NASA/CR-1999-209098
ICASE Report No. 99-11

Parallelization of an Object-oriented Unstructured Aeroacoustics Solver

Abdelkader Baggag
Purdue University, West-Lafayette, Indiana

Harold Atkins
NASA Langley Research Center, Hampton, Virginia

Can Özturan
Bogazici University, Istanbul, Turkey

David Keyes
ICASE, Hampton, Virginia and
Old Dominion University, Norfolk, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center, Hampton, VA
Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

February 1999

PARALLELIZATION OF AN OBJECT-ORIENTED UNSTRUCTURED AEROACOUSTICS SOLVER [*]

ABDELKADER BAGGAG[†], HAROLD ATKINS[‡], CAN ÖZTURAN[§], AND DAVID KEYES[¶]

Abstract. A computational aeroacoustics code based on the discontinuous Galerkin method is ported to several parallel platforms using MPI. The discontinuous Galerkin method is a compact high-order method that retains its accuracy and robustness on non-smooth unstructured meshes. In its semi-discrete form, the discontinuous Galerkin method can be combined with explicit time marching methods making it well suited to time accurate computations. The compact nature of the discontinuous Galerkin method also makes it well suited for distributed memory parallel platforms. The original serial code was written using an object-oriented approach and was previously optimized for cache-based machines. The port to parallel platforms was achieved simply by treating partition boundaries as a type of boundary condition. Code modifications were minimal because boundary conditions were abstractions in the original program. Scalability results are presented for the SGI Origin, IBM SP2, and clusters of SGI and Sun workstations. Slightly superlinear speedup is achieved on a fixed-size problem on the Origin, due to cache effects.

Key words. discontinuous Galerkin method, object-oriented, unstructured grids, Euler equations, high-order accuracy, superlinear speedup

Subject classification. Computer Science

1. Motivation. Computational Aeroacoustics (CAA) involves the direct simulation of sound generation and/or propagation about an aircraft or an aircraft component. To be of practical value in the aircraft design process, these massive computations must be performed quickly, and to do so requires efficient use of parallel computer platforms.

CAA methods must provide both temporal and spatial accuracy beyond what the second-order discretizations employed in most other areas of computational aerodynamics are capable of providing. In addition, such methods must be easy to apply to complex geometries without sacrifice of accuracy or robustness. These requirements further complicate the design of the parallel implementation. For instance, traditional high-order finite-difference methods are not compact and the amount of data that must be moved across partition boundaries increases considerably with the order of the method. The requirement for time accuracy means all partitions must be advanced in lock step. Common techniques used in steady calculations, such as lagging some information or communicating only after several iterations, cannot be employed.

The discontinuous Galerkin method is a relatively new approach that satisfies the numerical requirements of CAA and the algorithmic requirements of parallel implementation. Discontinuous Galerkin is a compact

[†]Department of Computer Sciences, Purdue University, 1398 Computer Science Building, West-Lafayette, IN 47907-1398, (e-mail: `baggag@cs.purdue.edu`)

[‡]Aerodynamic and Acoustic Methods Branch, NASA Langley Research Center, Hampton, VA 23681-2199, (e-mail: `h.l.atkins@larc.nasa.gov`)

[§]Department of Computer Engineering, Bogazici University, Istanbul, Turkey, (e-mail: `ozturaca@boun.edu.tr`)

[¶]ICASE NASA Langley Research Center, Hampton, VA 23681-2199, (e-mail: `keyes@icase.edu`) and Old Dominion University, Department of Computer Science, Norfolk, VA 23529-0162, (e-mail: `keyes@cs.odu.edu`).

method that can be applied to structured or unstructured grids. Many of the method's accuracy and stability properties have been rigorously proven [1, 2, 3, 4, 5] for arbitrary element shapes, any number of spatial dimensions, and even for nonlinear problems, which lead to a very robust method. It has been demonstrated in mesh refinement studies [6] that the accuracy of this method does not depend upon the smoothness of the mesh. Furthermore, the method requires no special treatment near boundaries, which are problematic for many high-order methods. These features are crucial for the robust treatment of complex geometries. In semi-discrete form, the discontinuous Galerkin method can be combined with explicit time-marching methods, such as Runge-Kutta, to create a method well suited for CAA applications. The method has been criticized for its high storage and high computational requirements; however, a recently developed quadrature-free implementation [6] has greatly ameliorated these concerns. The quadrature-free form of the discontinuous Galerkin method has been implemented and validated [6, 7, 8] in an object-oriented code for the prediction of aeroacoustic scattering from complex configurations.

The same compactness that permits the accuracy to be insensitive to mesh smoothness makes the method well suited for implementation in parallel machines. Biswas, Devine, and Flaherty [9] applied a third-order quadrature-based discontinuous method to a scalar wave equation on a NCUBE/2 hypercube platform. They reported a 97.57% parallel efficiency on 256 processors.

In this work, the discontinuous code developed by Atkins has been ported to several parallel platforms using MPI. The code solves the unsteady linear Euler equations on an unstructured mesh in two dimensions. A detailed description of the numerical algorithm can be found in reference [6]; however, the code structure has not been previously described. The first section of this article provides a brief description of the numerical method followed by a more detailed description of the code structure and the object-oriented design of the code. The second section describes the parallelization strategy and the modifications to the code to implement that strategy. The third section presents performance results of the code on the Origin2000 and several other computing platforms.

2. Discontinuous Galerkin Method. The discontinuous Galerkin method can be applied to an equation in conservation form

$$(2.1) \qquad \frac{\partial U}{\partial t} + \nabla \cdot \vec{F}(U) = 0$$

defined on a domain Ω with appropriate boundary conditions. The domain is divided into smaller, nonoverlapping elements Ω_i that cover the domain, i.e., $\Omega = \cup \Omega_i$. The discontinuous Galerkin method is obtained by applying a traditional Galerkin method to each element [10]. That is, a finite-dimensional basis set is selected for each element, the solution in each element is approximated in terms of an expansion in that basis set, and the governing equations are then projected on each member of the basis set and cast in a weak form

$$(2.2) \qquad \int_{\Omega_i} b_k \frac{\partial V_i}{\partial t} \, d\Omega - \int_{\Omega_i} \nabla b_k \cdot \vec{F}(V_i) \, d\Omega + \int_{\partial \Omega_i} b_k \vec{F}^R(\overline{V_i}, \overline{V_j}) \cdot \vec{n_i} ds = 0$$

where $\vec{n_i}$ in an outward-pointing, surface-element normal, $B \equiv \{ b_k, \quad 1 \leq k \leq N(p, d) \}$ denotes the basis set of degree p in d space dimensions,

$$(2.3) \qquad V_i \equiv \sum_{l=1}^{N(p,d)} v_{i,l} b_l \approx U_{\Omega_i}$$

is the approximate solution in element Ω_i, V_j denotes the approximate solution in a set of neighboring elements $\{\Omega_j\}$, and $\overline{V_i}$ and $\overline{V_j}$ denote the trace of the solution on a segment of the element boundary. The

coefficients of the expansion $v_{i,l}$ become the new unknown variables. In the fully discrete approach [11], the basis set contains both temporal and spatial functions. In the semi-discrete approach, which is used here, the basis set contains only spatial functions, and the solution expansion coefficients are functions of time (i.e., $v(t)_{i,l}$.)

Because each element has its own approximate solution, the global solution is discontinuous at the edge between elements. This discontinuity is resolved through the use of an approximate Riemann flux vector $\vec{F}^R(\overline{V_i}, \overline{V_j})$. The approximate Riemann flux provides the crucial coupling between elements, as well as the correct upwind bias that is required to ensure stability. It is, in fact, the only means by which neighboring elements communicate. Thus, each element communicates only with its nearest neighbors regardless of the order of the method. In the present work, the Riemann flux is approximated by a simple Lax-Fredrichs type flux of the form

$$(2.4) \qquad \vec{F}^R(\overline{V_i}, \overline{V_j}) \cdot \vec{n_i} \; = \; \left\{ \left[\vec{F}(\overline{V_i}) + \vec{F}(\overline{V_j})\right] \cdot \vec{n_i} \; - \; \lambda\left(\overline{V_j} - \overline{V_i}\right) \right\}/2$$

where λ is greater than the maximum of the absolute of the eigenvalues of $\left[\vec{F}(\overline{V_i}) + \vec{F}(\overline{V_j})\right]/2$.

When the basis functions are polynomials of degree p, the order of accuracy of the method has been rigorously proven to be at least $p + \frac{1}{2}$ [1]. In practice, however, a polynomial basis of degree p will usually give an error that converges at a rate of h^{p+1} where h is some average mesh size. In this work the term "r-order method" is used to refer to a method with a polynomial basis of degree $r - 1$.

In the quadrature-free formulation, developed by Atkins and Shu in [6], the flux vector $\vec{F}$ is approximated in terms of the basis set. The approximate Riemann flux is approximated in terms of a lower dimensional basis set $\bar{b}_l$ associated with the segment of the element boundary that is common to Ω_i and Ω_j

$$\vec{F}(V_i) \; \approx \; \sum_{l=1}^{M} \vec{f}_{i,l}\, b_l \qquad \vec{F}(\overline{V_i}) \cdot \vec{n_i} \; \approx \; \sum_{l=1}^{M} \overline{f}_{i,l}\, \bar{b}_l \qquad M \; \geq \; N(p,d)$$

and

$$\vec{F}^R(\overline{V_i}, \overline{V_j}) \cdot \vec{n_i} \; \equiv \; \sum_{l=1}^{M} \overline{f}_{i,l}^R\, \bar{b}_l \; = \; \frac{1}{2}\sum_{l=1}^{M} \left[\overline{f}_{i,l} + \overline{f}_{j,l} \; - \; \lambda\left(\bar{v}_j \; - \; \bar{v}_i\right)\right] \bar{b}_l$$

In the linear case, the expansion is exact with $M = N$; however, in the nonlinear case, the degree of the flux expansion may be higher than that of the solution. With these approximations, the volume and boundary integrals can be evaluated exactly, instead of by quadrature, leading to a simple implementation.

$$(2.5) \qquad \frac{\partial \mathbf{V}}{\partial t} \; = \; \left[(\mathbf{M}^{-1}\mathbf{A}) \cdot \mathbf{F} \; - \; \sum_{\{j\}} (\mathbf{M}^{-1}\mathbf{B}_j)\, \bar{\mathbf{F}}_j^R \right]$$

where

$$\mathbf{M} \equiv \left[\int_{\Omega} b_k\, b_l d\Omega\right], \qquad \mathbf{A} \equiv \left[\int_{\Omega} \nabla b_k\, b_l d\Omega\right], \qquad \mathbf{B}_j \equiv \left[\int_{\partial\Omega} b_k\, \bar{b}_l ds\right],$$

$$\mathbf{V} \equiv [v_{i,l}], \qquad \mathbf{F} \equiv \left[\vec{f}_{i,l}\right], \qquad \text{and} \qquad \bar{\mathbf{F}}^R \equiv \left[\overline{f}_{i,l}^R\right].$$

and $\overline{\partial\Omega}$ denotes the portion of $\partial\Omega$ common to elements Ω_i and Ω_j.

3

By performing the integrations on similarity elements, such as a unit triangle or square, the equations can be recast as.

$$(2.6) \qquad \frac{\partial \mathbf{V}}{\partial t} = J_i^{-1} \left[(\mathbf{M}^{-1}\mathbf{A}) \cdot J_i \mathbf{J}_i^{-1}\mathbf{F} - \sum_{\{j\}} (\mathbf{M}^{-1}\mathbf{B}_j) \, J_i \mathbf{J}_i^{-1} \bar{\mathbf{F}}_j^R \right]$$

where $\mathbf{J}_i$ denotes the Jacobian of the coordinate transformation from the similarity element to Ω_i, and $J_i = |\mathbf{J}_i|$. The matrices $\mathbf{M}^{-1}\mathbf{A}$ and $\mathbf{M}^{-1}\mathbf{B}_j$ are constant matrices that apply to all elements of a given type, and can be evaluated easily and exactly for any type of element shape. Hence they can be precomputed at a considerable savings. The details of the derivation can be found in [6].

3. Code Structure, Data Structures, and Object Model. The design of the program was motivated by the desire to maximize the advantages offered by the discontinuous Galerkin method while avoiding deficiencies that are common to traditional methods for unstructured grids. This motivation was, of course, in addition to the usual interest in efficiency, code reuse, and code maintenance.

Traditional flow solvers for unstructured meshes usually require more storage and have a slower computational rate than methods developed for structured grids. The extra storage arises from the pointers that are required to identify nearest neighbors, and sometimes, second nearest neighbors. The slower computational rate results from the gather-scatter type operations that occur at nearly every step of the algorithm. Unlike traditional finite-difference methods, the discontinuous Galerkin method has a large amount of data within each element and a large amount of work that is local to the element. By blocking the data by element and by a segregation of the methods according to whether or not gather-scatter operations are required, the usual weaknesses of methods for unstructured grids are eliminated. As will be seen later, these techniques also lead to a code structure that is easily and efficiently ported to parallel platforms.

The residual evaluation (the right-hand side of equation (2.6)) can be decomposed into a few fundamental operations. Figure 3.1 shows these operations and the flow of data between these steps. Each operation has been grouped according to whether or not gather-scatter is required. The Element group contains all the data and operations that are completely local to the individual element. These operations can be processed in any order. The Edge group contains operations that require gather-scatter type operations and inter-element communication. The operations of the Element group are further divided into those that depend only on the geometry of the element and those that depend only on the particular governing equations.

These three groups naturally lead to the adoption of three primary base classes: `Element`, `Edge`, and `Physics`. All three base classes are virtual and the `Element` and `Physics` classes are pure virtual. Specific element shapes (e.g., square, triangle, tetrahedron, etc.) and governing equations (e.g., scalar advection, Euler, etc.) are implemented in subclasses as illustrated in Figure 3.2. Each `Element` object contains a list of elements of a similar type (i.e., same shape, basis set, etc.) which eliminates the overhead of runtime dynamic binding. The solution data within the `Element` object is blocked by element. Because the collection of elements within a given object are of the same type, the size of structure of the data blocks are constant. The `Element` and `Physics` classes share data and are tightly coupled.

The `Edge` class has the task of evaluating $\bar{\mathbf{F}}_j^R$ on each edge from data in the elements on either side. An `Edge` object performs this task for a list of similar edges. The elements on either side of an edge are arbitrarily designated as being on the left or right sides of the edge. The `Edge` object does not contain any solution data. Instead, the `Edge` object contains only two pointers for each edge that points directly to a block of edge data within an element object. The `Edge` object accesses the data for $[\bar{V}, \bar{F}]_{left}$, and $[\bar{V}, \bar{F}]_{right}$, computes the approximate Riemann flux, and stores the result back in the `Element` object in the space allocated for
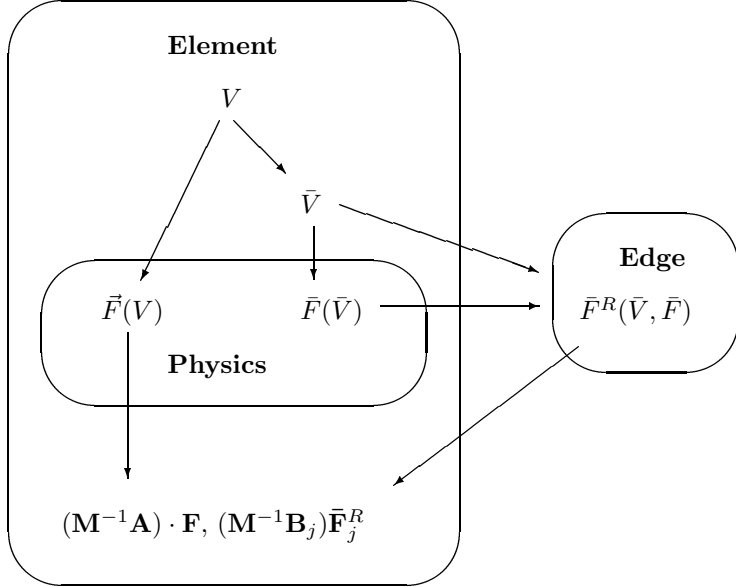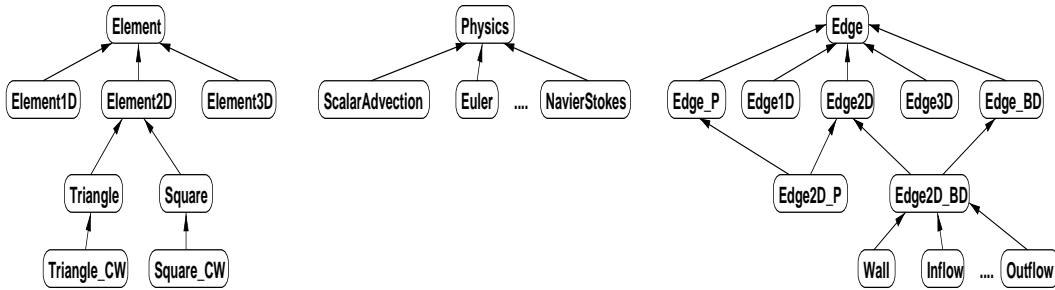
FIG. 3.1. *Computational groupings.*

FIG. 3.2. *Class hierarchies for the object model.*

$\bar{F}_{left}$ and $\bar{F}_{right}$. The base `Edge` class has a generic method for evaluating the approximate Riemann flux; however, this method is also overloaded in specialized subclasses to optimize the method for the number of spatial dimensions, or to treat cases in which the `Physics` of the left and right elements are different.

Boundary conditions are implemented as a special type of subclass in which an element exists on only one side of the edge (the left side by convention). Any boundary condition can be imposed either by supplying a special version of the approximate Riemann flux, or by supplying solution data for the side where the element is missing. The boundary edge class `Edge_BD` is a pure virtual class that sets up and initializes the additional data needed to impose most boundary conditions. New boundary conditions are easily implemented simply by creating a new derived class that supplies the required data or evaluates the flux in the desired manner.

A particular problem is represented by lists of pointers to `Element` objects and `Edge` objects so that elements of different types can be readily mixed. The first object in the `Edge` list usually contains all the interior edges, and the remaining objects support boundary conditions.

4. Parallel Design Consideration. The parallelization using a domain decomposition approach was easily implemented by treating the partition edges as a special boundary condition. The `Edge_P` class provides

Process A   Process B

V V

$\bar{V}$ $\bar{V}$

$\bar{F}(\bar{V})$ $\bar{F}(\bar{V})$

post Send/Recv   post Send/Recv

$\vec{F}(V)$ $\vec{F}(V)$

$(\mathbf{M}^{-1}\mathbf{A})\cdot\mathbf{F}$ $(\mathbf{M}^{-1}\mathbf{A})\cdot\mathbf{F}$

$(\mathbf{M}^{-1}\mathbf{B}_j)\bar{\mathbf{F}}^R_{\text{int.}}$ $(\mathbf{M}^{-1}\mathbf{B}_j)\bar{\mathbf{F}}^R_{\text{int.}}$

wait Send/Recv   wait Send/Recv

$(\mathbf{M}^{-1}\mathbf{B}_j)\bar{\mathbf{F}}^R_{\text{part.}}$ $(\mathbf{M}^{-1}\mathbf{B}_j)\bar{\mathbf{F}}^R_{\text{part.}}$

interior edges $\bar{F}^R(\bar{V},\bar{F})$

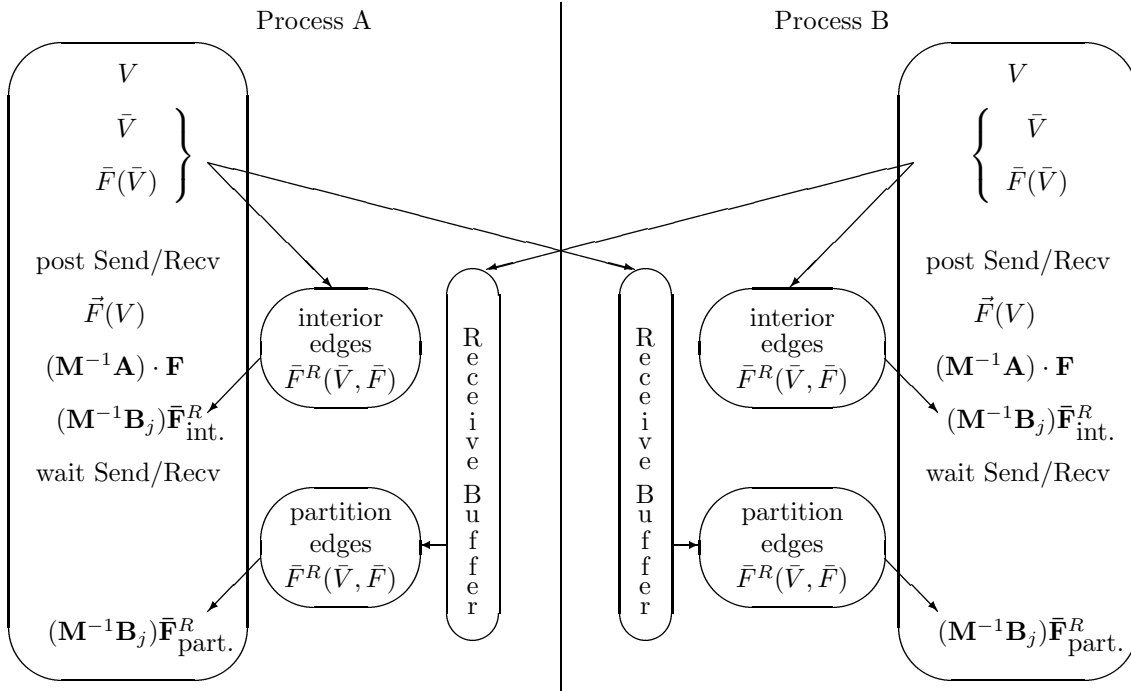partition edges $\bar{F}^R(\bar{V},\bar{F})$

Receive Buffer

FIG. 4.1. *Sequence of operations in the parallel residual evaluation.*

storage for send and receive buffers and methods to initialize and manage the new data. The `Edge_P` class has only three new methods, `InitPids()`, `BeginSendRecv()`, and `EndSendRecv()`, and overloads two methods of the base `Edge` class. The method `InitPids()` initializes the data that describes the structure of the send and receive buffers (i.e. how many neighboring partitions, who are they, and what part of the send buffer goes to each). The method `BeginSendRecv()` collects data from elements on the left side of a partition edge in the send buffer and posts the sends. This method also posts the receives using asynchronous message passing which enables communication/computation overlap, and prevents deadlock due to the large buffer size. The method `EndSendRecv()` provides a barrier that ensures the synchronization required by the time accurate calculation. The base `Edge` class methods that allocate data and initialize pointers are overloaded by the `Edge_P` class methods. The pointers that would normally point to the data in the element on the right side of the edge are now initialized to point to a block of data in the receive buffer. The actual flux computation is inherited from the base `Edge` class, and all code written for the `Element`, `Physics`, and `Edge` classes remains unchanged. The `Edge_P` class contains only about 120 lines of code, out of approximately 20K lines of C++ user code for the overall application, exclusive of linked libraries.

The computation was reordered to maximize the overlap of communication and computation as shown in Figure 4.1. First, the edge solution and flux ($\bar{V}$ and $\bar{F}$) are computed, the send buffer is loaded, and the sends and receives are posted. While the communication is occuring, the volume flux $\vec{F}$ is computed for all elements, the approximate Riemann flux is computed for all interior edges, all boundary conditions are applied, and the contribution of the volume flux to the residual is evaluated. Finally, the contribution of $\bar{F}^R$ is computed and the solution is updated after all communication has been completed.

Another important aspect of the parallelization task is the domain decomposition. The original code defined an initial grid structure that completely described the coordinates, element connectivity, and boundary conditions of the problem. In the parallel version, each processor reads or creates a structure that defines its portion of the domain. The remainder of the initialization process proceeds as in the original code. In an

earlier version, the domain was decomposed using the Parallel Mesh Environment (PME) software developed by Özturan [12]; however, the present version makes use of the PARMETIS [13] software for the domain decomposition. Each processor reads or creates the global grid structure, generates the input for PARMETIS, and creates the grid structure for its partition of the domain. Currently the partitioning method adds about 455 lines of code.

5. Benchmark Problem. The parallel code is used to solve problems from the Second Benchmark Problems in Computational Aeroacoustics Workshop [14] held in 1997. The physical problem is to find the sound field generated by a propeller scattered off by the fuselage of an aircraft. The fuselage is idealized as a circular cylinder and the noise source (propeller) is modeled as a line source so that the computational problem is two-dimensional. The linearized Euler equations are in the form of equation (2.1) where

$$
\mathbf{U} = \begin{bmatrix} \rho \\ p \\ u \\ v \end{bmatrix} \quad \text{and} \quad \vec{\mathbf{F}} = \begin{bmatrix} M_x\rho & M_y\rho \\ M_xp + u & M_yp + v \\ M_xu + p & M_yu \\ M_xv & M_yv + p \end{bmatrix}
$$

For the test problem, $M_x = M_y = 0$ and the initial conditions are

$$
\mathbf{U}(\cdot, 0) = \begin{bmatrix} 1 \\ \exp\left[(\ln 2)\left(\frac{(x-4)^2 + y^2}{0.2^2}\right)\right] \\ 0 \\ 0 \end{bmatrix}
$$

The boundary conditions consist of: a zero-normal velocity at the surface of the cylinder, i.e., $\mathbf{v} \cdot \mathbf{n} = 0$; and a radiation boundary condition for $x, y \to \infty$. The problem is to find the pressure $p(t)$ at the three points $A(r = 5, \theta = 90)$, $B(r = 5, \theta = 135)$, and $C(r = 5, \theta = 180)$. Figures 5.1 and 5.2 show a typical partitioned mesh used on the following performance test and the corresponding time history of the pressure at point A.
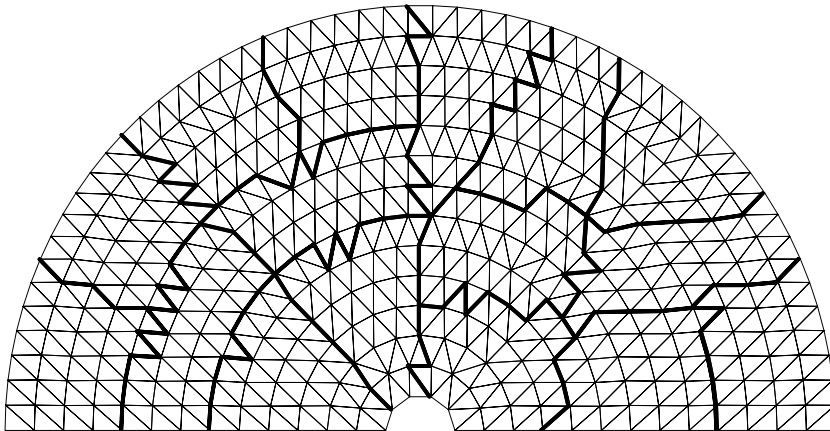

FIG. 5.1. *Partitioned mesh for the solved problem*

6. Results and Discussion. Performance tests have been conducted on the SGI Origin2000 and IBM SP2 platforms, and on clusters of workstations. The first test case applied a third-order method on a coarse
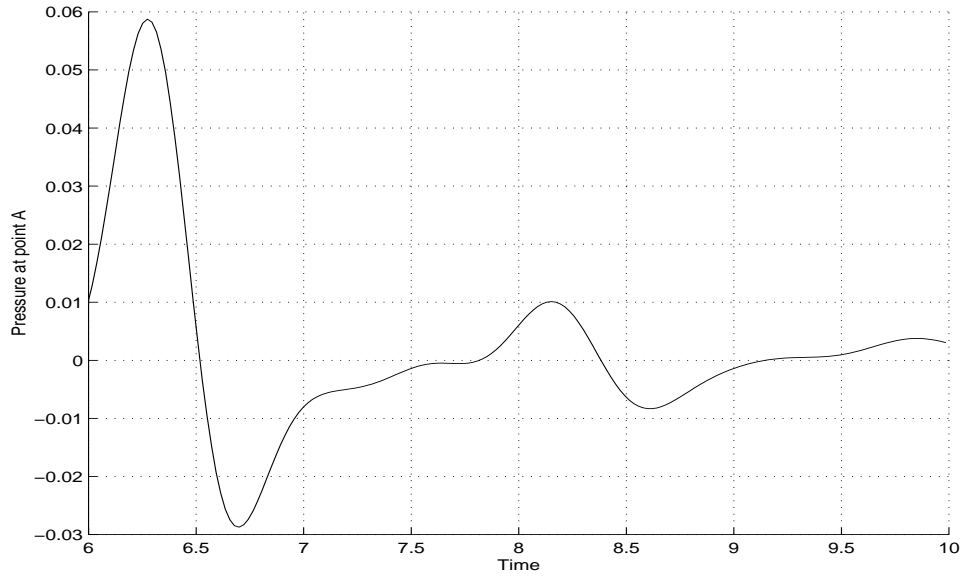
Fig. 5.2. *Pressure at point A*

mesh of only 800 elements. Near linear speedup is obtained on both machines (Figure 6.1); however, the partition size becomes small on more than 8 or 10 processors and performance begins to drop off noticeably. This small problem was also run on two clusters of, resp., SGI and Sun workstations in an FDDI network
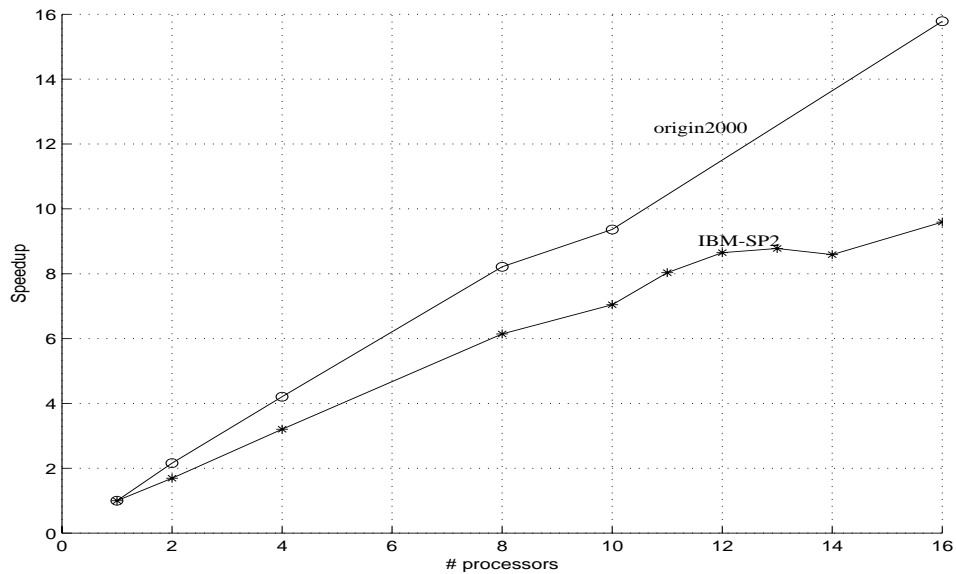

Fig. 6.1. *Performance on the Origin2000 and SP2 for a problem with 800 third-order elements.*

(shown in Table 6.1). The two clusters consisted of similar but not identical hardware. The network was not dedicated to the cluster but carried other traffic. All timings are reported in seconds.

Two larger problems were used to evaluate the code on the Origin2000. For these cases a fifth-order method was used and the problem size was further increased by decreasing the element size and by varying the location of the outer boundary. Tables 6.2 and 6.3 present detailed statistics about the mesh, per

Small fixed problem size, various platforms						
DOF = 24,000		# edges = 1,176		# vertices= 421		
# Processors	SP2		SGI		SUN	
	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
1	378	1.00	311	1.00	316	1.00
2	197	1.92	156	2.03	160	1.97
4	102	3.70	93	3.34	89	3.55
8	53	7.13	58	5.36	103	3.06
16	31	12.2				
32	23	16.4				

TABLE 6.1

Performance on SP2 and workstation clusters for problem with 800 third-order elements.

Intermediate fixed problem size, Origin2000			
DOF = 1,366,560	# edges = 34,393		# vertices= 11,690
# Processors	$\lfloor$# Elements/Processor$\rfloor$	Speedup	Rate (10^6)
1	22,776	1.000	6.35903
2	11,388	1.670	7.60002
4	5,694	3.409	7.45281
8	2,847	7.856	6.44211
10	2,277	8.680	7.28149
12	1,898	13.811	5.49986
16	1,423	21.275	4.76637
32	711	44.748	4.38155
48	474	64.288	4.92951
64	355	68.157	4.73365

TABLE 6.2

Performance on Origin2000 for problem with 22,776 fifth-order elements.

processor load, speedup, and computational rate. *The computational rate* is defined as the maximum wall clock time of any processor divided by the number of degrees of freedom per processor. All processors have essentially the same wall clock time because the time accurate calculation is synchronized at each stage of the Runge-Kutta. In this larger problem, superlinear speedup is obtained as the number of elements per processor is decreased. This result is due to the improvement in cache performance that occurs when a fixed problem is divided into smaller parts as the number of processors is increased, and workingsets become cache-resident.

7. Conclusions. An object-oriented computational aeroacoustics code was ported to several distributed memory parallel platforms using MPI. The port was achieved with only a few changes to the existing code and the code performance obtained was excellent. The discontinuous Galerkin method provides a significant amount of computational work that is local to each element that can be effectively used to hide the communication overhead. Strategies employed in the object-oriented design of the serial code to avoid inherent problems in methods for unstructured grids also facilitated the parallel implementation. The use of object

Large fixed problem size, Origin2000			
DOF = 2,423,520	# edges = 60,891		# vertices= 20,572
# Processors	$\lfloor$# Elements/Processor$\rfloor$	Speedup	Rate (10^6)
1	40,392	1.000	6.26362
2	20,196	1.663	7.52042
4	10,098	3.306	7.56411
8	5,049	6.568	7.57382
12	3,366	10.795	6.93328
16	2,524	15.756	6.23546
32	1,262	42.573	4.59007
64	631	69.686	5.44298

TABLE 6.3

Performance on Origin2000 for problem with 40,392 fifth-order elements.

oriented features, such as virtual functions, also greatly facilitated the port. The use of virtual functions, which require dynamic binding, did not hurt performance because it was applied at a coarse grain level.

REFERENCES

[1] C. JOHNSON AND J. PITKÄRATA, *An Analysis of the Discontinuous Galerkin Method for a Scalar Hyperbolic Equation*, Mathematics of Computation, 46 (1986), pp. 1–26.

[2] B. COCKBURN AND C.-W. SHU, *TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework*, Mathematics of Computation, 52 , No. 186 (1989), pp. 411–435.

[3] B. COCKBURN, S. Y. LIN, AND C.-W. SHU, *TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws III: One Dimensional Systems*, Journal of Computational Physics, 84, No. 1 (1989), pp. 90–113.

[4] B. COCKBURN, S. HOU, AND C. W. SHU, *The Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case*, Mathematics of Computation, 54, No. 190 (1990), pp. 545–581.

[5] G. JIANG AND C.-W. SHU, *On Cell Entropy Inequality for Discontinuous Galerkin Methods*, Mathematics of Computation, 62, No. 206 (1994), pp. 531–538.

[6] H. L. ATKINS AND C. W. SHU, *Quadrature-Free Implementation of Discontinuous Galerkin Method for Hyperbolic Equations*, AIAA Journal, 36 (1998), pp. 775–782.

[7] H. L. ATKINS, *Continued Development of the Discontinuous Galerkin Method for Computational Aeroacoustic Applications*, AIAA Paper 97-1581, May 1997.

[8] ——, *Local Analysis of Shock Capturing Using Discontinuous Galerkin Methodology*, AIAA Paper 97-2032, June 1997.

[9] R. BISWAS, K. D. DEVINE, AND J. FLAHERTY, *Parallel, Adaptive Finite Element Methods for Conservation Laws*, Applied Numerical Mathematics, 14, No. 1-3 (1994), pp. 225–283.

[10] C. A. J. FLETCHER, *Computational Galerkin Methods*, Springer-Verlag, New York, 1984.

[11] R. B. LOWRIE, P. L. ROE, AND B. VAN LEER, *A Space-Time Discontinuous Galerkin Method for the*

 Time-Accurate Numerical Solution of Hyperbolic Conservation Laws, AIAA Paper 95-1658, June 1995.

[12] C. ÖZTURAN, *Draft-Parallel Mesh Environment User's Manual*, to be submitted as an ICASE report.

[13] G. KARYPIS AND V. KUMAR, *Parallel multilevel k-way partitioning scheme for irregular graphs*, Technical Report 96-036, Dept. of Computer Science, University of Minnesota, 1996.

[14] NASA Conference Publication 3352, *Second Computational Aeroacoustics (CAA) Workshop on Benchmark Problems*, Proceedings of a workshop sponsored by NASA, June 1997.