Computer Assisted Mechanics and Engineering Sciences, 5: 173–183, 1998.

Parallel iterative computational methods for 3D finite element flow simulations

Vinay Kalro and Tayfun Tezduyar
Aerospace Engineering and Mechanics
Army HPC Research Center
University of Minnesota
1100 Washington Ave, Minneapolis, MN 55415, USA.

(Received January 24, 1997)

In this paper we discuss sparse matrix computational methods, and their parallel implementations, for evaluating matrix-vector products in iterative solution of coupled, nonlinear equations encountered in finite element flow simulations. Based on sparse computation schemes, we introduce globally-defined preconditioners by mixing clustered element-by-element preconditioning concept with incomplete factorization methods. These preconditioners are implemented on a CRAY T3D parallel supercomputer. In addition to being tested in a number of benchmarking studies, the sparse schemes discussed here are applied to 3D simulation of incompressible flow past a circular cylinder.

1. INTRODUCTION

Distributed-memory, massively parallel supercomputers have become major players in large-scale flow simulations. Finite element flow computations are well suited to parallel architectures since they involve similar operations applied to a large number of elements.

Earlier examples of Single-Instruction-Multiple-Data (SIMD) implementations within a data parallel paradigm on the Thinking Machines CM-200 and CM-5 can be found in Behr et al [1]. Flow simulations using a Multiple-Instruction-Multiple-Data (MIMD) implementation within a message-passing model on the CRAY T3D were reported in Tezduyar et al [2].

Krylov space based iterative update techniques are now widely used for solution of the equation systems arising from finite element discretizations. Here, the global coefficient matrix of such an equation system is needed solely for matrix-vector product evaluations. Implementations involving matrix-vector products at the element level, followed by global assembly, are simple to parallelize but memory intensive. Furthermore, these implementations involve superfluous operations and storage, because a typical node is shared by multiple elements. Matrix-free implementations [3] alleviate the memory crunch, but at the cost of repetitive operations and increased execution time.

In the sparse computation schemes discussed here, the global matrix is stored using a node-based data structure. Both memory needed and operations performed in matrix-vector product evaluations are reduced (compared to element schemes) by a factor of ~ 2 for hexahedral meshes and ~ 7 for tetrahedral meshes.

We also explore in this paper sparse schemes for globally-defined advanced preconditioners. In virtually all of our earlier incompressible flow computations on parallel platforms we used diagonal preconditioners. These have lower overheads, but in some cases an exhaustive number of iterations is required to achieve convergence. A number of advanced element-by-element (EBE) and clustered element-by-element (CEBE) preconditioners for flow computations have been proposed earlier [4, 5, 6]. Efficient parallel implementation of these preconditioners remain as a challenging research area. Here we introduce a preconditioning method based on mixing CEBE preconditioning concept with incomplete factorizations with zero fill in (ILU0) [7]. We call this preconditioner CEBE-ILU0.

We also assess the performance of a variation of CEBE-ILU0 which we call L-ILU0. The L-ILU0 is a local (cluster level) preconditioner where all the off-diagonal terms for mesh nodes lying on cluster boundaries are set to zero. In our implementation, a single cluster of elements is assigned to each processor. We will also refer to these clusters as "partitions".

In Section 2, the Compressed Sparse Row (CSR) data structure [7] for sparse storage is reviewed, and its modification to accommodate the nodal-block structure of the matrices arising from finite element formulations is explained. The CEBE-ILU0 preconditioner is described in Section 3. Parallel implementations using a message-passing paradigm on the CRAY T3D are discussed in Section 4. Scalability and performance of the computations based on sparse schemes with diagonal preconditioners are demonstrated in Section 5. Performance of the CEBE-ILU0 preconditioner and its effectiveness compared to diagonal preconditioning is assessed in Section 6. In Section 7, we describe application of sparse schemes with diagonal preconditioner to 3D simulation of time-dependent incompressible flow around a circular cylinder. Concluding remarks are presented in Section 8.

2. DATA STRUCTURES FOR SPARSE SCHEMES

In our computations, the CSR data structure is used together with the nodal-block structure. The CSR data structure consists of an array $\mathbf{A}\,(nz)$ which contains the non-zero entries, an array $\mathbf{ja}\,(nz)$ which contains the column indices of the non-zero entries, and $\mathbf{ia}\,(n+1)$ whose entries point to the beginning of the rows in $\mathbf{A}$ and $\mathbf{ja}$. Here nz is the total number non-zero entries, and n is the number of equations.

For finite element meshes, the patterns for $\mathbf{ia}$ and $\mathbf{ja}$ (and hence for $\mathbf{A}$) is deduced from the nodal connectivity. When there are ndf unknowns per node, we can exploit the nodal-block structure of the matrix and modify $\mathbf{A}\,(nz)$ to $\mathbf{A}\,(ndf, nz, ndf)$. The arrays $\mathbf{ia}$ and $\mathbf{ja}$ described above are now used to traverse the second axis of $\mathbf{A}$.

To avoid repetitive computation of locations in the global matrix during the assembly operations, the map from element stiffness matrix to global (sparse) matrix is computed and stored in $\mathbf{e2g}\,(nen, nen, ne)$. Here nen is the number of nodes per element and ne is the number of elements in the finite element mesh.

3. CEBE-ILU0 PRECONDITIONER

The CEBE preconditioner, developed by Liou and Tezduyar [6], was successfully used, with the GMRES update technique [8], for both incompressible and compressible flow computations. Here the mesh is decomposed into clusters of elements. The element-level matrices are assembled to cluster-level matrices, and the CEBE preconditioners are formed as sequential products of these cluster-level matrices. The CEBE scheme is rather memory intensive, owing to the fact that the cluster-level matrix storage is not sparse (since complete factorizations are required). Furthermore, it was shown [6] that the larger the cluster size, the more effective the preconditioner.

Bearing the above factors in mind, we propose a preconditioning method utilizing the CEBE concept. Here, a single (significantly large) cluster resides on each processor. No additional data structures and communications are required for the formation of the preconditioner. Instead of complete factorizations of the cluster-level matrices, we use only incomplete factorizations, namely ILU0. The CEBE-ILU0 preconditioner is outlined below.

Consider the system:

$$\mathbf{Ax} = \mathbf{b}. \tag{1}$$

Such a system needs to be solved at each step of the Newton–Raphson method used for iteratively solving a coupled, nonlinear equation system. The nonlinear equation system is formed at every time step or pseudo-time step.

In scaled form, the system can be written as:

$$\tilde{\mathbf{A}}\mathbf{x} = \tilde{\mathbf{b}}, \tag{2}$$

where:

$$\tilde{\mathbf{A}} = \mathbf{W}^{-\frac{1}{2}}\mathbf{A}\mathbf{W}^{-\frac{1}{2}}, \tag{3}$$

$$\tilde{\mathbf{x}} = \mathbf{W}^{\frac{1}{2}}\mathbf{x}, \tag{4}$$

$$\tilde{\mathbf{b}} = \mathbf{W}^{-\frac{1}{2}}\mathbf{b}, \tag{5}$$

and

$$\mathbf{W} = \mathrm{diag}\,(\mathbf{A})\,. \tag{6}$$

The finite element mesh is partitioned into N_p contiguous clusters (partitions) of ne_p elements. Each partition is assigned to a processing node. The global matrix $\tilde{\mathbf{A}}$ is expressed as an assembly of partition matrices:

$$\tilde{\mathbf{A}} = \sum_{p=1}^{N_p} \tilde{\mathbf{A}}_p, \tag{7}$$

where:

$$\tilde{\mathbf{A}}_p = \sum_{e=1}^{ne_p} \tilde{\mathbf{A}}^e. \tag{8}$$

Consider the incomplete factorization of the sparse partition matrix:

$$\left(\mathbf{I} + \tilde{\mathbf{A}}_p - \tilde{\mathbf{W}}_p\right) \doteq \mathbf{L}_p^0\mathbf{U}_p^0, \tag{9}$$

where $\mathbf{L}_p^0$ and $\mathbf{U}_p^0$ are the lower and upper incomplete factors respectively.

The preconditioner is then written as:

$$\tilde{\mathbf{P}} = \prod_{p=1}^{N_p} \mathbf{L}_p^0\mathbf{U}_p^0. \tag{10}$$

and the preconditioned system is written as:

$$\tilde{\mathbf{P}}^{-1}\tilde{\mathbf{A}}\mathbf{x} = \tilde{\mathbf{P}}^{-1}\tilde{\mathbf{b}}, \tag{11}$$

Remarks:

1. In the implementations, $\tilde{\mathbf{P}}^{-1}$ is not formed explicitly, instead its action is effected through the incomplete factors.

2. Our experience with one-sided and two-sided scaling indicates that both perform comparably, hence one-sided scaling, which needs fewer operations, is normally the method of our choice.

3. In the L-ILU0 preconditioner, all the off-diagonal entries of partition boundaries are set to zero. Diagonal preconditioning is used for these nodes, while ILU0 preconditioning is used for interior nodes.

4. PARALLEL IMPLEMENTATIONS ON THE CRAY T3D

An efficient parallel implementation of a finite element solution technique should incorporate the features listed below.

- Arrangement of data across processors in a fashion which maximizes data locality and maintains load balance.
- Data structures and communication strategies which maintain a favorable ratio of on-processor to off-processor operations.
- Scalable performance with increasing machine size.

In a distributed memory implementation, three levels of data structures are required at each of the N_p processors.

- Element-level data structures. The finite element mesh is divided into N_p contiguous, evenly-sized partitions of elements. Each partition is assigned to a unique processor. The goal of partitioning is to obtain minimal number of boundary nodes (i.e. the nodes shared across mesh partitions) since these nodes are sources off-processor communication. The nodes comprising a partition are reordered locally on each processor. The interior nodes are numbered ahead of the boundary nodes. The element connectivity for the partition is based on this reordering.
- Partition-level data structures. The partition-level data structures store information for the nodes comprising the partition mesh. The CSR data structures described in Section 2 are setup (locally) for each partition. Partition vectors and sparse matrices can be assembled independently on each processor, free from any off-processor interaction.
- Global-level data structures. These are required for the solution of the global equation system. Each processor is associated with a set of global nodes. A majority of these nodes are the interior nodes of the partition corresponding to that processor. This ensures favorable ratio of on-processor to off-processor operations.

Communication of data between the element and global levels occurs in the two stages listed below.

- Global-level $\rightleftharpoons$ Partition-level. These involve off-processor communications. As a preprocessing step, the communication routes are computed based on the assignment of the global nodes. Off-processor communication is achieved using the *PVM* [9] message-passing library.
- Partition-level $\rightleftharpoons$ Element-level. These involve only on-processor operations. Note that no information is stored at the element level. When element-level data is required for element-based computations, it is obtained from partition-level vectors using local connectivity.

The transfer of data from the global-level to the partition-level (and subsequently to the element-level) is called a *gather* operation, and the reverse transfer a *scatter* operation. Communication is also required for computing the inner products of global vectors. These are called *reduction* operations. For the *reduction* operations, processors are arranged in a tree structure. Each processor computes its component of the global inner product; these are then communicated through the levels of the tree and summed. This requires $\mathbf{O}(\log_2(N_p))$ communication steps.

Computation of global matrix-vector products comprises gathering the partition components from the global vectors, followed by sparse partition-level matrix-vector multiplications, and scattering back to global vectors.

The preconditioning phase is implemented as follows. As explained earlier, the preconditioning matrix is formed as a series product of partition matrices. For all the interior nodes, multiplication with the corresponding partition component of the preconditioning matrix is executed in parallel.

However, the boundary nodes have to be dealt with in a sequential fashion. For this purpose, the processors are colored such that no two processors in the same color group share nodes.

The Greedy Algorithm [7] is used to color the processors (clusters) as described below:

- Establish the connectivity between clusters, where clusters are declared to be neighbors if they share nodes.
- Apply Greedy Algorithm

1. Initialize:
   do i = 1,....N_p
   Color(i) = 0;
   end do
   ncol = 1;
2. Color Clusters:
   do i = 1,2....,N_p
   Search all the neighboring clusters and determine the first color (mincolor) not present;
   If there is no such color then
   $ncol = ncol + 1$;
   $Color(i) = ncol$;
   else
   $Color(i) = mincolor$;
   end if
   end do

Information on the boundary nodes shared by processors of different color is processed, and a communication trace similar to the gather setup is stored. An ILU0 factorization is performed in parallel for each partition component of the preconditioning matrix. The forward and backward solves are thus implemented as follows:

1. Parallel forward solve for all interior nodes on the partitions.
2. Sequential loop over colors; for each color:
   - Forward solve followed by a backward solve for all boundary nodes of partitions with the current color.
   - Update the boundary nodes on unprocessed colors.
3. Parallel backward solve for all interior nodes on the partition.

Remark:

1. With the L-ILU0 preconditioner, the sequential solves and corresponding communications are totally avoided.

5. SCALABILITY AND PERFORMANCE OF THE SPARSE SCHEMES WITH DIAGONAL PRECONDITIONERS

In this section, we evaluate the scalability and performance of the sparse schemes with diagonal preconditioners. These schemes are used for stabilized, finite element formulation [10] of incompressible flow simulations. The timings are based on a single step of a Newton–Raphson iteration sequence. All the stages involved in that Newton–Raphson step are accounted for in the timings. These include element-based computations, assembly of partition-level matrices and vectors, formation of the global residual vector, and solution of the resulting linear equation system with the

GMRES update technique. One point quadrature is used for tetrahedral meshes and eight points for hexahedral meshes. Four different meshes are used for the evaluations. Table 1 gives a brief description of each of these meshes which are drawn from typical finite element flow simulations. The Krylov space size in the GMRES update is set to 20. The timings (measured in seconds) are split into the components listed below.

- COMM: Time for the communication-based components.
- FORM: Time for the formation of the equation system.
- MV: Time for partition-level matrix-vector products.
- COMP: Total time for all the computation-based components
- TOTAL: Overall time (COMP + COMM)

Table 1. Description of the meshes used for performance evaluations of sparse schemes with diagonal preconditioners.

MESH	GEOMETRY	TYPE	NODES	ELEMENTS
1	AIRCRAFT	TET	192,595	1,110,046
2	AUTOMOBILE	TET	448,695	2,815,518
3	SPHERE	HEX	509,432	495,000
4	PARACHUTE	HEX	1,114,241	1,085,016

From Table 2 we observe that the computation-intensive components (FORM and MV) exhibit a nearly linear speed-up with the machine size. For a given mesh, communication costs increase with machine size due to the increase in the number of boundary nodes. The proportion of computation is higher for hexahedral meshes due to larger number of quadrature points. Provided the mesh is sufficiently large to load the processor, close to linear speed-up is obtainable for overall performance. The memory requirements are ~ 4.5 Kbytes/node for tetrahedral meshes, and ~ 6.5 Kbytes/node for hexahedral meshes.

Table 2. Timings (measured in seconds) for the performance evaluations of sparse schemes with diagonal preconditioners.

MESH	PN	COMM	FORM	MV	COMP	TOTAL	COMM(%)
1	32	2.10	10.72	3.96	15.96	18.06	11.07
1	64	2.14	5.36	2.66	8.02	10.16	21.06
2	64	3.48	13.64	5.15	20.27	23.75	14.65
2	128	3.84	6.84	2.92	10.52	14.36	26.74
3	64	3.56	37.39	8.60	47.70	51.26	6.95
3	128	4.00	18.73	4.46	24.04	28.04	14.26
4	128	5.87	41.02	10.68	53.60	59.47	9.87
4	256	6.98	21.54	5.47	27.97	34.95	19.97

6. PERFORMANCE OF THE CEBE-ILU0 PRECONDITIONER

In this section, the performance of the CEBE-ILU0 preconditioner is evaluated and compared with diagonal preconditioning.

Three test problems which are listed below are considered.

- Flow around a SPHERE at $Re = 400$.
- Flow around a CYLINDER at $Re = 300$.
- Flow around a ram-air PARACHUTE at $Re = 10^7$.

The problem descriptions are given in Table 3.

Table 3. Description of the problems used for evaluating the performance the CEBE-ILU0 preconditioners.

MESH	GEOMETRY	TYPE	NODES	ELEMENTS	EQNS	N_p	NCOLOR
1	SPHERE	TET	43,282	258,569	162,096	32	7
2	CYLINDER	HEX	197,948	186,240	760,017	64	8
3	PARACHUTE	HEX	469,493	455,520	1,833,216	128	9

The normalized timings (seconds/GMRES iteration) and percentage cost of various components of the solution step are shown in Tables 4 and 5. These correspond to a Krylov space size of 50.

Table 4. Normalized timings (seconds/GMRES iteration) for the solution step with the CEBE-ILU0 preconditioner.

CASE	MV	GS	REDN	HESS	PCBE	SCBE	CCBE	TCBE
SPH	.0493	.0251	.0141	.0075	.0459	.1388	.0764	.2611
CYL	.1885	.0636	.0777	.0686	.1515	.3981	.1810	.7306
PARA	.2027	.1390	.0932	.0838	.1829	.6042	.4303	1.2174

Table 5. Percentage costs per iteration for the solution step with the CEBE-ILU0 preconditioner.

CASE	MV	GS	REDN	HESS	PCBE	SCBE	CCBE	TCBE
SPH	13.80	7.02	3.94	2.10	12.85	38.86	21.39	73.12
CYL	16.69	5.63	6.88	6.07	13.42	35.26	16.03	64.72
PARA	11.67	8.00	5.37	4.83	10.53	34.80	24.78	70.12

We define below the symbols used in these tables which were not defined previously in Section 5.

- GS: Gather and scatter operations.
- REDN: Reduction operation.
- HESS: Formation and solution of reduced Hessenberg system.
- PCBE: Parallel forward and backward solves in CEBE-ILU0 preconditioning.
- SCBE: Sequential forward and backward solves in CEBE-ILU0 preconditioning.

- CCBE: Communications involved in CEBE-ILU0 preconditioning.
- TCBE: Total time for CEBE-ILU0 preconditioning.

The times for MV, GS, and TCBE vary linearly with the Krylov space size (ikg), while the times for REDN and HESS vary quadratically with ikg. Our observations from Tables 4 and 5 are given below.

1. Approximately 60-70% of the time is spent in the preconditioning phase. This means CEBE-ILU0 preconditioning takes 2.5–3.5 times more time than diagonal preconditioning.
2. A significant amount of time is spent in the sequential solve phase ($\sim$ 50-60% for SCBE + CCBE). This time is proportional to the number of colors. If $ncol$ is the number of colors used to color the processors, each processor will be active for only $1/ncol$ of the time during sequential solves. A more efficient scheme would entail allowing more than one cluster per processor. Each processor would then hold clusters belonging to every color and would be active throughout the preconditioning phase.
3. Figures 1–3 show the performance of CEBE-ILU0, L-ILU0 and diagonal preconditioners as functions of CPU time and number of iterations. For the test cases presented, the diagonal preconditioner works out to be most economical for a convergence level of 2 orders of magnitude, followed by L-ILU0 for a convergence level of 3 orders of magnitude. However, the CEBE-ILU0 preconditioner exhibits superior performance in terms of overall convergence.

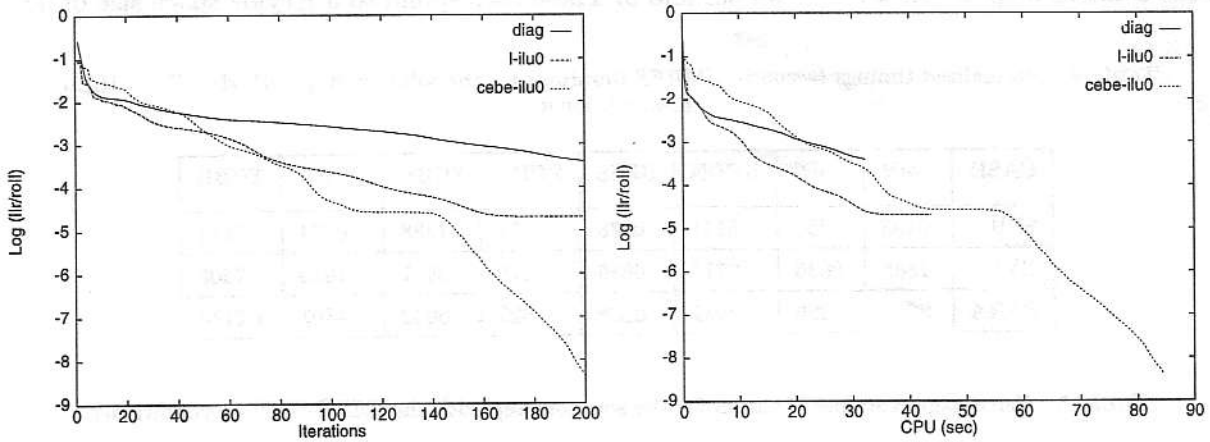

Fig. 1. Convergence of CEBE-ILU0 and L-ILU0 preconditioners for 3D flow past a sphere at $Re = 400$.

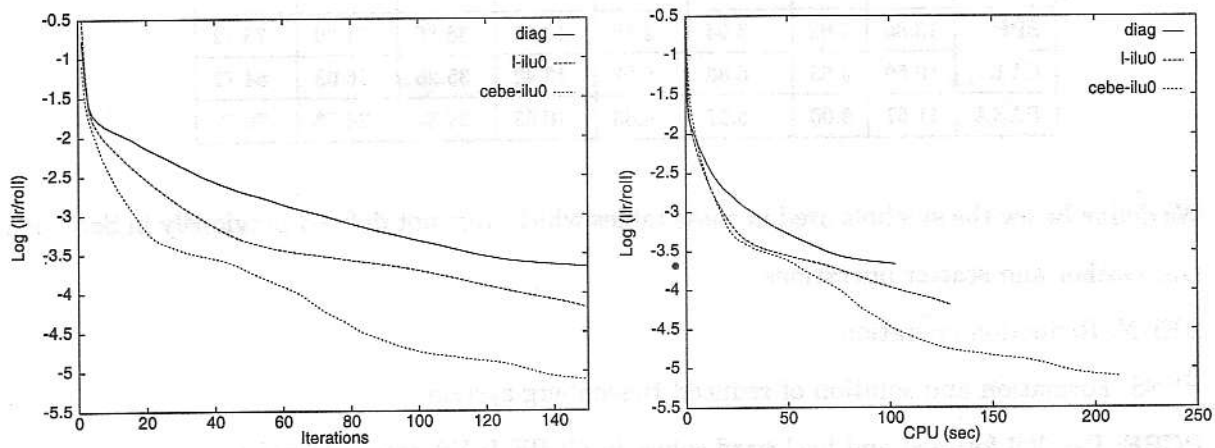

Fig. 2. Convergence of CEBE-ILU0 and L-ILU0 preconditioners for 3D flow past a cylinder at $Re = 300$.

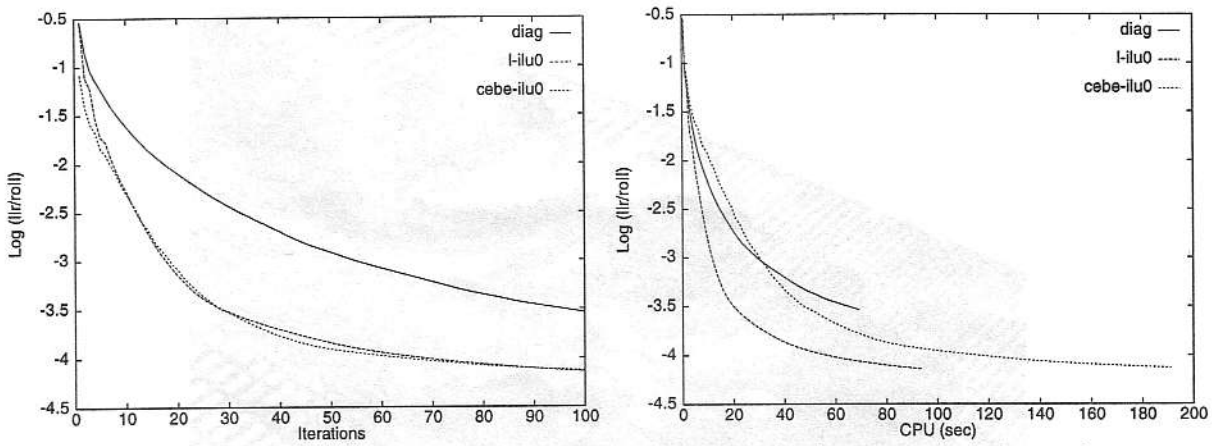

Fig. 3. Convergence of CEBE-ILU0 and L-ILU0 preconditioners for flow past a ram-air parachute at $Re = 10^7$.

7. Example application: 3D time-dependent flow past a circular cylinder

In this section, results from the 3D simulation of time-dependent incompressible flow around cylinders at Reynolds numbers 300 and 800 are presented. Earlier results with matrix-free computations were reported in Kalro and Tezduyar [11]. A hexahedral mesh with 197,948 nodes and 186,240 elements is used. This simulation requires solution of 760,107 coupled, nonlinear equations at every time step. Diagonal preconditioning is used in the iterations with GMRES update technique. These computations take 33.15 sec/time step on a 64-processor CRAY T3D. Visualization of the isosurfaces of vorticity clearly indicates 3D effects in the form of spanwise waves along the axis of the cylinder. These structures are regularly spaced at $Re = 300$ (see Fig. 4). At $Re = 800$ (see Fig. 5) there is a breakdown in structure and the wake is turbulent.

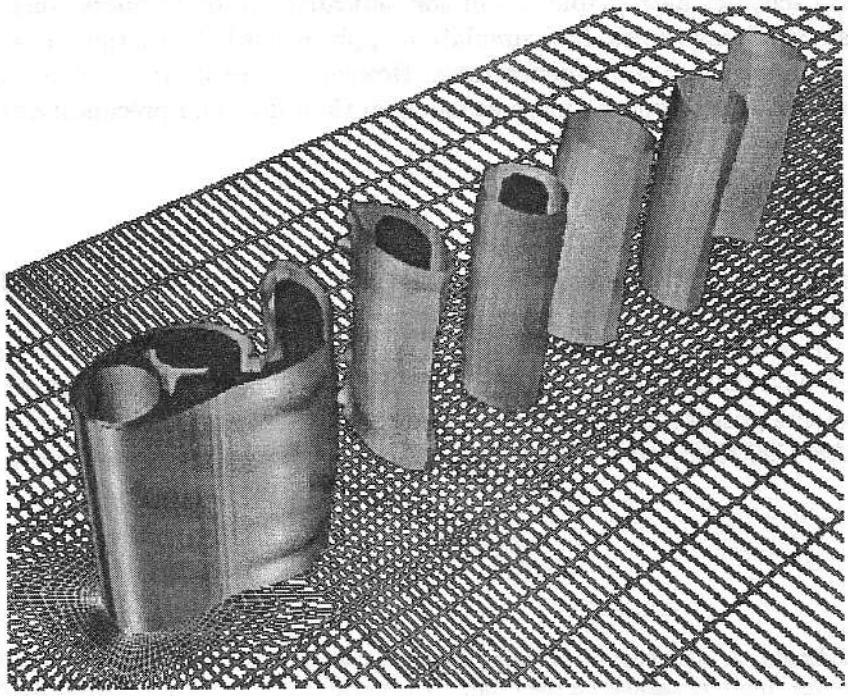

Fig. 4. 3D flow past a cylinder at $Re = 300$

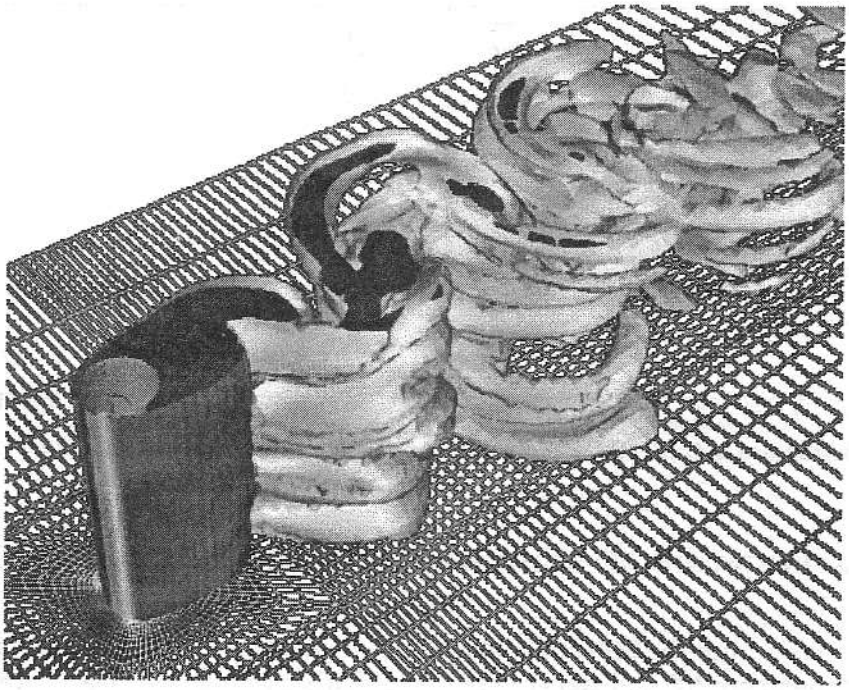

Fig. 5. 3D flow past a cylinder at $Re = 800$

8. Concluding remarks

In this paper we presented parallel methods for computing matrix-vector products using a sparse storage scheme. The conventional CSR storage format was modified to utilize the nodal-block structure of matrices arising from finite element discretizations. The resulting sparse-scheme-based flow solver shows good scalability and enables carrying out large-scale simulations in short turn around times. The sparse schemes also enable use of sophisticated preconditioners, such as CEBE-ILU0 and L-ILU0. Tests involving typical flow simulation applications indicate that these preconditioners are capable of yielding better convergence rates. However, depending upon the convergence level desired, they may not necessarily be more economical than diagonal preconditioning.

Acknowledgment

This research was sponsored by ARO under grant DAAH04-93-G-0514, by ARPA under NIST contract 60NANB2D1272, and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. CRAY C90 time was provided in part by the University of Minnesota Supercomputer Institute.

References

[1] M. Behr, A. Johnson, J. Kennedy, S. Mittal, T.E. Tezduyar. Computation of incompressible flows with implicit finite element implementations on the Connection Machine. *Computer Methods in Applied Mechanics and Engineering*, **108**: 99–118, 1993.

[2] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, C. Waters. 3D simulation of flow problems with parallel finite element computations on the CRAY T3D. In: *Computational Mechanics '95, Proceedings of International Conference on Computational Engineering Science*, Mauna Lani, Hawaii, 1995.

[3] Z. Johan. *Data Parallel Finite Element Techniques for Large-Scale Computational Fluid Dynamics*. Ph.D. thesis, Department of Mechanical Engineering, Stanford University, 1992.
[4] T.J.R. Hughes, J. Winget, I. Levit, T.E. Tezduyar. New alternating direction procedures in finite element analysis based upon EBE approximate factorizations. In: S. Atluri, N. Perrone, eds., *Recent Developments in Computer Methods for Nonlinear Solid and Structural Mechanics*, 75–109, ASME, 1983.
[5] T.E. Tezduyar, J. Liou. Grouped element-by-element iteration schemes incompressible flow computations. *Computer Physics Communications*, **53**: 441–453, 1989.
[6] J. Liou, T.E. Tezduyar. A clustered element-by-element iteration method for finite element computations. In: R. Glowinski et al., eds., *Domain Decomposition Methods for Partial Differential Equations*, Chapter 13, 140–150, SIAM, 1991.
[7] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, 1996.
[8] Y. Saad, M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, **7**: 856–869, 1986.
[9] A. Geist et al. *PVM: Parallel Virtual Machine, A User's Guide for Networked Parallel Computing*. MIT Press, Cambridge, MA, 1994.
[10] T.E. Tezduyar. Stabilized finite element formulations for incompressible flow computations. *Advances in Applied Mechanics*, **28**: 1–44, 1991. ← 1992
[11] V. Kalro, T. Tezduyar. Parallel finite element computation of 3D incompressible flows on MPPs. In: W. G. Habashi, ed., *Solution Techniques For Large-Scale CFD Problems*. Computational Methods in Applied Sciences, Wiley, 1995.