Mesh Decomposition and Communication Procedures for Finite Element Applications on the Connection Machine CM-5 System

Mesh Decomposition and Communication Procedures for Finite Element Applications on the Connection Machine CM-5 System

Zdeněk Johan
Kapil K. Mathur
S. Lennart Johnsson
Thomas J.R. Hughes

TR-08-94

April 1994

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

To appear in *The International Conference on High-Performance Computing and Networking*, HPCN Europe '94, Munich, April 18–20, 1994.

Mesh Decomposition and Communication Procedures for Finite Element Applications on the Connection Machine CM-5 System

Zdeněk Johan[1], Kapil K. Mathur[1],
S. Lennart Johnsson[1]* and Thomas J.R. Hughes[2]

[1] Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142, USA
[2] Division of Applied Mechanics, Stanford University, Stanford, CA 94305, USA

Abstract. The objective of this paper is to analyze the impact of data mapping strategies on the performance of finite element applications. First, we describe a parallel mesh decomposition algorithm based on recursive spectral bisection used to partition the mesh into element blocks. A simple heuristic algorithm then renumbers the mesh nodes. Large three-dimensional meshes demonstrate the efficiency of those mapping strategies and assess the performance of a finite element program for fluid dynamics.

1 Introduction

Data distribution is a crucial issue when implementing finite element techniques on distributed-memory parallel computers. Communication between processing nodes can become a bottleneck if the finite element data structures are not carefully mapped to the processing nodes. In order to minimize this bottleneck, we have developed a set of data mapping strategies and implemented them on the Connection Machine CM-5 system. Special library communication routines taking advantage of data locality to reduce data transfer between processing nodes are used to perform the gather and scatter operations found in finite element applications. Decomposition timings for large tetrahedral meshes are presented, as well as the effect of data mapping on the performance of a finite element program for computational fluid dynamics.

2 Data Mapping Strategies

Both elements and nodes of an unstructured mesh are mapped onto the vector units of the CM-5 system. We have designed a two-step procedure which performs these mappings:

1. First, the mesh is decomposed into element blocks made of adjacent elements.
2. The mesh nodes are then mapped onto the vector units using the mesh partitioning as a criterion for chosing the placement of each node.

The objective of these mappings is to achieve as much locality between the nodes and the elements as possible to minimize data transfer through the CM-5 data network. In order to achieve the best computational load balance possible in the finite element program itself, we constrain the elements and the nodes to be uniformly distributed across the vector units, i.e., all vector units hold the same number of elements (resp. nodes) except for the last one which gets whatever elements (resp. nodes) remain. The implementation of both mapping strategies is done on the CM-5 system itself.

2.1 Mesh Partitioning

The recursive spectral bisection (RSB) algorithm was chosen as the basis of the data mapping strategies described in this paper. The RSB algorithm was proposed by Pothen, Simon and Liou for reordering sparse matrices [1]. Simon then applied it to unstructured mesh partitioning [2]. The RSB algorithm has since found wide acceptance in the scientific community because of the high-quality partitionings it generates.

The RSB algorithm is based on a graph representation of the mesh topology. It is therefore insensitive to regions of highly concentrated elements or to element distorsion. In our implementation, the graph is generated through the *dual mesh connectivity*, which identifies the elements sharing a face with a given element.

* Also affiliated with the Division of Applied Sciences, Harvard University

In this representation, the mesh elements become the graph vertices and the internal faces correspond to the graph edges. The mesh partitioning is performed using an iterative process which decomposes the whole mesh into two partitions, each of which in turn is decomposed into two partitions, and so on. This process ends when there are as many partitions as vector units in the CM-5 configuration considered. Each iteration of the process just described involves several computational steps:

1. Possible disconnections in a partition are identified using a frontal algorithm.
2. The smallest non-zero eigenvalue and its associated eigenvector (also called the *Fiedler vector*) of the Laplacian matrix $\mathbf{L}$, defined as

$$L_{ij} = \begin{cases} -1, & \text{if elements } i \text{ and } j \text{ share a face;} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

$$L_{ii} = -\sum_{\substack{j=1 \\ j \neq i}}^{n_{el}} L_{ij} \tag{2}$$

   are computed using the Lanczos algorithm. Each Lanczos step includes three dot-product operations, one matrix-vector product and an eigenanalysis of the tridiagonal matrix generated by the Lanczos process.
3. After convergence of the Lanczos algorithm, the components of the Fiedler vector are ranked, and this ranking is used to reorder the dual mesh connectivity.
4. The graph is then split in two, and this process is repeated on each subgraph.

The RSB algorithm can be computationally intensive since a series of eigenvalue problems have to be solved. In order to keep the partitioning time as small as possible, we have implemented the RSB algorithm on the CM-5 system in a data-parallel fashion. In this implementation, all elements of the mesh are treated in parallel. It implies a two-level parallelization: one level on the partitions generated at a given stage of the decomposition process and the other on the elements in each partition. Most of the resulting code is written in the CM Fortran language [3], except the eigenanalysis of the tridiagonal matrix which is implemented in CDPEAC (a macro-assembler) [4]. Details of the implementation can be found in [5].

2.2 Node Renumbering

Once the elements have been reordered to obtain element blocks, the mesh nodes are renumbered using the following procedure:

1. Each element is assigned the element block number to which it belongs.
2. Each element sends the block number to the nodes it is associated with. Nodes receiving the same block number from their neighboring elements are marked as "interior nodes" and their location code is the block number received. The other nodes are marked as "boundary nodes" and they choose their location code at random from the block numbers they received.
3. Nodes are ranked based on their location code with the constraint of having interior nodes ranked before boundary nodes for the same location code.
4. Nodes are assigned to the vector units based on their location code in the order obtained at Step 3. Since all nodes may not be assigned during this phase because of the load-balance constraint described at the beginning of Section 2, this strategy forces interior nodes to have a greater probability than boundary nodes of being assigned to the same vector unit as the elements they are associated with.
5. Nodes which have not been assigned during Step 4 are distributed among the vector units which still have room left.

This procedure can be easily implemented in a data-parallel fashion, parallelization occuring over the elements for Steps 1 and 2 and over the nodes for Steps 3 through 5.

2.3 Partitioning Example

Both examples presented in this paper have been run in 64-bit arithmetic on a 32-processing node CM-5E system. This system is a prototype used internally at Thinking Machines and has the following hardware characteristics:

1. A processing node running at 39 MHz and composed of a SuperSPARC microprocessor controlling four vector units and 32 Mbytes of memory (which can be upgraded to 128 Mbytes), yielding a peak 64-bit floating-point performance of 156 Mflops/s/pn. Systems shipped to customers will have processing nodes running at 40 MHz.
2. A new network interface chip which can inject larger packets into the data network at a faster rate. The operating system running on the CM-5E at the time we performed the numerical tests (CMOST 7.3 beta 2.9) had the large packet mode disabled.
3. A SPARCstation 2 control processor. Systems shipped to customer sites will have SPARCstation 10 control processors.

The software and hardware restrictions of this CM-5E prototype should have little impact on the performance of the problems presented in this paper.

A tetrahedral mesh of an assembly part composed of 19,793 nodes and 81,649 elements, courtesy of Mark Shephard (Rensselaer Polytechnic Institute), was used as a partitioning illustration (see Fig.1). The graph representation of this mesh has 152,878 edges. The decomposition into 32 subdomains depicted in Fig.2 shows the quality of partitioning. Note that 128 subdomains are actually needed on a 32-node CM-5E, but the resulting picture is too confusing to be shown here. The total cost of partitioning the mesh into 128 subdomains is 24.6 seconds, making the RSB algorithm a competitive strategy for mesh decomposition. At this level of partitioning, there are 10,648 cuts in the graph, which represents 7.0% of the total number of graph edges. Detailed timings based on CM elapsed times (which correspond to the elapsed execution times while the program is not swapped out by the operating system on the processing nodes) are presented in Tables 1 and 2. Figure 3 presents the cost of the RSB algorithm as the bisection procedure progresses. The $O(\log_2(\text{no. of partitions}))$ cost seen in this figure is due to the two-level parallelization of the RSB algorithm. The time spent renumbering the nodes using the algorithm presented in Section 2.2 is 0.2 seconds.

Table 1. Assembly part. CM elapsed times for different parts of the RSB algorithm for a partitioning into 128 subdomains on a 32-node CM-5E system.

	Timings	Percentage
ident. of connected blocks	5.7 s	23.2%
comp. of Fiedler vector	16.8 s	68.3%
data ranking/reordering	1.3 s	5.3%
miscellaneous	0.8 s	3.2%
Total	24.6 s	100.0%

Table 2. Assembly part. Cost analysis for the computation of the Fiedler vector.

	Timings	Percentage
matrix-vector products	8.2 s	48.8%
dot-products	3.7 s	22.0%
eigenanalyses	1.5 s	8.9%
SAXPYs and miscellaneous	3.4 s	20.3%
Total	16.8 s	100.0%

3 Fluid Dynamics Application

We have implemented in CM Fortran a finite element program for solving the compressible Euler and Navier-Stokes equations [6,7]. It is based on the Galerkin/least-squares formulation proposed by Hughes et al. [8] and Johnson et al. [9]. In the case of steady flow computations, an implicit time-marching scheme is used to reach steady-state. A preconditioned matrix-free GMRES algorithm is employed to solve the nonsymmetric systems of equations arising from the finite element discretization. The gather and scatter operations are performed using special communication procedures available from the Connection Machine Scientific Software Library [10].

To illustrate the performance improvements achieved by proper data mappings, we have computed the inviscid flow around a Falcon Jet flying at Mach 0.85 and at an angle of attack of 1 degree on a 32-node CM-5E system. The mesh, courtesy of Dassault Aviation, has 19,417 nodes and 109,914 tetrahedra (see Fig.4 for a view of the surface mesh on the airplane). A one-point integration rule was used on the elements. A freestream uniform flow was chosen as initial condition, and the solver was marched 50 time steps at a CFL number of 10, which was sufficient to reach steady-state. Two computations were performed successively using the following mapping strategies (referred to as Strategy 1 and Strategy 2, respectively):

1. Random mapping of elements and nodes to the processing nodes.
2. Mapping of elements and nodes according to the procedures described in Section 2.

Timings for the data mapping and the finite element solver (which is a series of gather/compute/scatter cycles) are given in Table 3. This example shows that proper mapping can improve overall speed of the program by more than a factor of two, even if the partitioning time is included in the total time. In the case of Strategy 2, the computation part of the solver achieves 39.5 Mflops/s/pn. The gather and scatter operations yield bandwidths of 24.7 Mbytes/s/pn and 20.0 MBytes/s/pn, respectively. The overall performance of the finite element solver is 1.0 Gflops/s, which is 20% of the peak hardware performance. Substantial computational efficiency can therefore be achieved on distributed-memory computers for finite element applications as long as a careful mapping of the data structures is performed.

Table 3. Falcon Jet. CM elapsed times for different parts of the finite element program run on a 32-node CM-5E system.

	Strategy 1	Strategy 2
data mapping	—	35 s
gather operations	202 s	17 s
computations	180 s	177 s
scatter operations	234 s	26 s
Total time	10 min 16 s	4 min 15 s

References

1. Pothen, A., Simon, H.D., and Liou, K.-P.: Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. **11** (1990) 430–452
2. Simon, H.D.: Partitioning of unstructured problems for parallel processing. Comput. Systems Engrg. **2** (1991) 135–148
3. CM Fortran Language Reference Manual, Version 2.1 (Thinking Machines Corporation, 1994)
4. VU Programmer's Handbook, CMOST Version 7.2 (Thinking Machines Corporation, 1993)
5. Johan, Z., Mathur, K.K., Johnsson, S.L., and Hughes, T.J.R.: An efficient communication strategy for finite element methods on the Connection Machine CM-5 system. Comput. Methods Appl. Mech. Engrg. (in press)
6. Shakib, F., Hughes, T.J.R., and Johan, Z.: A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations. Comput. Methods Appl. Mech. Engrg. **89** (1991) 141–219
7. Johan, Z., Hughes, T.J.R., Mathur, K.K., and Johnsson, S.L.: A data parallel finite element method for computational fluid dynamics on the Connection Machine system. Comput. Methods Appl. Mech. Engrg. **99** (1992) 113–134
8. Hughes, T.J.R., Franca, L.P., and Hulbert, G.M.: A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-diffusive equations. Comput. Methods Appl. Mech. Engrg. **73** (1989) 173–189
9. Johnson, C., Szepessy, A., and Hansbo, P.: On the convergence of shock-capturing streamline diffusion finite element methods for hyperbolic conservation laws. Math. Comp. **54** (1990) 107–129
10. CMSSL for CM Fortran: CM-5 Edition, Version 3.1 (Thinking Machines Corporation, 1993)

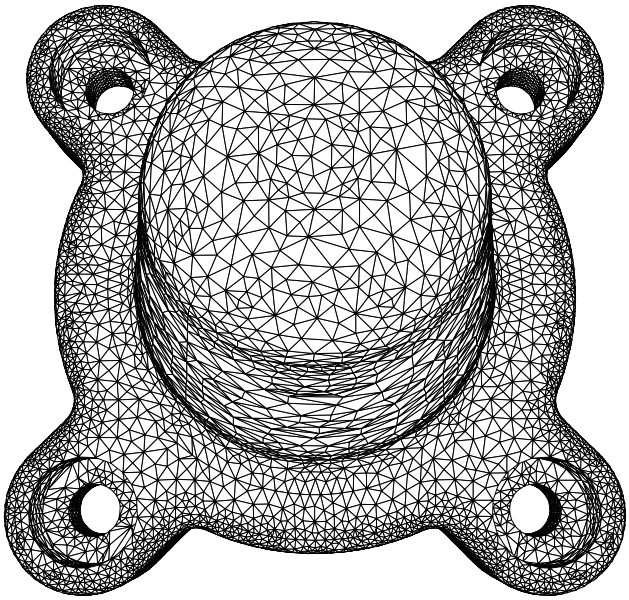

Fig. 1. Assembly part. View of the surface mesh.

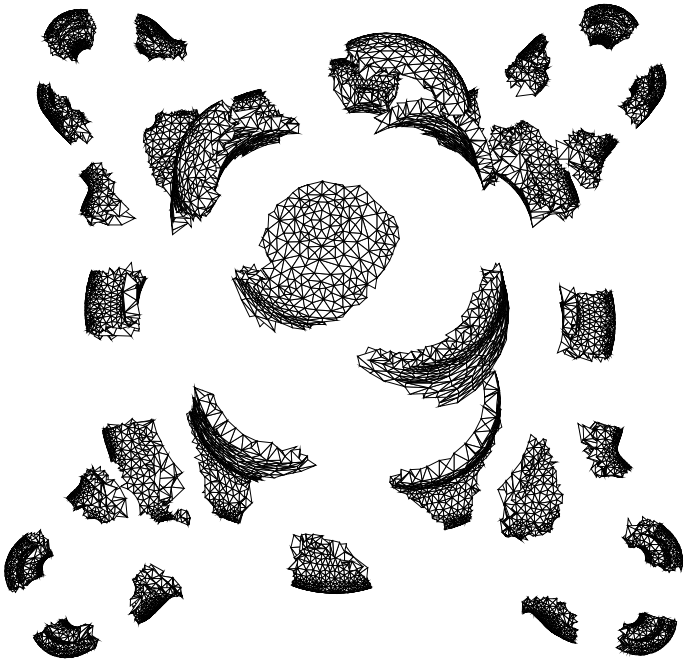

Fig. 2. Assembly part. Decomposition into 32 partitions.

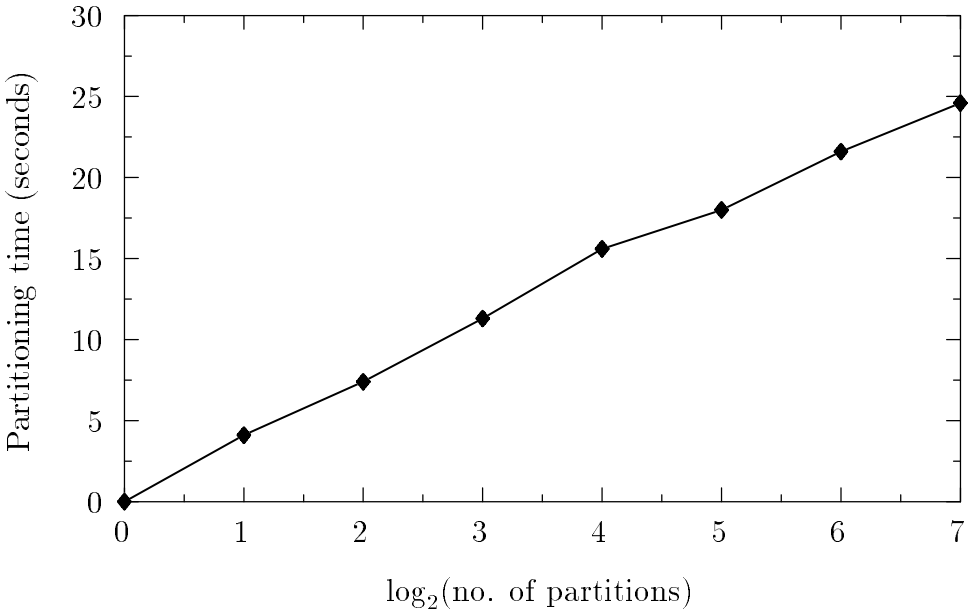

Fig. 3. Assembly part. Partitioning cost as a function of recursive bisection on a 32-node CM-5E.

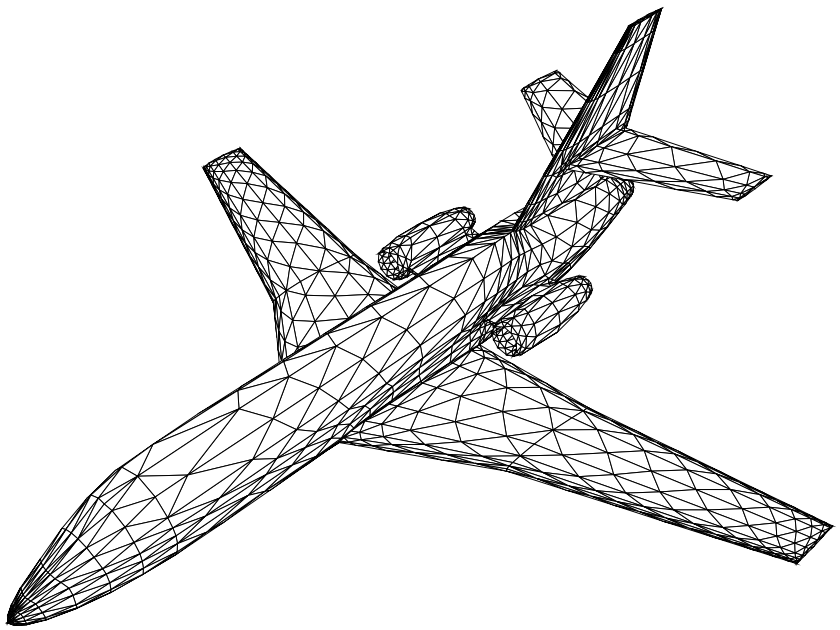

Fig. 4. Falcon Jet. View of surface mesh.