two and three dimensions are described. Lagrange elements of arbitrary order have been used for the presentation, The algorithms when a processor represents a finite element are also valid for Serendipity



Finite Element Mesh



```
cmf$layout elem_val(:serial,),icon(:serial,)
      dimension elem_val(4,512),icon(4,512)
      dimension node_val(1024)


      do i=1,4
         elem_val(i,:) = node_val(icon(i,:))
      enddo
```

FIG. 8. *CM-Fortran implementation of a "gather" operation for the example* 2×2 *discretization shown above. This pseudo-code assumes two data representations. In the first mapping, each processor represents an unassembled element. The ":serial" directive forces the four elemental values to be stored locally in the processor's memory. The second data representation maps an assembled nodal point to a processor. The interaction between the two data mappings is achieved through the use of the connectivity array,* **icon**.

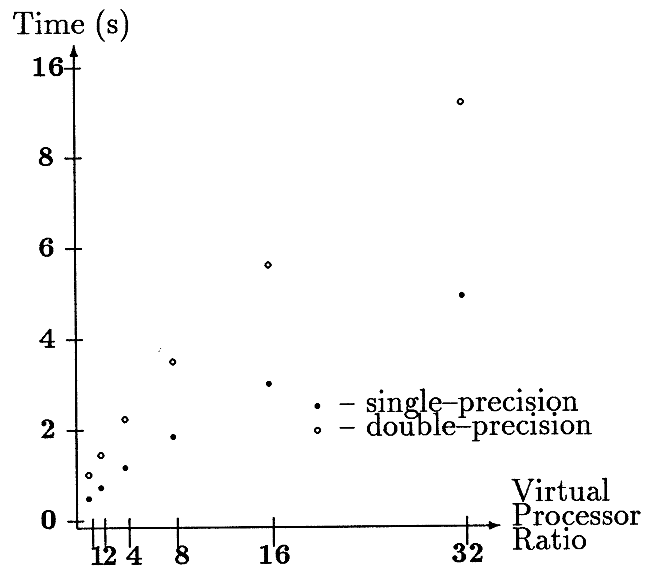


FIG. 9. *Time per conjugate gradient iteration (with diagonal preconditioning) for unstructured grids as a function of the virtual processor ratio. This data parallel implementation assumes two data representations. In the first mapping, each processor represents an unassembled element. The second data representation maps an assembled nodal point to a processor. The interaction between the two data mappings is included in the timings reported above.*

elements. When a processor represents a nodal point per element the concurrency for three dimensional problems is one to two orders of magnitude higher than with one processor per finite element. For structured grids, a data layout is presented that both exploits this fine grain concurrency, and the lattice emulation feature of the model data parallel architecture. For unstructured grids, two separate data layouts are used, one for the evaluation of the elemental stiffness matrices and the other for the solution of the sparse linear system arising out of the equilibrium equations.

For the elemental stiffness matrix computation an algorithm requiring no communication is presented. The performance is close to the peak arithmetic performance. A diagonally scaled conjugate gradient method is used for the solution of the equilibrium equations. The load balance is optimal. For a structured grid the execution time with one processor per unassembled nodal point is dominated by the communication among the group of processors representing a finite element. For an unstructured grid the communication is dominated by the inter–element communication required for the sparse matrix–vector product.

One significant result of this study is that data parallel architectures are very efficient for scientific computing. The high degree of concurrency inherent in the finite element method is compatible with the degree of concurrency necessary for data parallel architectures to give good performance [see also 7,8, 9]. However, special attention is required for optimizing data layout and communication between the processors of the data parallel architecture.

7. REFERENCES.

1. *Connection Machine Model CM–2 Technical Summary*, Technical Report, Thinking Machines Corporation, 1989.
2. S. L. Johnsson and K. K. Mathur, *Data Structures and Algorithms for the Finite Element Method on a Data Parallel Supercomputer*, Int. J. Numer. Meth. Engr., 29–4 (1990), pp. 881–908.
3. S. L. Johnsson and K. K. Mathur, *Experience with the Conjugate Gradient Method for Stress Analysis on a Data Parallel Supercomputer*, Int. J. Numer. Meth. Engr., 27–3 (1989), pp. 523–546.
4. K. K. Mathur and S. L. Johnsson, *Element Order and Convergence Rate of the Conjugate Gradient Method for Data Parallel Stress Analysis*, Proc. Supercomputing '89, (1989), pp. 337–343.
5. K. K. Mathur and S. L. Johnsson, *The Finite Element Method on a Data Parallel Computing System*, Int. J. High–Speed Comp., 1–1 (1989), pp. 29–44.
6. S. L. Johnsson and C. T. Ho, *Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes*, IEEE Trans. Comp., 38–9, (1989), pp. 1249–1268.
7. T. Belytschko, E. J. Plaskacz, J. M. Kennedy, and D. L. Greenwell, *Finite Element Analysis on the CONNECTION Machine*, Comp. Meth. in Appl. Meth. and Engr., in press.
8. T. Belytschko and E. J. Plaskacz, *SIMD Implementation of a Nonlinear Transient Shell Program with Partially Structured Meshes*, Int. J. Num. Meth. Engr., submitted.
9. C. Farhat, N. Sobh, and K. C. Park, *Transient Finite Element Computations on 65536 Processors: The Connection Machine*, Technical Report, CU–CSSC-89-06, Univ. of Colorado, 1989.