

# PARALLEL IMPLEMENTATION OF THE DISCONTINUOUS GALERKIN METHOD \*

ABDELKADER BAGGAG <sup>†</sup>, HAROLD ATKINS <sup>‡</sup>, AND DAVID KEYES <sup>§</sup>

## Abstract.

This paper describes a parallel implementation of the discontinuous Galerkin method. Discontinuous Galerkin is a spatially compact method that retains its accuracy and robustness on non-smooth unstructured grids and is well suited for time dependent simulations. Several parallelization approaches are studied and evaluated. The most natural and symmetric of the approaches has been implemented in an object-oriented code used to simulate aeroacoustic scattering. The parallel implementation is MPI-based and has been tested on various parallel platforms such as the SGI Origin, IBM SP2, and clusters of SGI and Sun workstations. The scalability results presented for the SGI Origin show slightly superlinear speedup on a fixed-size problem due to cache effects.

**Key words.** Discontinuous Galerkin Method, Parallelization strategies, Object Oriented, Unstructured Grids, Euler Equations, High-Order Accuracy, Superlinear Speedup

**Subject classification.** Computer Science

**1. Motivation.** The discontinuous Galerkin (DG) method is a robust and compact finite element projection method that provides a practical framework for the development of high-order accurate methods using unstructured grids. The method is well suited for large-scale time-dependent computations in which high accuracy is required. An important distinction between the DG method and the usual finite-element method is that in the DG method the resulting equations are local to the generating element. The solution within each element is not reconstructed by looking to neighboring elements. Thus, each element may be thought of as a separate entity that merely needs to obtain some boundary data from its neighbors. The compact form of the DG method makes it well suited for parallel computer platforms. This compactness also allows a heterogeneous treatment of problems. That is, the element topology, the degree of approximation and even the choice of governing equations can vary from element to element and in time over the course of a calculation without loss of rigor in the method.

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. The DG method has been shown in mesh refinement studies [6] to be insensitive to the smoothness of the mesh. Its compact formulation can be applied near boundaries without special treatment, which greatly increases the robustness and accuracy of any boundary condition implementation. These features are crucial for the robust treatment of complex geometries. In semi-discrete form, the DG method

---

\*This research was supported by the National Aeronautics and Space Administration under NASA contract No. NAS1-97046 while Baggag and Keyes were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

<sup>†</sup>Department of Computer Sciences, Purdue University, 1398 Computer Science Building, West-Lafayette, IN 47907-1398, (email: [baggag@cs.purdue.edu](mailto:baggag@cs.purdue.edu)).

<sup>‡</sup>Computational Modeling and Simulation Branch, NASA Langley Research Center, Hampton, VA 23681-2199, (email: [h.l.atkins@larc.nasa.gov](mailto:h.l.atkins@larc.nasa.gov)).

<sup>§</sup>Department of Mathematics & Statistics, Old Dominion University, Norfolk, VA 23529-0162, ISCR, Lawrence Livermore National Laboratory, Livermore, CA 94551-9989, and ICASE, NASA Langley Research Center, Hampton, VA 23681-2199, (email: [keyes@k2.llnl.gov](mailto:keyes@k2.llnl.gov)).

can be combined with explicit time-marching methods, such as Runge-Kutta. One of the disadvantages of the method is its high storage and high computational requirements; however, a recently developed quadrature-free implementation [6] has greatly ameliorated these concerns.

Parallel implementations of the DG method have been performed by other investigators. Biswas, Devine, and Flaherty [7] applied a third-order quadrature-based DG method to a scalar wave equation on a NCUBE/2 hypercube platform and reported a 97.57% parallel efficiency on 256 processors. Bey et al. [8] implemented a parallel *hp*-adaptive DG method for hyperbolic conservation laws on structured grids. They obtained nearly optimal speedups when the ratio of interior elements to subdomain interface elements is sufficiently large. In both works, the grids were of a Cartesian type with cell sub-division in the latter case.

The quadrature-free form of the DG method has been previously implemented and validated [6, 9, 10] in an object-oriented code for the prediction of aeroacoustic scattering from complex configurations. The code solves the unsteady linear Euler equations on a general unstructured mesh of mixed elements (squares and triangles) in two dimensions. The DG code developed by Atkins has been ported [11] to several parallel platforms using MPI. A detailed description of the numerical algorithm can be found in reference [6]; and the description of the code structure, parallelization routines and model objects can be found in reference [11]. In this work, three different parallelization approaches are described and efficiency results for the selected approach are reported.

The next section provides a brief description of the numerical method and is followed by a discussion of parallelization strategies, a citation of our standard test case, and performance results of the code on the Origin2000 and several other computing platforms.

**2. Discontinuous Galerkin Method.** The DG method is readily applied to any equation of the form

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

on a domain that has been divided into arbitrarily shaped nonoverlapping elements  $\Omega_i$  that cover the domain. The DG method is defined by choosing a set of local basis functions  $B = \{b_l, 1 \leq l \leq N(p, d)\}$  for each element, where  $N$  is a function of the local polynomial order  $p$  and the number of space dimensions  $d$ , and approximating the solution in the element in terms of the basis set

$$(2.2) \quad U_{\Omega_i} \approx V_i \equiv \sum_{l=1}^{N(p,d)} v_{i,l} b_l.$$

The governing equation is projected onto each member of the basis set and cast in a weak form to give

$$(2.3) \quad \int_{\Omega_i} b_k \frac{\partial V_i}{\partial t} d\Omega - \int_{\Omega_i} \nabla b_k \cdot \vec{F}(V_i) d\Omega + \int_{\partial\Omega_{ij}} b_k \vec{F}^R(\overline{V}_i, \overline{V}_j) \cdot \vec{n}_{ij} ds = 0,$$

where  $V_i$  is the approximate solution in element  $\Omega_i$ ,  $V_j$  denotes the approximate solution in a neighboring element  $\Omega_j$ ,  $\partial\Omega_{ij}$  is the segment of the element boundary that is common to the neighboring element  $\Omega_j$ ,  $\vec{n}_{ij}$  is the unit outward-normal vector on  $\partial\Omega_{ij}$ , and  $\overline{V}_i$  and  $\overline{V}_j$  denote the trace of the solutions on  $\partial\Omega_{ij}$ . The coefficients of the approximate solution  $v_{i,l}$  are the new unknowns, and the local integral projection generates a set of equations governing these unknowns. The trace quantities are expressed in terms of a lower dimensional basis set  $\overline{b}_l$  associated with  $\partial\Omega_{ij}$ .  $\vec{F}^R$  denotes a numerical flux which is usually an approximate Riemann flux of the Lax-Friedrichs type.

Because each element has a distinct local approximate solution, the solution on each interior edge is double valued and discontinuous. The approximate Riemann flux  $\vec{F}^R(\overline{V}_i, \overline{V}_j)$  resolves the discontinuity and

provides the only mechanism by which adjacent elements communicate. The fact that this communication occurs in an edge integral means the solution in a given element  $V_i$  depends only on the edge trace of the neighboring solution  $\bar{V}_j$ , not on the whole of the neighboring solution  $V_j$ . Also, because the approximate solution within each element is stored as a function, the edge trace of the solution is obtained without additional approximations.

The DG method is efficiently implemented on general unstructured grids to any order of accuracy using the quadrature-free formulation. 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  $b_l$ , and the approximate Riemann flux  $\vec{F}^R$  is approximated in terms of the lower basis set  $\bar{b}_l$ :

$$(2.4) \quad \vec{F}(V_i) \approx \sum_{l=1}^{N(p,d)} \vec{f}_{i,l} b_l, \quad \vec{F}^R(\bar{V}_i, \bar{V}_j) \cdot \vec{n}_i \equiv \sum_{l=1}^{N(p,d-1)} \vec{f}_{ij,l}^R \bar{b}_l.$$

With these approximations, the volume and boundary integrals can be evaluated analytically, instead of by quadrature, leading to a simple sequence of matrix-vector operations

$$(2.5) \quad \frac{\partial [v_{i,l}]}{\partial t} = (\mathbf{M}^{-1} \mathbf{A}) [f_{i,l}] - \sum_{\{j\}} (\mathbf{M}^{-1} \mathbf{B}_{ij}) [\vec{f}_{ij,l}^R],$$

where

$$(2.6) \quad \mathbf{M} \equiv \left[ \int_{\Omega} b_k b_l d\Omega \right], \quad \mathbf{A} \equiv \left[ \int_{\Omega} \nabla b_k b_l d\Omega \right], \quad \mathbf{B}_{ij} \equiv \left[ \int_{\partial\Omega_{ij}} b_k \bar{b}_l ds \right].$$

The matrices  $\mathbf{M}$ ,  $\mathbf{A}$ , and  $\mathbf{B}_{ij}$  depend only on the shape of the similarity element and the degree of the solution  $p$ . Thus, the set of matrices associated with a particular similarity element can be precomputed and applied to all elements that map to it at a considerable savings of both storage and computation. The residual of equation (2.5) is evaluated by the following sequence of operations:

$$\left. \begin{aligned} \left. \begin{aligned} [\bar{v}_{ij,l}] &= \mathbf{T}_{ij} [v_{i,l}] \\ [\vec{f}_{ij,l}] &= \vec{F}(\bar{V}_j) \cdot \vec{n}_{ij} \end{aligned} \right\} \quad \forall \Omega_i, \\ [\vec{f}_{ij,l}^R] &= F^R(\bar{V}_i, \bar{V}_j) \quad \forall \partial\Omega_{ij}, \\ \left. \begin{aligned} [f_{i,l}] &= \vec{F}(V_i) \\ \frac{\partial [v_{i,l}]}{\partial t} &= (\mathbf{M}^{-1} \mathbf{A}) [f_{i,l}] - \sum_{\{j\}} (\mathbf{M}^{-1} \mathbf{B}_{ij}) [\vec{f}_{ij,l}^R] \end{aligned} \right\} \quad \forall \Omega_i, \end{aligned} \right.$$

where  $\mathbf{T}_{ij}$  is the trace operator, and  $[\bar{(\ )}_{ij,l}]$  denotes a vector containing the coefficients of an edge quantity on edge  $j$ . Edges will be referred to as interior edges or boundary edges when it is necessary to distinguish between the two.

**3. Parallel Computation.** In this section, three different possible parallelization strategies for the DG method are described. The first approach is symmetric and easy to implement but results in redundant flux calculations. The second and third approaches eliminate the redundant flux calculations; however, the communication occurs in two stages making it more difficult to overlap with computation, and increasing the complexity of the implementation. The following notation will be used to describe the parallel implementation. Let  $\Omega$  denote any element, let  $\partial\Omega_p$  denote any edge on the partition boundary, and  $\partial\Omega_I$  denote

any other edge. The first approach is symmetric and is easily implemented in the serial code. It can be summarized as follows:

1. Compute  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}] \quad \forall \Omega$
2. Send  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}]$  on  $\partial\Omega_p$  to neighboring partitions
3. Compute  $[f_l]$  and  $(\mathbf{M}^{-1}\mathbf{A})[f_l] \quad \forall \Omega$ , and  $[\bar{f}_{j,l}^R] \quad \forall \partial\Omega_I$
4. Receive  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}]$  on  $\partial\Omega_p$  from neighboring partitions
5. Compute  $[\bar{f}_{j,l}^R] \quad \forall \partial\Omega_p$  and  $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega$

In this approach, nearly all of the computation is scheduled to occur between the nonblocking send and receive; however, the edge flux  $[\bar{f}_{j,l}^R]$  is doubly computed on all partition edges  $\partial\Omega_p$ . It is observed in actual computations that redundant calculation is not a significant factor. The calculation of  $[\bar{f}_{ij,l}^R]$  on all edges,  $\partial\Omega_{ij}$ , represents only 2% to 3% of the total CPU time, and the redundant computation is performed on only a fraction of these edges.

The above sequence reflects the actual implementation [11] used to generate the results to be shown later; however, this approach offers the potential for further improvement. By collecting the elements into groups according to whether or not they are adjacent to a partition boundary, some of the work associated with the edge integral can also be performed between the send and receive. Let  $\Omega_p$  denote any element adjacent to a partition boundary and  $\Omega_I$  denote any other element. The following sequence provides maximal overlap of communication and computation.

1. Compute  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}] \quad \forall \Omega_p$
2. Send  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}]$  on  $\partial\Omega_p$  to neighboring partitions
3. Compute  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}] \quad \forall \Omega_I$
4. Compute  $[f_l]$  and  $(\mathbf{M}^{-1}\mathbf{A})[f_l] \quad \forall \Omega$ , and  $[\bar{f}_{j,l}^R] \quad \forall \partial\Omega_I$
5. Compute  $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega_I$
6. Receive  $[\bar{v}_{j,l}]$  and  $[\bar{f}_{j,l}]$  on  $\partial\Omega_p$  from neighboring partitions
7. Compute  $[\bar{f}_{j,l}^R] \quad \forall \partial\Omega_p$  and  $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega_p$

**3.1. Other Parallelization Strategies.** Two variations of an alternative parallelization strategy that eliminates the redundant flux calculations are described. In these approaches, the computation of the edge flux  $[\bar{f}_{j,l}^R]$  on a partition boundary is performed by one processor and the result is communicated to the neighboring processor. The processor that performs the flux calculation is said to “own” the edge. In the first variation, all edges shared by two processors are owned by only one of the two processors. In the second variation, ownership of the edges shared by two processors is divided equally between the two. Let  $\partial\Omega_p^{(a)}$  denote any edge owned by processor A,  $\partial\Omega_p^{(b)}$  denote any edge owned by adjacent processor B, and  $\partial\Omega_p^{(ab)}$  denote any edge shared by processors A and B. For the purpose of illustration, let ownership of all  $\partial\Omega_p^{(ab)}$  in the first variation be given to processor A. Thus  $\{\partial\Omega_p^{(a)}\} \cap \{\partial\Omega_p^{(b)}\} = \{\emptyset\}$  in both variations, and  $\{\partial\Omega_p^{(ab)}\} \cap \{\partial\Omega_p^{(b)}\} = \{\emptyset\}$  in the first variation. Both computations can be summarized in the following steps:

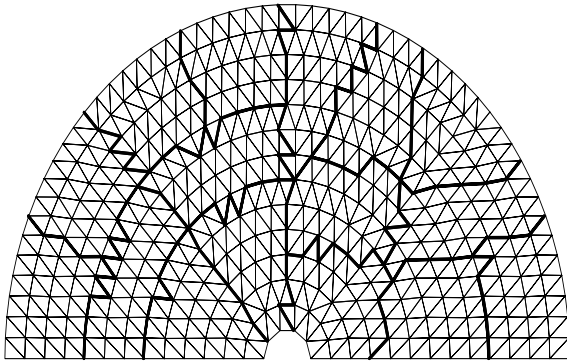
Process A

Process B

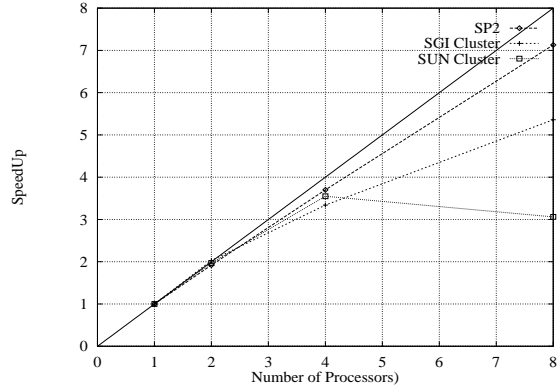
1. Compute $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \Omega_p$	Compute $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \Omega_p$
2. Send $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \partial \Omega_p \setminus \partial \Omega_p^{(a)}$	Send $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \partial \Omega_p \setminus \partial \Omega_p^{(b)}$
3. Compute $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \Omega_I$	Compute $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \Omega_I$
4. Compute $[f_i] \quad \forall \Omega$	Compute $[f_i] \quad \forall \Omega$
5. Receive $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \partial \Omega_p^{(a)}$	Receive $[\bar{v}_{j,l}]$ and $[\bar{f}_{j,l}] \quad \forall \partial \Omega_p^{(b)}$
6. Compute $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_p^{(a)}$	Compute $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_p^{(b)}$
7. Send $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_p^{(a)}$	Send $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_p^{(b)}$
8. Compute $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_I$	Compute $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_I$
9. Compute $(\mathbf{M}^{-1}\mathbf{A})[f_i] \quad \forall \Omega$	Compute $(\mathbf{M}^{-1}\mathbf{A})[f_i] \quad \forall \Omega$
10. Compute $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega_I$	Compute $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega_I$
11. Receive $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_p \setminus \partial \Omega_p^{(a)}$	Receive $[\bar{f}_{j,l}^R] \quad \forall \partial \Omega_p \setminus \partial \Omega_p^{(b)}$
12. Compute $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega_p$	Compute $(\mathbf{M}^{-1}\mathbf{B}_j)[\bar{f}_{j,l}^R] \quad \forall \Omega_p$

It is clear that under these strategies, there are no redundant flux calculations. In both variations, the total amount of data sent is actually less than that of the symmetric approach presented earlier. However, because the sends are performed in two stages, it is more difficult to overlap the communication with useful computation. Also, the unsymmetric form of edge ownership in the first variation introduces a difficulty in balancing the work associated with the edge flux calculation. In the second variation, the number of sends is twice that of the symmetric approach; however, because  $\{ \partial \Omega_p^{(ab)} \} \cap \{ \partial \Omega_p^{(b)} \} = \{ \emptyset \}$  in the first variation, the total number of sends in this approach is the same as in the symmetric approach presented earlier.

**4. Physical Problem.** The parallel code is used to solve problems from the Second Benchmark Problems in Computational Aeroacoustics Workshop [12] held at Florida State University in 1997. The physical problem is the scattering of acoustic waves and is well represented by the linearized Euler equations written in the form of equation (2.1). Details of the problem can be found in reference [12]. Figure (4.1.a) shows a typical partitioned mesh.



(a)



(b)

FIG. 4.1. Partitioned mesh (a), Performance on SP2 and workstations clusters (b)

**5. 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 mesh of only 800 elements. This small problem was run on the SP2 and two clusters of, resp., SGI and

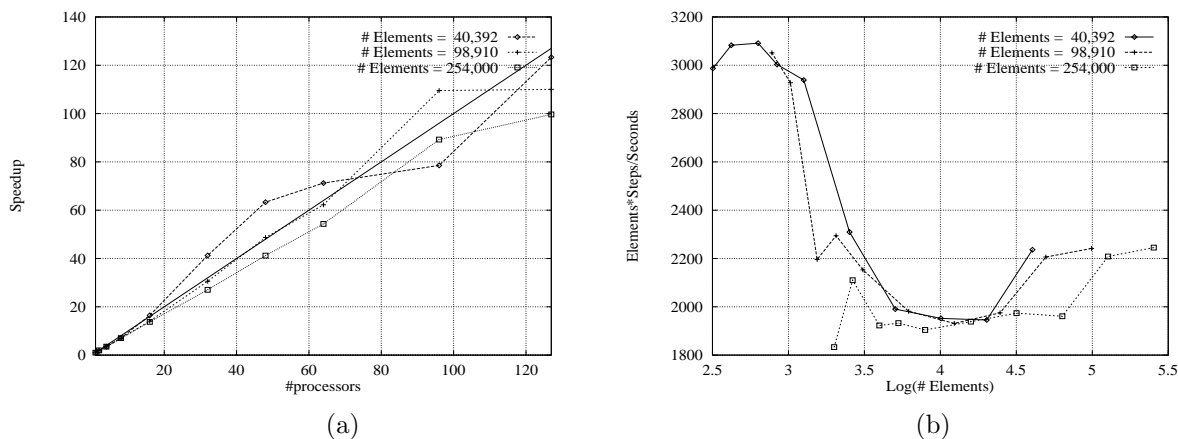


FIG. 5.1. Speedup for large domain on Origin 2000 (a), Computational Rate (b)

Sun workstations in an FDDI network. The two clusters consisted of similar but not identical hardware. Also, the network was not dedicated to the cluster but carried other traffic. As seen in Figure (4.1.b), near linear speedup is obtained on all three machines when the partition size is sufficiently large ( $> 100$  element/processor). As the domain is divided over more processors, the number of elements assigned to each processor becomes too small and the communication overhead becomes apparent. The performance figures are far below ideal; however, four to eight distributed workstations still provide a valuable performance improvement.

Three 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 controlled by varying the element size and by varying the location of the outer boundary. In the small and medium size problems, a slight superlinear speedup is obtained for some range of processors. 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. The larger problem does not fit in cache in 128 processors and no superlinear speedup is observed; however, it is expected that performance will improve as the number of processors is increased. This is supported by Figure (2.b) which shows the computational rate as a function of the number of elements on each processor. The *computational rate* is defined as the average number of elements per processor times number of time steps divided by the wall clock time. (Note: because the calculation is time accurate, all processors are synchronized at each stage of the Runge-Kutta, and thus, the wall clock time of all processors is essentially the same.) Both the small and medium size problems show a 50% increase in the computational rate at about 2000 elements elements per processor. The computational rates for all three problem sizes are similar indicating good scalability. Thus, computational rate may be a better indicator of scalability than the usual “speedup” measure.

**6. Conclusions.** Three parallelization strategies for the DG method have been described. A simple symmetric strategy introduces a redundant flux calculation, but maximizes the overlap of computation and communication. The redundant flux calculation can be eliminated by introducing a sequence of sends; however, with this approach it is more difficult to hide communication with computation, and it leads to a

complex implementation. In practice, the overhead due to the redundant flux calculation is negligible. The symmetric parallelization approach is implemented in an object-oriented computational aeroacoustics code using MPI. Performance results are presented for several distributed memory parallel platforms. The DG method provides a significant amount of computational work that is local to each element, this is due to the compact form of the method which can be effectively used to hide the communication overhead. The compact character of the method is exploited in the implementation. The parallel implementation gives superlinear speedup for large problems. This is attributed to the overlapping of computation as well as cache accelerations that occur when workingsets are cache resident.

## REFERENCES

- [1] C. Johnson, J. Pitkäranta, *An Analysis of the Discontinuous Galerkin Method for a Scalar Hyperbolic Equation*, Mathematics of Computation, 46, (1986), pp. 1–26.
- [2] B. Cockburn, C. W. Shu, *TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework*, Mathematics of Computation, Vol. 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, Vol. 84, No. 1, 1989, pp. 90–113.
- [4] B. Cockburn, S. Hou, C. W. Shu, *The Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws IV: The Multidimensional Case* Mathematics of Computation, Vol. 54, No. 190, 1990, pp. 545–581.
- [5] G. Jiang, C. W. Shu, *On Cell Entropy Inequality for Discontinuous Galerkin Methods*, Mathematics of Computation, Vol. 62, No. 206, 1994, pp. 531–538.
- [6] H. L. Atkins, C. W. Shu, *Quadrature-Free Implementation of Discontinuous Galerkin Method for Hyperbolic Equations*, AIAA Journal, 36 (1998), pp. 775–782.
- [7] R. Biswas, K. D. Devine, and J. Flaherty, *Parallel, Adaptive Finite Element Methods for Conservation Laws*, Applied Numerical Mathematics, Vol. 14, No. 1-3, 1994, pp. 225–283.
- [8] K. S. Bey, J. T. Oden, A. Patra, *A parallel hp-adaptive discontinuous Galerkin method for hyperbolic conservation laws*, Applied Numerical Mathematics, 20, 1996, pp. 321–336.
- [9] H. L. Atkins, *Continued Development of the Discontinuous Galerkin Method for Computational Aeroacoustic Applications*, AIAA Paper 97-1581, May 1997.
- [10] H. L. Atkins, *Local Analysis of Shock Capturing Using Discontinuous Galerkin Methodology*, AIAA Paper 97-2032, June 1997.
- [11] A. Baggag, H. Atkins, C. Özturan, and D. Keyes, *Parallelization of an Object-oriented Unstructured Aeroacoustics Solver*, ICASE Report N0. 99-11, Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, TX, March 22-24, 1999.
- [12] NASA Conference Publication 3352, *Second Computational Aeroacoustics Workshop on Benchmark Problems*, Proceedings of a workshop sponsored by NASA, June 1997.