INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 21, 877–884 (1995)

A CASE STUDY IN PARALLEL COMPUTATION: VISCOUS FLOW AROUND AN ONERA M6 WING

ZDENĚK JOHAN

Centric Engineering Systems, Inc., 3393 Octavius Drive, Suite 201, Santa Clara, CA 95054, U.S.A.

KAPIL K. MATHUR

D. E. Shaw & Co., New York, NY 10036, U.S.A.

S. LENNART JOHNSSON

Division of Applied Sciences, Harvard University, 33 Oxford Street, Cambridge, MA 02138, U.S.A.

AND

THOMAS J. R. HUGHES

Division of Applied Mechanics, Stanford University, Durand Building, Stanford, CA 94305, U.S.A.

SUMMARY

We examine the solution of a practical engineering problem on a parallel computer. The problem involves the steady laminar viscous flow about an ONERA M6 wing and the computer is a 64-processing-node Connection Machine CM-5E. We show that efficient domain decomposition procedures lead to a balanced load on the processors and low communication times. The net result is that solutions can be attained in roughly 20 min elapsed time for a 48,011-node, 266,566-element unstructured mesh. We conclude that this is sufficiently fast to support the design process.

KEY WORDS: ONERA M6 wing; parallel computing; viscous flow

1. INTRODUCTION

Parallel computers are capable of significantly reducing computation time on problems of practical engineering interest. One anticipates the reduction in time to be significant enough so that analysis of complex models can be used in the early design process. This is presently not the case, as complex analysis is usually reserved for very late in the design process. In order to bring to fruition the vision of efficient parallel analysis of engineering problems, one needs to realize that computing in a parallel environment involves a number of additional complexities when compared with computing in a sequential environment. Paramount among these are domain decomposition, load balancing and communication costs. Thus, unfortunately, parallel computing is simply more complex than sequential computing. In this paper we consider a case study of the laminar viscous flow about an ONERA M6 wing calculated on a 64-processing-node CM-5E system, each processing node being composed of four vector units. A vector unit will be referred to as a *processor* in the remainder of this paper. The model consists of 48,011 nodes and 266,566 elements. We discuss and analyse the recursive spectral bisection of the model in 256 subdomains and assess its impact on load balancing and communication costs. Despite these additional complexities, we conclude that a very efficient procedure can be

CCC 0271–2091/95/100877–08
Received 16 March 1995

developed enabling a solution to be attained in roughly 20 min elapsed time. This is sufficiently fast to support the preliminary design phase.

2. PARALLEL IMPLEMENTATION ON THE CM-5E SYSTEM

Data distribution is a crucial issue when implementing finite element techniques on distributed memory parallel computers. Communication between processors can become a bottle-neck if the finite element data structures are not carefully mapped to the processors. In order to minimize this bottle-neck, we have developed a set of data-mapping strategies and implemented them on the Connection Machine CM-5E system. Special library communication routines taking advantage of data locality to reduce data transfer between processors are used to perform the gather and scatter operations found in finite element applications.

2.1. Data-mapping strategies

Both elements and nodes of an unstructured mesh are mapped onto the processors of the CM-5E 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 and each block is mapped onto a processor.
2. The mesh nodes are then mapped onto the processors using the mesh partitioning as a criterion for choosing 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-5E data network. In order to achieve the best computational load balance possible in the finite element programme itself, we constrain the elements and the nodes to be uniformly distributed across the processors, i.e. all processors 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-5E system itself.

2.1.1. 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 *et al.* 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. 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. The process ends when there are as many partitions as processors in the CM-5E 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 **L**, defined as

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

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

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-5E 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 Reference 5.

2.1.2. 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 neighbouring 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 processors based on their location code in the order obtained in step 3. Since all nodes may not be assigned during this phase because of the load balance constraint described at the beginning of Section 2.1, this strategy forces interior nodes to have a greater probability than boundary nodes of being assigned to the same processor as the elements they are associated with.
5. Nodes which have not been assigned during step 4 are distributed among the processors which still have room left.

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

2.2. Communication primitives

Gather and scatter communication primitives have been designed to take advantage of the data-mapping strategies presented in the previous subsection, therefore reducing data transfer between processors as much as possible. The gather operation (which transfers data from a node-based data structure to an element-based data structure) is used to illustrate the proposed procedure.

1. The processor holding a given element partition knows which components of the node-based data structure will be needed by the elements of that partition. These values are therefore gathered and stored into a local buffer. One can note that this data transfer is either local (i.e. on processor) if the node-renumbering algorithm has placed the node on the same processor as the partition, or off-processor otherwise.
2. Elements in each partition then gather values from the local buffers in order to perform element-based computations. This is a purely local data transfer.

This two-step procedure has the advantage of eliminating all redundant data transfer that could possibly happen between processors. For example, if two elements residing on the same processor need to gather values from the node residing on another processor, these values are gathered only once during the first step and are then 'spread' to the elements during the second step. The scatter operation is implemented in a similar fashion, the elements first scattering values into local buffers which are in turn scattered to the nodes.

This procedure and the node-renumbering scheme detailed in Section 2.1.2 are the key components of high-bandwidth gather/scatter primitives as shown in the following example.

2.3. *Numerical example*

This example is the computation of a steady viscous flow at Mach 0·5 and Reynolds number 500 (based on the chord length at the wing root) around an ONERA M6 wing placed at an angle of attack of 0°. The tetrahedral mesh, courtesy of Rainald Löhner (The George Washington University), is composed of 48,011 nodes and 266,556 elements. The graph representation of the mesh has 527,966 edges. Figure 1 presents a view of the surface mesh on the outer boundaries of the domain. One can see the high concentration of boundary elements on the plane of symmetry near the root of the wing. The partitioning and fluid flow programmes were compiled with CMF 2.1 and were run in 64 bit arithmetic on a 64-processing-node CM-5E system equipped with 256 vector units. This system was running the Connection Machine operating system CMOST 7.3. All reported timings correspond to *CM elapsed times*.

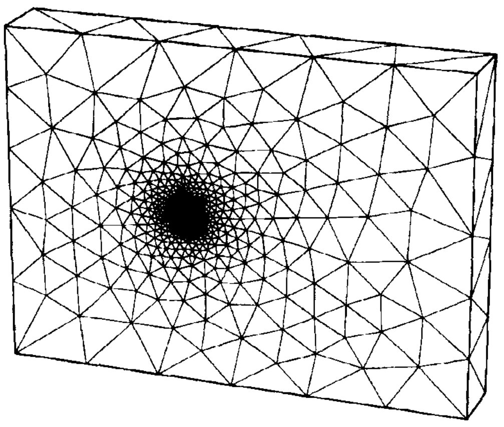
Figure 1. M6 wing: view of surface mesh on outer boundaries

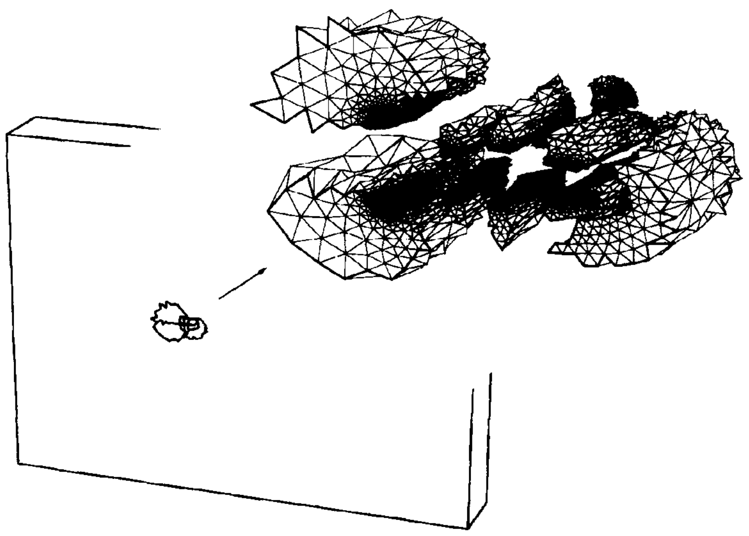

Figure 2. M6 wing: decomposition into 16 subdomains

2.3.1. Mesh decomposition. A decomposition of the mesh into 16 subdomains is depicted in Figure 2. Note that 256 subdomains are actually needed for the CM-5E configuration considered (one subdomain per vector unit). Figure 3 shows the cost of the parallel RSB algorithm as the bisection procedure progresses. The sub-$O(\log_2$ (no. of partitions)) cost is due to the combined effects of the two-level parallelization of the algorithm (see Section 2.1.1) and the decrease in the number of Lanczos iterations as the bisection procedure progresses.

The total cost of partitioning the mesh into 256 subdomains is 61 s. At this level of partitioning there are 57,003 cuts in the graph, representing 10·8 per cent of the total number of graph edges. Table I gives the computing costs of the various parts of the RSB algorithm. The computation of the Fiedler vector using the Lanczos algorithm dominates with almost 80 per cent of the total time. A more detailed cost analysis of the Lanczos algorithm is presented in Table II. One can deduce from these two tables that about 75 per cent of the total time is spent in communication between processors (the communication-dominated portions of the code are the identification of connected blocks, matrix–vector products, and data ranking and reordering). None the less, the parallel RSB algorithm exhibits good performance on the CM- 5E system.

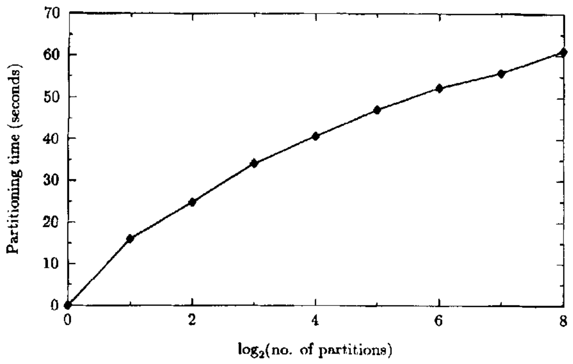

Figure 3. M6 wing: partitioning cost as a function of recursive bisection on 64-node CM-5E system

Table I. M6 wing: elapsed times for various parts of RSB algorithm for partitioning into 256 subdomains on 64-node CM-5E system

	Time (s)	Percentage
Identification of connected blocks	9·3	15·2
Computation of Fiedler vector	47·4	77·6
Data ranking/reordering	2·3	3·8
Miscellaneous	2·1	3·4
Total	61·1	100·0

Table II. M6 wing: cost analysis for computation of Fiedler vector

	Time (s)	Percentage
Matrix–vector products	31·7	66·9
Dot-products	5·5	11·6
Eigenvalue analyses	3·0	6·3
SAXPYs and miscellaneous	7·2	15·2
Total	47·4	100·0

2.3.2. Fluid flow computation. The steady state computation was converged to engineering accuracy (three orders of magnitude in residual reduction) in 500 time steps at CFL number 2. A one-point integration rule was used on each element. Views of the wing surface mesh and pressure contours on the wing are shown in Figures 4 and 5 respectively. Timings for the computation and communication (i.e. gather and scatter) parts of the programme are given in Table III. In this example the computation part achieves 36·8 Mflops/s/pn. The gather and scatter operations yield bandwidths of 20·5 and 20·3 Mbytes/s/pn respectively. The overall performance of the solver is 1·9 Gflops/s, which is about 20 per cent of the peak hardware performance. The convergences of the drag force F_D and side force F_S as a function of the time step number are presented in Figures 6 and 7 respectively. One can see that convergence (as far as the aerodynamicist is concerned) is actually achieved after about 350 time steps. This viscous computation could therefore have been done in less than 20 min.

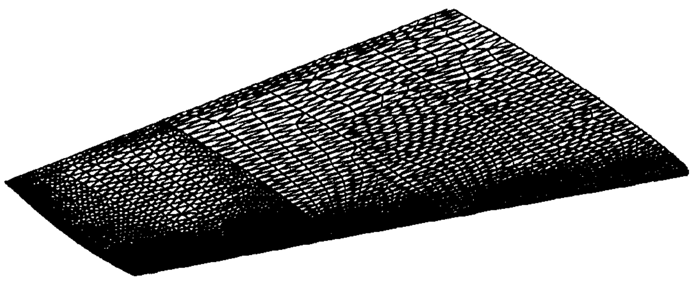

Figure 4. M6 wing: view of mesh on wing surface

Table III. M6 wing: CM elapsed times for various parts of finite element programme run on 64-node CM-5E system

	Time (s)
Gather operations	83
Computations	1104
Scatter operations	147
Total	1334*

* 22 min, 14 s.

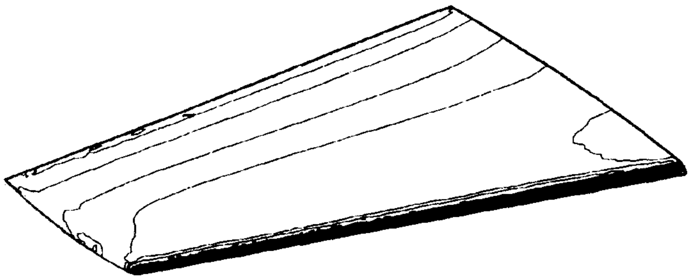

Figure 5. M6 wing: pressure contours on wing surface

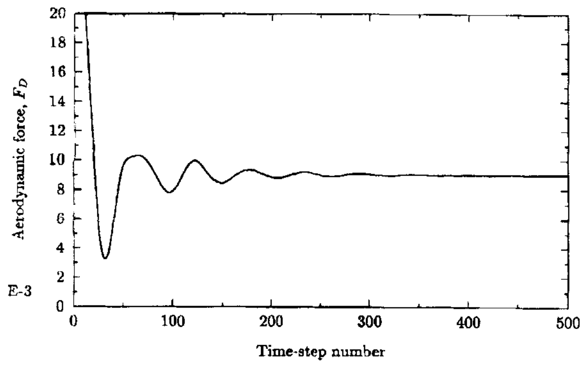

Figure 6. M6 wing: convergence of drag force

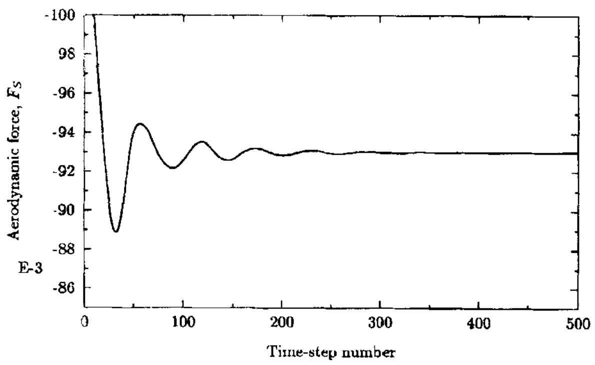

Figure 7. M6 wing: convergence of side force

3. CONCLUSIONS

Practical solution of engineering problems on parallel computers involves consideration of aspects not present in sequential computing. For example, one needs to address issues of domain decomposition, load balancing and communication costs in addition to efficiently programmed algorithms. As a practical example of these issues we have considered the solution of a laminar viscous flow about an ONERA M6 wing on a 64-processing-node CM-5E system. We have shown that a parallel implementation of the recursive spectral bisection algorithm leads to a very efficient domain decomposition of high quality. This in turn leads to very low communication costs. Thus a steady viscous flow of a model comprised of 48,011 nodes and 266,566 elements can be obtained on a 64-processing-node CM-5E system in roughly 20 min elapsed time. This is sufficiently fast to support the engineering design process.

REFERENCES

1. A. Pothen, H. D. Simon and K.-P. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl.*, **11**, 430–452 (1990).
2. H. D. Simon, 'Partitioning of unstructured problems for parallel processing', *Comput. Syst. Eng.*, **2**, 135–148 (1991).
3. *CM Fortran Language Reference Manual, Version 2.1*, Thinking Machines Corporation, Cambridge, MA, 1994.
4. *VU Programmer's Handbook, CMOST 7.2*, Thinking Machines Corporation, Cambridge, MA, 1993.
5. Z. Johan, K. K. Mathur, S. L. Johnsson and T. J. R. Hughes, 'An efficient communication strategy for finite element methods on the Connection Machine CM-5E system', *Comput. Methods Appl. Mech. Eng.*, **113**, 363–387 (1994).