AIAA JOURNAL
Vol. 32, No. 10, October 1994

Parallel Implicit Unstructured Grid Euler Solvers

V. Venkatakrishnan*
Institute for Computer Applications in Science and Engineering, Hampton, Virginia 23681

A mesh-vertex finite volume scheme for solving the Euler equations on triangular unstructured meshes is implemented on a multiple-instruction/multiple-data stream parallel computer. An explicit four-stage Runge-Kutta scheme is used to solve two-dimensional flow problems. A family of implicit schemes is also developed to solve these problems, where the linear system that arises at each time step is solved by a preconditioned GMRES algorithm. Two partitioning strategies are employed: one that partitions triangles and the other that partitions vertices. The choice of the preconditioner in a distributed memory setting is discussed. All of the methods are compared both in terms of elapsed times and convergence rates. It is shown that the implicit schemes offer adequate parallelism at the expense of minimal sequential overhead. The use of a global coarse grid to further minimize this overhead is also investigated. The schemes are implemented on a distributed memory parallel computer, the Intel iPSC/860.

Introduction

TRIANGULAR meshes have become popular in computational fluid dynamics. They are ideal for handling complex geometries and for adapting to flow features, such as shocks and boundary layers. Considerable attention has been focused on improving the spatial operator, which has evolved to a very high degree of sophistication for the Euler and Navier-Stokes equations.[1,2] Distributed-memory parallel computers seem to offer an avenue for doing large problems very fast due to their scalability. For the goal of sustained high-performance on these machines to be realized, many fundamental issues need to be addressed. Among these are scalable algorithms and software.

Explicit schemes used in computational fluid dynamics are completely parallel. They only require a simple update procedure that involves local dependencies. On a parallel computer, such schemes typically only require communication to nearest neighbors. Implicit schemes, on the other hand, require the solution of coupled equations that involves global dependencies. However, there are some applications, such as certain classes of unsteady flows, where explicit schemes are quite useful. When steady-state solutions are sought, explicit schemes typically require thousands of time steps to converge and exhibit very slow convergence rates. Implicit schemes allow larger time steps to be taken and usually result in better convergence rates. Implicit schemes have to be designed carefully since the work involved at each time step could be substantial.

Shakib et al.,[3] Whitaker et al.,[4] and Whitaker[5] have used the generalized minimum residual technique (GMRES) of Saad and Schultz[7] with diagonal preconditioning to solve the linear system of equations arising from an implicit discretization of the compressible Navier-Stokes equations on unstructured grids. Venkatakrishnan and Mavriplis[6] tested a family of implicit schemes on the Cray Y-MP for solving the two-dimensional compressible Navier-Stokes equations on unstructured meshes. They concluded that GMRES with incomplete lower-upper (ILU) preconditioning (GMRES/ILU) was superior to other implicit schemes over a whole range of flow conditions and was as efficient as the unstructured multigrid strategy of Mavriplis and Jameson.[2]

On distributed memory parallel computers, the design of implicit schemes is more difficult since parallelism and load balance during the implicit phase are additional considerations. Johan et al.[8] have implemented an implicit iterative solution strategy based on the diagonal-preconditioned matrix-free GMRES algorithm on the Connection Machine CM-2. Venkatakrishnan et al.[9] and Das et al.[10] have also shown that it is possible to obtain supercomputer performance when solving explicit unstructured grid problems on the Intel iPSC/860. By paying careful attention to the partitioning of the mesh, communication schedule, and data structures, they have been able to show that two to three times the speed of a Cray Y-MP/1 could be obtained with 128 processors of the Intel iPSC/860. The effects of using various strategies for partitioning the unstructured grids on communication costs have been examined as well.

In this paper, the implicit algorithms developed in Ref. 6 are explored as candidate schemes on a distributed-memory parallel computer. Two different ways of partitioning the unstructured mesh across processors are explored. The issues in implementing the GMRES algorithm and the preconditioners in parallel are addressed. Results for a typical flow around a multielement airfoil are presented, and the performances of the explicit and implicit schemes on the iPSC/860 are compared using the two different partitioning strategies. Finally, the use of a global coarse grid to improve convergence is investigated with the best implicit scheme.

Governing Equations and Spatial Discretization

The Euler equations in integral form for a control volume Ω with boundary $\partial\Omega$ read

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} \boldsymbol{u}\,\mathrm{d}v + \oint_{\partial\Omega} F(\boldsymbol{u}, \boldsymbol{n})\,\mathrm{d}s = 0 \tag{1}$$

Here $\boldsymbol{u}$ is the solution vector comprised of the conservative variables density, the two components of momentum, and total energy. The vector $\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{n})$ represents the inviscid flux vector for a surface with normal vector $\boldsymbol{n}$. The net efflux through the control volume boundary is termed the residual. The variables $\boldsymbol{u}$ are stored at the vertices of a triangular mesh. The control volumes are nonoverlapping polygons that surround the vertices of the mesh. They form the dual of the mesh, which is composed of segments of medians. Associated with each edge of the original mesh is a (segmented) dual edge. The contour integrals in Eq. (1) are replaced by discrete path integrals over these dual edges. The flux $\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{n})$ is replaced by a numerical flux function. The construction of the numerical fluxes is done in two stages:

1) First, a piecewise-linear reconstruction of the variables is performed. The variables are then interpolated to the edges of the control volumes. The gradients at the vertices that are required for this step are also computed as discrete path integrals over the edges of the control volumes.

Presented as Paper 94-0759 at the AIAA 32nd Aerospace Sciences Meeting, Reno, NV, Jan. 10–13, 1994; received Jan. 26, 1994; revision received April 20, 1994; accepted for publication April 25, 1994.

*Senior Staff Scientist, MS 132C, NASA Langley Research Center. Member AIAA.

2) The variables on either sides of the control volume edges are interpreted as initial data for Roe's approximate Riemann solver.[11] A Riemann problem is defined by Eq. (1) subject to two constant states as initial data in one dimension. More details on the spatial discretization may be found in Barth and Jespersen.[1] A four-stage Runge-Kutta scheme is used to advance the solution in time for the explicit scheme. The residuals are evaluated by looping over the edges of the original mesh, and vectorization is achieved by coloring the edges. For the parallel implementation, it is assumed that the triangulation of the computational domain and the mesh connectivity information are provided.

Two slightly different schemes are derived based on whether the cells or the vertices of the mesh are partitioned. For explicit schemes, identical solutions and convergence histories are obtained irrespective of this choice. The only difference lies in the way the communication is performed, which is discussed in the following paragraph. With the implicit schemes considered in this paper, on the other hand, one will in general obtain different convergence histories depending on this choice. The reason for this is that the schemes considered in this paper are only implicit within each processor (in the preconditioning phase). Cell partitioning leads to an overlap of vertices, whereas vertex partitioning yields none, and therefore, the two techniques produce different results.

With cell partitioning, each triangle is assigned to a partition, and the interpartition boundaries consist of edges of the original mesh. The vertices and the edges on the interpartition boundaries are duplicated. Figure 1 shows a triangular mesh under a two-way cell partitioning. Also shown is the numbering of vertices local to each processor. Two communication phases are then required at each stage of the four-stage Runge-Kutta time integration, one during the computation of the gradients and the other during the formation of the residuals. At an interpartition boundary vertex, which is shared by two or more partitions, each processor only sees a fraction of the control volume and thus computes only partial contributions to the integrals. The communication at the interpartition boundaries then consists of summing these local contributions to the integrals such as volumes, fluxes, and gradients.

With vertex partitioning, each vertex is assigned uniquely to a partition, and the interpartition boundaries consist of the dual edges (the edges of the control volumes). Figure 2 shows a triangular mesh under a two-way vertex partitioning. The numbering of vertices is indicated as tuples, where the first index refers to the numbering for processor 0 and the second index refers to the numbering for processor 1. Again, two communication phases are required at each stage of the four-stage Runge-Kutta time integration. The communication required is different, however. The processors exchange data at two rows of vertices that are incident to the interpartition boundary edges. Now each processor can compute the entire integrals for all of the vertices it owns. Thus, duplication of the flux calculations occurs at the interpartition boundary edges, but this is not a crucial issue on medium-grained parallel computers.

Simon[12] has considered three different recursive partitioning strategies for partitioning unstructured meshes. In the context of an explicit two-dimensional Euler solver, Venkatakrishnan et al.[9] showed that the spectral bisection strategy[23] was superior to the coordinate and graph bisection strategies in terms of the communication costs. The spectral bisection technique produces uniform, mostly connected subdomains with short boundaries. Therefore, in this work, the recursive spectral bisection is used for partitioning.

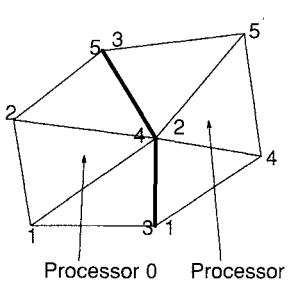

Fig. 1 Two-way cell partitioning for a simple triangulation.

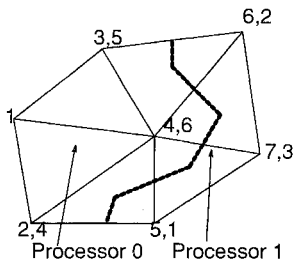

Fig. 2 Two-way vertex partitioning for a simple triangulation.

After partitioning, global values of the data structures required to define the unstructured mesh are given local values within each partition. We thus dispense with any references to global indices. Das et al.[10] in their work develop primitives that allow access to global data address space, so that the transformation from global to local data structures is unnecessary. Access to global data structures is highly desirable for adaptive grid applications but is not necessary when dealing with static meshes. In the present implementation, each local data set also contains the information that a partition requires for communication at its interpartition boundaries. The information required for communication at the interpartition boundaries is precomputed using sparse matrix data structures. Slightly different data structures are used depending on whether cell partitioning or vertex partitioning is employed. Each subgrid is assigned to one processor. It was shown in Ref. 9 that the mapping of the subgrids to processors is not a crucial issue on the Intel iPSC/860. Therefore, a naive mapping that assigns subgrid 0 to processor 0, subgrid 1 to processor 1, and so on, is employed. Partitioning, conversion from global to local addresses, and generation of the data structures required for communication at the interpartition boundaries are done on a workstation as a preprocessing step.

Implicit Scheme

After discretizing Eq. (1) in space, the following system of coupled ordinary differential equations is obtained:

$$M\frac{dW}{dt} + R(W) = 0 \tag{2}$$

Here W is the vector of unknowns over all of the mesh points, and M is the mass matrix that represents the relationship between the average value in a control volume and the values at the vertices (the vertex representing the control volume and its nearest neighbors). It is only a function of the mesh and, hence, a constant matrix for a static mesh. Since a steady-state solution is sought, time accuracy is not an issue, and M can be replaced by the identity matrix yielding the following system of ordinary differential equations for the vector of unknowns W:

$$\frac{dW}{dt} + R(W) = 0 \tag{3}$$

If the time derivative is replaced by

$$\frac{dW}{dt} = \frac{W^{n+1} - W^n}{\Delta t} \tag{4}$$

an explicit scheme is obtained by evaluating $R(W)$ at time level n. An implicit scheme is obtained by evaluating $R(W)$ at level $n+1$. In the latter case, linearizing R about time level n, we obtain

$$\left(\frac{I}{\Delta t} + \frac{\partial R}{\partial W}\right)\Delta W_i = -R_i \tag{5}$$

$$\Delta W_i = (W^{n+1} - W^n)_i \tag{6}$$

Equation (5) represents a large nonsymmetric linear system of equations for the updates of the vector of unknowns and needs to be solved at each time step. As Δt tends to infinity, the method reduces to the standard Newton's method. The term $\partial R/\partial W$ symbolically represents the implicit side upon linearization and involves the Jacobian matrices of the flux vectors with respect to the conservative variables. Because of storage considerations, only a lower order representation of the operator is employed. As a result of this approximation, Eq. (5) can never approach Newton's method (with its associated quadratic convergence property) due to the mismatch of the right- and left-hand side operators. Based on the work of Mulder and van Leer,[13] the time step in Eq. (5) is allowed to vary inversely proportional to the L_2 norm of the residual. Since there is a mismatch of operators in Eq. (5), it is necessary to limit the maximum time step.

There are many methods in linear algebra literature for solving nonsymmetric systems of linear equations, but in this work as in Ref. 6 only the GMRES technique developed by Saad and Schultz[7] is considered. The GMRES technique is quite efficient for solving sparse nonsymmetric linear systems and is outlined next. Let x_0 be an approximate solution of the system

$$Ax + b = 0 \tag{7}$$

where A is an invertible matrix. The solution is advanced from x_0 to x_k as

$$x_k = x_0 + y_k \tag{8}$$

GMRES(k) finds the best possible solution for y_k over the Krylov subspace $< v_1, Av_1, A^2v_1, \ldots, A^{k-1}v_1 >$ by solving the minimization problem

$$\|r_k\| = \mathrm{Min}_y\|v_1 + Ay\| \tag{9}$$

$$v_1 = Ax_0 + b, \qquad r_k = Ax_k + b \tag{10}$$

GMRES procedure forms an orthogonal basis $v_1, v_2, \ldots, v_k$ (termed search directions) spanning the Krylov subspace by a modified Gram-Schmidt method. These search directions need to be stored. As k increases, the storage increases linearly and the number of operations, quadratically. GMRES can also be thought of as an optimal polynomial acceleration scheme. Preconditioning greatly improves the performance of GMRES as well as the other related iterative methods. It clusters the eigenvalues around unity so that the optimal polynomial generated by GMRES can better annihilate the errors associated with each eigenvalue.

Preconditioning and Parallelism Issues

Instead of Eq. (7) the preconditioned iterative methods solve the following system:

$$AQ(Q^{-1}x) + b = 0 \tag{11}$$

The system of linear equations in Eq. (11) is referred to as the right preconditioned system and Q as the right preconditioner. The role of the preconditioner is to cluster the eigenvalues around unity. The choice of preconditioners and the issues in implementing the preconditioned GMRES on the parallel computer are addressed next.

On a distributed memory parallel computer, the same least-squares problem of Eq. (9) is seen by all of the processors. Although this results in some duplication of work, the main nonlocal kernels of the GMRES are distributed across multiple processors. These kernels include sparse matrix-vector multiplication, dot products, and L_2 norm evaluations. Whereas on a Cray Y-MP, vectorization for the sparse matrix-vector product was achieved by using an edge-oriented data structure for the matrix and coloring the edges of the graph,[6] this is not the optimal way to compute the matrix-vector product on a parallel computer, where locality is of utmost importance. This is true even when dealing with a single node of the parallel computer because of the memory hierarchy. Therefore the usual row-oriented sparse matrix data structure is used. We have found that even on a single node this approach outperforms the one that uses the edge-based data structure by a factor of 2. In the case of cell partitioning, the rows corresponding to the interpartition boundary vertices are distributed across the processors sharing them, whereas the remaining rows are assigned uniquely to processors. In the case of vertex partitioning, the rows are uniquely assigned to processors. Akin to the explicit scheme, each processor computes its share of the matrix vector multiplication. The communication step with cell partitioning consists of summing the local contributions at the interpartition boundaries. The communication step with vertex partitioning consists of exchange of the vector components at the two rows of vertices incident to the interpartition boundary edges. This is followed by the matrix-vector multiplication at each processor. More details on the implementations of the matrix-vector product on vector-parallel and distributed memory computers may be found in Ref. 14.

In most practical problems of interest, the choice of the preconditioner is very important, but the effort involved in applying the preconditioner should not be prohibitive. The implicit scheme without preconditioning possesses complete parallelism, except for the duplication of some work when solving the least-squares problem in GMRES. On a parallel computer, the parallelism in the preconditioning phase is an important additional consideration. A simple choice is a block diagonal preconditioner that computes the inverse of the 4×4 diagonal block associated with a mesh point. The LU decomposition of the 4×4 blocks and the forward and back solutions are local and, hence, are inherently parallel.

A family of preconditioners has been developed by Meijerink and van der Vorst[15] wherein a sparsity pattern is specified and the entries outside this pattern that occur during the factorization are ignored. In Ref. 6, an ILU(0) preconditioner was considered. ILU(0) refers to an incomplete lower-upper factorization with no fill-in allowed beyond the original nonzero pattern. By using a level scheduling[16] (also known as wave front ordering), it is possible to obtain parallelism with this preconditioner. Under this permutation of the matrix, unknowns within a wave front can be eliminated simultaneously. However, since the degree of parallelism varies with the wave front, it cannot be easily exploited on a distributed memory parallel computer. A fixed partitioning strategy for the mesh incurs substantial load imbalance, whereas a dynamic partitioning strategy entails substantial data movement and, hence, increased communication costs. Greenbaum[17] has found that using a fixed partitioning strategy when solving triangular systems of equations on a regular grid results in low upper bounds on efficiency even in the absence of communication. A higher degree of parallelism in ILU(0) can be achieved by using a different ordering of unknowns, but typically such an ordering adversely affects the convergence of the underlying iterative method. Therefore, for general sparse matrices, the ILU(0) preconditioner is ill suited for implementation on a distributed memory parallel computer. Therefore, we settle on an ILU preconditioner that is processor-implicit, i.e., ILU(0) is carried out for all of the vertices internal to a processor. Thus, at a macrolevel, the overall preconditioner can be viewed as an approximate block Jacobi iteration, wherein each block is assigned to a processor and an approximate LU factorization, viz., ILU(0), is carried out. A block here refers to a macroblock consisting of all of the unknowns assigned to a processor. When cell partitioning is employed, a block diagonal preconditioner is used for all of the vertices on the interprocessor boundaries, where a block refers to the 4×4 matrix associated with each vertex. The overall preconditioner is weaker than the global ILU(0) and degenerates to a block diagonal preconditioner in the limit of one grid point per processor. Thus, as the number of processors increases, degradation in convergence is to be expected. This degradation should be moderate, since the iPSC/860 is a coarse-grained parallel computer.

When using vertex partitioning, there is no overlap of vertices between processors, and hence no special interface preconditioning is required when GMRES/ILU is used. To facilitate communication, vertices on either sides of the partition boundaries are

duplicated, thus leading to a minimal overlap. In the preconditioning phase, ILU factorization is carried out for each processor by zeroing out the matrix entries whose column numbers lie outside the processor domain. This is equivalent to solving the problem within each processor with zero Dirichlet boundary conditions during the preconditioning. This approximation is consistent with the steady-state solution $\Delta W \equiv 0$ everywhere. Again, degradation in convergence is to be expected as the number of processors increases. More iterations will be required to obtain the same level of convergence. This additional work will be called the sequential overhead.

To minimize the sequential overhead, we appeal to techniques developed in domain decomposition. One of the most successful methods in use in domain decomposition is the Schwarz alternating procedure for overlapping subdomains, which can also be implemented as a preconditioner. Two variants of this procedure have been developed in the literature: the additive and the multiplicative algorithms (see Dryja and Widlund[18]). The term additive denotes that the preconditioner can be carried out independently for each subdomain. The scheme outlined earlier is an example of an additive Schwarz preconditioner. In contrast, the multiplicative Schwarz method requires that the preconditioner be applied in a sequential way by cycling through the subdomains in some order. It is possible to extract some coarse-grained parallelism by coloring the subdomains, but the potential is limited. Therefore, in a parallel context, the additive Schwarz method is preferred.

A powerful idea for elliptic problems advocated by Dryja and Widlund[18] is the use of a coarse grid to bring some global influence to bear on the problem, similar in spirit to a two-level multigrid algorithm. The coarse grid operator is applied multiplicatively, i.e., the coarse grid problem is solved first. Its solution is subsequently used by the processors during the additive (parallel) phase as Dirichlet data at the subdomain boundaries. Applying the coarse grid in this manner does impose a penalty in a parallel setting; it becomes a sequential bottleneck. Cai et al.[19] have applied the multiplicative and additive Schwarz algorithms to the solution of nonsymmetric elliptic problems. They have observed almost h-independent convergence, where h is the fine grid size, provided the coarse grid is fine enough. In their work, the coarse grid operator was formed by discretizing the partial differential equation on a coarse grid. However, in our application, this will require a triangulation followed by a discretization on this coarse grid. Generation of interpolation operators to transfer information between the coarse and the fine grids will also be necessary. We avoid all of these complexities by appealing to an alternative way of obtaining a coarse grid operator described by Wesseling.[20] A coarse grid Galerkin operator is easily derived from a given fine grid operator by specifying the restriction and prolongation operators. We choose the restriction operator to be a simple summation of fine grid values and the prolongation operator to be injection. Under this choice, the coarse grid discretization is similar to the one used in an agglomeration multigrid strategy (see Lallemand et al.[21] and Venkatakrishnan and Mavriplis[22]). It amounts to identifying all of the vertices that belong to a subdomain by one coarse grid vertex and summing the equations and the right-hand sides associated with them. Thus the coarse grid system has as many vertices as the number of subdomains. At each time step, a coarse grid system is formed and solved by using a direct solver. The data obtained from the coarse grid are used on the boundaries as Dirichlet data for each subdomain. We have found that, in practice, a direct solver is seldom needed to solve the coarse grid system; an iteration of incomplete LU decomposition seems to suffice. We have also found that at least one cycle of implicit smoothing similar to that employed in multigrid context by Mavriplis and Jameson[2] is needed to mitigate the adverse effects of injection of the solution from the coarse to the fine grid. Therefore, on the fine grid, after injection, one Jacobi iteration is performed on the following system of implicit equations:

$$(I + \varepsilon d)u_i^{\text{new}} = u_i^{\text{old}} + \sum_{j=1}^{d} \varepsilon u_j^{\text{new}} \tag{12}$$

where ε is taken to be 0.5, d is the degree of the vertex i, and the summation is over the neighbors of each vertex. This smoothing step involves communication at the boundaries.

The vertex-partitioned scheme is more amenable to the investigation of the coarse grid operator than the cell-partitioned scheme, and therefore we only discuss this particular implementation. Each processor first forms parts of the coarse grid matrix and the right-hand side at every time step. A global concatenation is performed so that each processor has the entire coarse grid system. This system is solved redundantly by each processor by forming approximate lower and upper factors. During the preconditioning phase, each processor forms a portion of the right-hand side. After a global concatenation, each processor carries out forward and backward solutions and deduces the appropriate Dirichlet data. We have also developed a weaker smoother that dispenses with communication associated with the Jacobi smoothing but yields comparable convergence. This technique, termed modified Jacobi smoothing, smooths the neighboring coarse grid data (to be used as Dirichlet data) with the data that the processor holds. This step is given by the following relation:

$$(I + \varepsilon)U_D^{\text{new}} = U_D + \varepsilon U_{\text{LOC}} \tag{13}$$

where U_D is the old Dirichlet data, U_D^{new} new is the new Dirichlet data, and U_{LOC} is the value of the coarse grid vertex assigned to the processor.

Performance on the Intel iPSC/860

Flow past a four-element airfoil in a landing configuration at a freestream Mach number $M_\infty = 0.2$ and an angle of attack of 5 deg is considered as a test case. Performance results are presented for two problem sizes that are representative of two-dimensional inviscid flows. The coarse mesh has 6019 vertices, 17,473 edges, 11,451 triangles, 4 bodies, and 593 boundary edges. The fine mesh has 15,606 vertices, 45,878 edges, 30,269 triangles, 4 bodies, and 949 boundary edges. In the Cray implementation of the explicit code,[1] vectorization is achieved by coloring the edges of the mesh and the code runs at 150 megaflops. The implicit code was not optimized for the Cray Y-MP, since it was developed on the Intel iPSC/860. The result is that it runs in an almost scalar fashion on the Cray, except for the right-hand side computation. However, a similar implicit unstructured mesh Navier-Stokes code was imple-

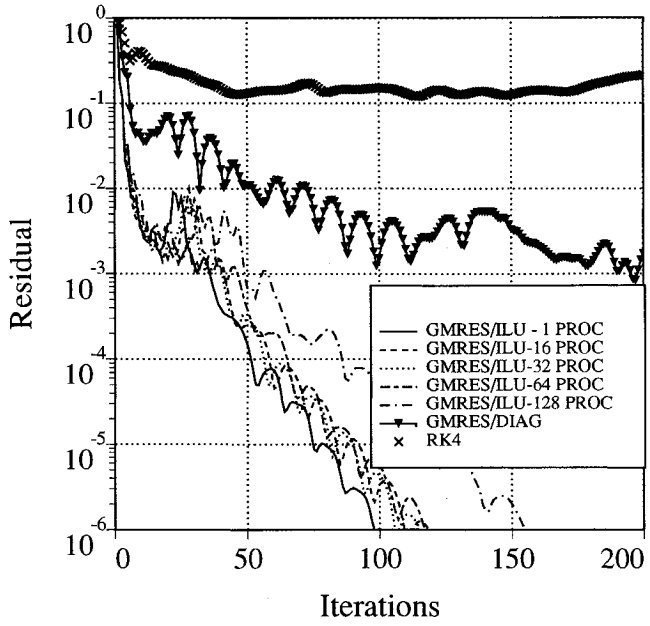

Fig. 3 Convergence histories with various schemes on the fine mesh with cell partitioning.

Table 1 Performance of the cell-partitioned implicit scheme on the Intel iPSC/860—6019 vertices

Scheme	Measure	No. of processors					
		1	4	8	16	32	64
RK4	Time/iteration, s	—	1.06	0.58	0.33	0.20	0.13
	Conv. rate/iteration	0.973	0.973	0.973	0.973	0.973	0.973
GMRES/DIAG	Time/iteration, s	—	3.12	1.72	1.00	0.64	0.46
	Conv. rate/iteration	0.874	0.874	0.874	0.874	0.874	0.874
GMRES/ILU	Time/iteration, s	—	4.47	2.40	1.34	0.82	0.55
	Conv. rate/iteration	0.791	0.797	0.798	0.793	0.799	0.802

Table 2 Performance of the cell-partitioned implicit scheme on the Intel iPSC/860—15,606 vertices

Scheme	Measure	No. of processors				
		1	16	32	64	128
RK4	Time/iteration, s	—	0.85	0.44	0.25	0.16
	Conv. rate/iteration	0.997	0.997	0.997	0.997	0.997
GMRES/DIAG	Time/iteration, s	—	2.27	1.30	0.81	0.56
	Conv. rate/iteration	0.968	0.968	0.968	0.968	0.968
GMRES/ILU	Time/iteration, s	—	3.16	1.74	1.04	0.68
	Conv. rate/iteration	0.870	0.880	0.883	0.885	0.903

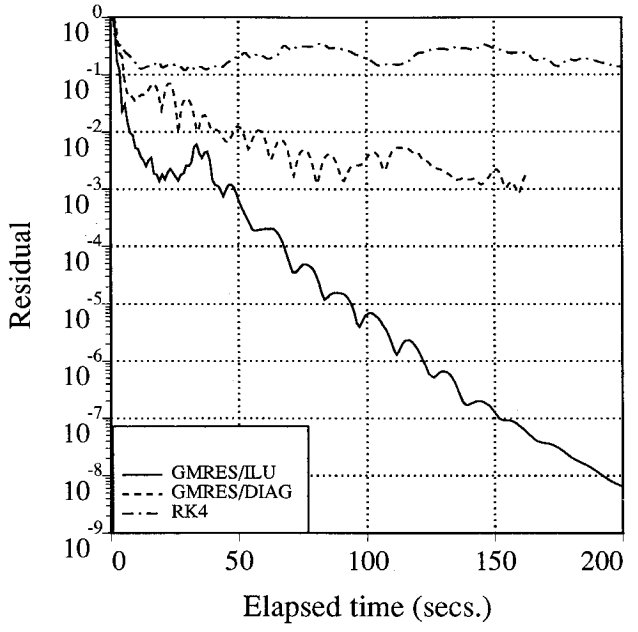

Fig. 4 Convergence histories as a function of elapsed times on the fine mesh with 64 processors.

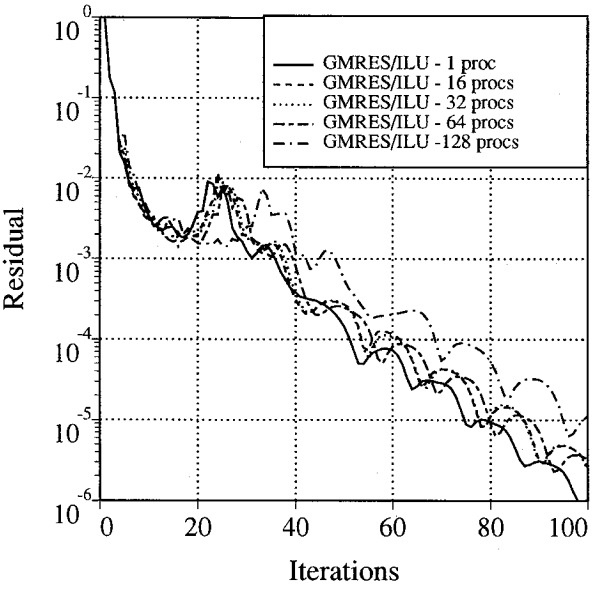

Fig. 5 Convergence histories with GMRES/ILU on the fine mesh with vertex partitioning.

mented earlier on the Cray Y-MP and optimized in Ref. 6, which runs at around 110–120 megaflops. The megaflop numbers are based on operation counts using the Cray hardware performance monitor. The megaflop ratings of the parallel implementation may be obtained by scaling the execution times by those of the Cray implementation.

The explicit scheme is a four-stage Runge-Kutta scheme and uses a Courant-Friedrichs-Lewy (CFL) number of 1.4. With the GMRES/DIAG scheme, the startup CFL number is 3, and the CFL number is allowed to vary inversely proportional to the L_2 norm of the residual up to a maximum of 30. With GMRES/ILU, the startup CFL number is 20, and the CFL number is allowed to vary inversely proportional to the L_2 norm of the residual up to a maximum of 200,000. With both of the implicit schemes, the number of GMRES search directions used is 15.

First, the explicit and implicit schemes are compared when cell partitioning is employed. When using GMRES/ILU, a block diagonal preconditioning is employed for the vertices on the interprocessor boundaries. Figure 3 shows the convergence characteristics of the explicit and implicit schemes as a function of the number of iterations for the fine mesh. It may be observed that the explicit scheme is barely converging, whereas the implicit schemes converge much faster. The GMRES/ILU processor-implicit preconditioning exhibits degradation in convergence as the number of processors increases, but the degradation is moderate. Even with 128 processors, it performs much better than the GMRES/DIAG. In examining Fig. 3, one can see that the convergence histories with GMRES/ILU gravitate toward those of GMRES/DIAG as the number of processors increases. In the limit of one grid point per processor the two will be identical. Since the problem does not fit on one processor of the Intel iPSC/860, the uniprocessor runs were carried out on the Cray Y-MP. The times per iteration in seconds and the convergence rates are shown in Tables 1 and 2 for the coarse and fine meshes, respectively. The convergence rate is defined as

$$\text{Rate} = \left(\frac{R_n}{R_1}\right)^{\frac{1}{n-1}} \tag{14}$$

where R_n is the L_2 norm of the residual at the end of nth time step, and R_1 is the residual at the end of the first time step. It may be ob-

served that the convergence rates of the explicit scheme (RK4) and the implicit scheme (GMRES/DIAG) are independent of the number of processors used, whereas that of GMRES/ILU exhibits a slight degradation with an increasing number of processors. Finally, since time to completion is of ultimate interest, Fig. 4 shows the convergence histories for the fine mesh problem as a function of the elapsed time with the number of processors fixed at 64. It clearly shows the superiority of the GMRES/ILU processor-implicit technique over the explicit and the GMRES/DIAG schemes.

Next, the explicit and implicit schemes are compared when vertex partitioning is employed. Figure 5 shows the convergence histories for the fine mesh when using vertex partitioning. The convergence histories are comparable to those obtained with cell partitioning (Fig. 3). Tables 3 and 4 show the times per iteration in seconds and the convergence rates. In comparing the results with those for cell partitioning (Tables 1 and 2), one can see that the elapsed times with vertex partitioning are slightly better than those with cell partitioning for all of the algorithms, especially for the implicit schemes. In addition, as the number of processors increases, the vertex-partitioned GMRES/ILU scheme exhibits less degradation in convergence in comparison with the cell-partitioned scheme. The convergence rates shown in Tables 3 and 4 bear this out as well. Even with 128 processors, the GMRES/ILU scheme requires only about 20% more iterations than the ideal uniprocessor scheme to obtain the same level of convergence (five orders of reduction in the residual norm).

Finally, we examine the effects of using a coarse grid as discussed in the last section to improve convergence for the 15,606 vertices mesh. Figure 6 shows the convergence histories as a function of iterations for the uniprocessor, 32-processor, and 128-processor cases with and without the use of a coarse grid. A cycle of modified Jacobi smoothing is employed as part of the preconditioner to stabilize the procedure with the coarse grid system, and the coarse grid system is solved redundantly by all processors. The convergence has improved significantly, illustrating the power of the coarse grid; the convergence with 128 processors is even better than that obtained with the uniprocessor scheme. Unfortunately, this improved convergence does not translate into a reduction in the time required to solve the problem. This is illustrated in Fig. 7, which shows the convergence histories as a function of elapsed times on 32 and 128 processors with and without the coarse grid. In both the 32- and the 128-processor cases, it may be observed that the times required to solve the problem are nearly the same with and without the use of the coarse grid. On a per iteration basis, the elapsed times for the 32-processor case are 1.73 and 1.87

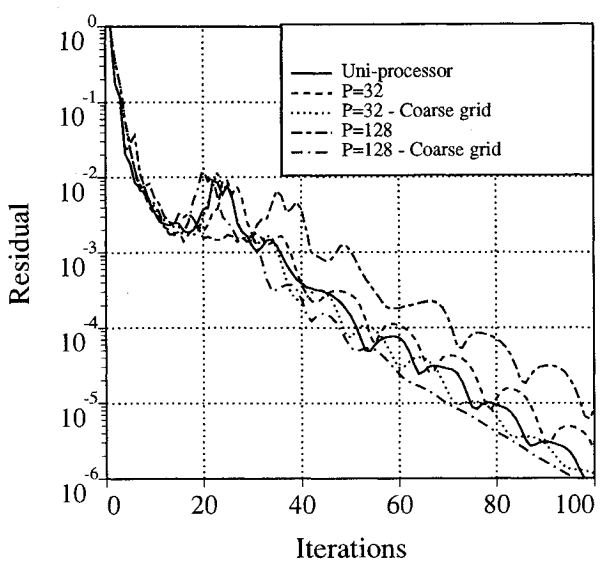

Fig. 6 Convergence histories for the fine mesh as a function of iterations with and without the use of a coarse grid.

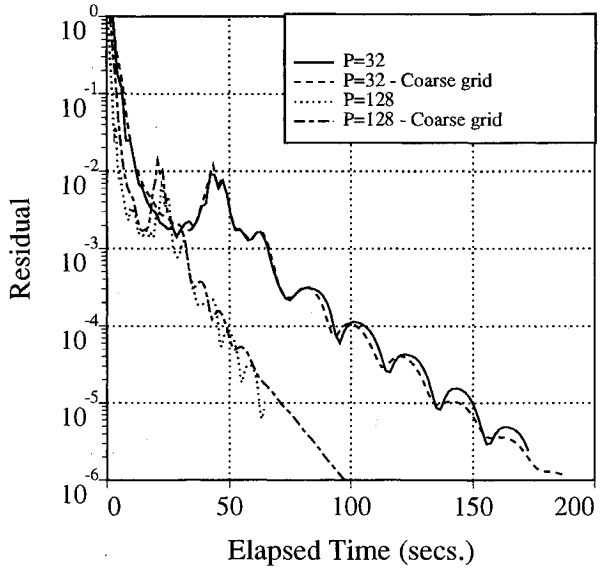

Fig. 7 Convergence histories for the fine mesh as a function of elapsed times with and without the use of a coarse grid.

Table 3 Performance of the vertex-partitioned implicit scheme on the Intel iPSC/860—6019 vertices

		No. of processors					
Scheme	Measure	1	4	8	16	32	64
RK4	Time/iteration, s	—	1.07	0.59	0.32	0.20	0.13
	Conv. rate/iteration	0.973	0.973	0.973	0.973	0.973	0.973
GMRES/DIAG	Time/iteration, s	—	3.06	1.66	0.95	0.59	0.42
	Conv. rate/iteration	0.874	0.874	0.874	0.874	0.874	0.874
GMRES/ILU	Time/iteration, s	—	4.42	2.36	1.32	0.77	0.52
	Conv. rate/iteration	0.791	0.795	0.796	0.797	0.797	0.797

Table 4 Performance of the vertex-partitioned implicit scheme on the Intel iPSC/860—15,606 vertices

		No. of processors				
Scheme	Measure	1	16	32	64	128
RK4	Time/iteration, s	—	0.78	0.43	0.25	0.15
	Conv. rate/iteration	0.997	0.997	0.997	0.997	0.997
GMRES/DIAG	Time/iteration, s	—	2.19	1.24	0.75	0.51
	Conv. rate/iteration	0.968	0.968	0.968	0.968	0.968
GMRES/ILU	Time/iteration, s	—	3.07	1.73	1.07	0.65
	Conv. rate/iteration	0.870	0.878	0.878	0.880	0.891

s, respectively, without and with the coarse grid. For the 128-processor case, these times are 0.64 and 1.02 s. This points to a major drawback of using the coarse grid system to improve convergence, especially on a parallel computer. With too small a coarse grid system, the effort required to solve the system is minimal, but so is the improvement in convergence. With a larger coarse grid system, the gain in convergence is substantial but comes at a greater cost. The coarse grid operator, being sequential in nature, predominates as the number of processors increases. When invoking the coarse grid operator, a large fraction of the time is spent in the global concatenation. Thus, if the parallel computer were to have better communication rates, the technique would be more competitive in terms of elapsed time as well.

Conclusions

It has been shown that implicit schemes can be carefully designed to yield good performance when solving unstructured grid problems on parallel computers. Explicit schemes are not competitive despite their parallelism. The GMRES technique with ILU preconditioning within each processor performs better than the GMRES procedure with block diagonal preconditioning. In the case where cells are partitioned, a block diagonal preconditioning for the interface points is shown to be effective. It is shown that vertex partitioning yields slightly better results, both in terms of elapsed times and convergence rates. The implicit schemes presented in this paper offer almost complete parallelism at the expense of minimal sequential overhead, thus resulting in efficient parallel algorithms. Finally, it is demonstrated that the use of a global coarse grid, although improving convergence, does not result in a reduction in the time to solve the problem on the Intel iPSC/860.

Acknowledgments

Part of this work was done while the author was employed by Computer Sciences Corporation, Moffett Field, CA, during which time the work was supported under NASA Contract NAS2-12961. This research was also supported under NASA Contract NAS1-19480 while the author was in residence at the Institute for Computer Applications in Science and Engineering. The author thanks the Numerical Aerodynamics Simulation facility at NASA Ames Research Center for the use of the Intel iPSC/860. The author also thanks Tim Barth of NASA Ames Research Center for providing the explicit Cray code and David Keyes and Moulay Tidiri of ICASE for the many helpful discussions on domain decomposition methods.

References

[1]Barth, T. J., and Jespersen, D., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, Jan. 1989.

[2]Mavriplis, D. J., and Jameson, A., "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Grids," *AIAA Journal*, Vol. 26, No. 7, 1988, pp. 824–831.

[3]Shakib, F., Hughes, T. J. R., and Johan, Z., "A Multi-Element Group Preconditioned GMRES Algorithm for Nonsymmetric Problems Arising in Finite Element Analysis," *Computer Methods in Applied Mechanics and Engineering*, Vol. 87, 1989, pp. 415–456.

[4]Whitaker, D. L., Slack, D. C., and Walters, R. W., "Solution Algorithms for the Two-Dimensional Euler Equations on Unstructured Meshes," AIAA Paper 90-0967, Jan. 1990.

[5]Whitaker, D. L., "Three-Dimensional Unstructured Grid Euler Computations Using a Fully-implicit, Upwind Method," AIAA Paper 93-3337, July 1993.

[6]Venkatakrishnan, V., and Mavriplis, D. J., "Implicit Solvers for Unstructured Meshes," *Journal of Computational Physics*, Vol. 105, No. 1, 1993, pp. 83–91.

[7]Saad, Y., and Schulz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal of Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[8]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," *Computer Methods in Applied Mechanics and Engineering*, Vol. 99, 1992, pp. 113–134.

[9]Venkatakrishnan, V., Simon, H. D., and Barth, T. J., "A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids," *The Journal of Supercomputing*, Vol. 6, 1992, pp. 117–137.

[10]Das, R., Mavriplis, D. J., Saltz, J., Gupta, S., and Ponnusamy, R., "The Design and Implementation of a Parallel Unstructured Euler Solver Using Software Primitives," AIAA Paper 92-0562, Jan. 1992; also *AIAA Journal*, Vol. 32, No. 3, 1994, pp. 489–496.

[11]Roe, P. L., "Characteristic-Based Schemes for the Euler Equations," *Annual Review of Fluid Mechanics*, Vol. 18, 1986, pp. 337–365.

[12]Simon, H. D., "Partitioning of Unstructured Problems for Parallel Processing," *Computing Systems in Engineering*, Vol. 2, No. 2/3, 1991, pp. 135–148.

[13]Mulder, W. A., and van Leer, B., "Implicit Upwind Methods for the Euler Equations," AIAA Paper 83-1930, July 1983.

[14]Venkatakrishnan, V., "Parallel Computation of Ax and A^Tx," *International Journal of High Speed Computing* (to be published).

[15]Meijerink, J. A., and van der Vorst, H. A., "Guidelines for the Usage of Incomplete Decompositions in Solving Sets of Linear Equations as They Occur in Practical Problems," *Journal of Computational Physics*, Vol. 44, No. 1, 1981, pp. 134–155.

[16]Anderson, E., and Saad, Y., "Solving Sparse Triangular Systems on Parallel Computers," *International Journal of High Speed Computing*, Vol. 1, No. 1, 1989, pp. 73–96.

[17]Greenbaum, A., "Solving Sparse Triangular Linear Systems Using FORTRAN with Parallel Extensions on the NYU Ultracomputer Prototype," New York Univ., Technical Report Ultracomputer Note 99, New York, April 1986.

[18]Dryja, M., and Widlund, O. B., "Towards a Unified Theory of Domain Decomposition Algorithms for Elliptic Problems," *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, edited by T. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 3–21.

[19]Cai, X., Gropp, W. D., and Keyes, D. E., "A Comparison of Some Domain Decomposition and ILU Preconditioned Algorithms for Nonsymmetric Elliptic Problems" (to be published).

[20]Wesseling, P., *An Introduction to Multigrid Methods*, Wiley, New York, 1992.

[21]Lallemand, M. H., Steve, H., and Dervieux, A., "Unstructured Multigridding by Volume Agglomeration: Current Status," *Computers and Fluids*, Vol. 21, No. 3, 1992, pp. 397–433.

[22]Venkatakrishnan, V., and Mavriplis, D. J., "Agglomeration Multigrid for the Three-Dimensional Euler Equations," AIAA Paper 94-0069, Jan. 1994.

[23]Potheu, A., Simon, H. D., and Liou, K-P., "Partitioning Sparse Matrices with Eigenvectors of Graphs," *SIAM Journal of Mathematical Analysis Applications*, Vol. 11, July 1990, pp. 430–452.