AIAA JOURNAL
Vol. 30, No. 4, April 1992

Efficient Massively Parallel Euler Solver for Two-Dimensional Unstructured Grids

Steven W. Hammond* and Timothy J. Barth†
NASA Ames Research Center, Moffett Field, California 94035

We describe a data parallel mesh-vertex upwind finite-volume scheme for solving the Euler equations on triangular unstructured meshes. A novel vertex-based partitioning of the problem is introduced that minimizes the computation and communication costs associated with distributing the computation to the processors of a massively parallel computer. Finally, we compare the performance of this unstructured computation on 8K processors of the Connection Machine CM-2 with one processor of a Cray-YMP. Our experiments show that 8K processors of the CM-2 achieve 86% of the performance of one processor of the Cray-YMP on the unstructured mesh computations described here.

I. Introduction

We develop an upwind finite-volume flow solver for the Euler equations that is well-suited for massively parallel implementation. The mathematical formulation of this flow solver was proposed and implemented on the Cray-2 by Barth and Jespersen.[1] A novel partitioning of the problem is introduced that minimizes the computation and communication costs on a massively parallel computer. In a mesh-vertex scheme, solution variables are associated with each vertex of the mesh and flux computation is performed at edges of the nonoverlapping control volumes that surround each vertex. Implementations on sequential and vector computers typically perform this computation edgewise, i.e., programs loop over each edge of the control volume to perform flux calculations. The resulting flux calculation contributes to two control volumes that share the particular edge. To parallelize this operation, one could assign one edge to each processor (edge based). The problem with an edge-based partition is that the unknowns are associated with the vertices of the mesh. Thus, as the flux calculations are performed, each processor needs to gather information from the two processors holding incident vertex data and then distribute results.

We introduce a novel problem partitioning that assigns each *vertex* of the mesh to one processor of the computer (vertex based). Flux computations in the vertex-based scheme are identical to the edge-based scheme but are computed by processors associated with vertices. We show that the vertex-based scheme requires 50% less communication and asymptotically identical amounts of computation compared to the edge-based approach.

The rest of the paper is organized as follows. In Sec. II, we describe the algorithm used. Section III describes the desirable properties of a mesh-vertex scheme. In Sec. IV, we compare the numerical properties of the unstructured finite volume scheme with ARC2D on a sample airfoil. In Sec. V, we prove the optimality of our scheme and describe the use of directed graphs in balancing the amount of work assigned to each processor. Section VI discusses compiled communications on the Connection Machine (CM). Section VII contains results and comparisons, and conclusions are presented in Sec. VIII.

Received Oct. 29, 1990; presented as Paper 91-0441 at the AIAA 29th Aerospace Sciences Meeting, Reno, NV, Jan. 7–10, 1991; revision received May 7, 1991; accepted for publication May 20, 1991.

*Visiting Research Associate, Research Institute for Advanced Computer Science, Mail Stop T045-1; Ph.D. Student, Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180.

†Research Scientist, Computational Fluid Dynamics Branch, Mail Stop 202A.

II. Algorithm Description

In this work, we consider the integral form of the Euler equations of gas dynamics in a general region Ω with bounding curve $\partial\Omega$:

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega} \boldsymbol{u}\,\mathrm{d}a + \int_{\partial\Omega} \boldsymbol{f}(\boldsymbol{u},\boldsymbol{n})\,\mathrm{d}l = 0 \tag{1}$$

In this equation, $\boldsymbol{u}$ represents the vector of conserved variables for conservation of mass, momentum, and energy. The vector function $\boldsymbol{f}(\boldsymbol{u},\boldsymbol{n})$ represents the flux of mass, momentum, and energy through a surface with normal orientation $\boldsymbol{n}$. In developing a finite-volume scheme, we consider this integral form of the Euler equations for some tessellation $T(\Omega)$ of Ω comprising control volumes c_i, such that $c_i \cap c_j = 0$, $i \neq j$ and $\cup c_i = \Omega$. We then apply Eq. (1) to each control volume

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{c_i} \boldsymbol{u}\,\mathrm{d}a + \int_{\partial c_i} \boldsymbol{f}(\boldsymbol{u},\boldsymbol{n})\,\mathrm{d}l = 0 \tag{2}$$

Fundamental to the present method is the definition of the integral cell average $\bar{\boldsymbol{u}}$

$$(\bar{\boldsymbol{u}}A)_{c_i} = \int_{c_i} \boldsymbol{u}\,\mathrm{d}a, \quad A_{c_i} = \int_{c_i} \mathrm{d}a \tag{3}$$

Using this definition, we rewrite Eq. (2),

$$\frac{\mathrm{d}}{\mathrm{d}t}(\bar{\boldsymbol{u}}A)_{c_i} + \int_{\partial c_i} \boldsymbol{f}(\boldsymbol{u},\boldsymbol{n})\,\mathrm{d}l = 0 \tag{4}$$

In Godunov's method[3] and the extension considered here, the integral cell averages of $\boldsymbol{u}$ are treated as the fundamental unknowns. From this perspective, we interpret Eq. (4) as an evolution equation for the cell average in c_i. To evolve Eq. (4), we must evaluate the flux integral on ∂c_i. Godunov's method assumes a piecewise constant distribution of the solution in each cell (control volume). This is simply a constant function with value equal to the cell average. This situation is depicted in Fig. 1 for a one-dimensional mesh.

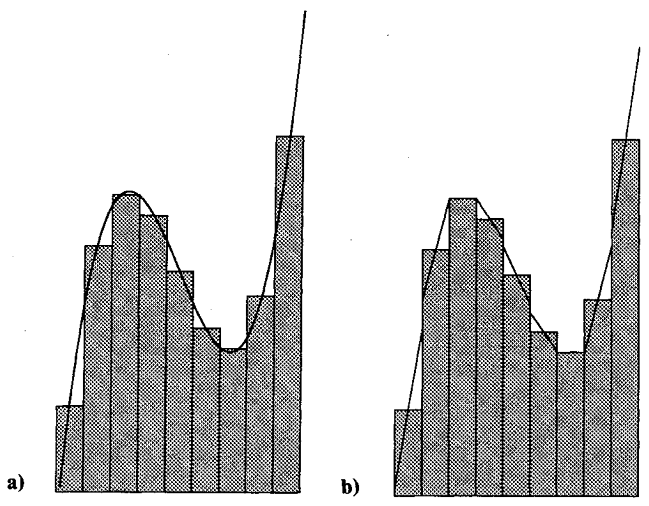

Fig. 1 Cell average for a one-dimensional mesh: a) function; b) piecewise linear approximation.

We have implemented a higher order extension of Godunov's scheme to arbitrary shaped cells using piecewise linear distributions of $\boldsymbol{u}$ in each cell. We denote the linear distribution of any component of $\boldsymbol{u}$ in cell c_i expanded about its centroid (x_{c_i}, y_{c_i}) by

$$u(x,y)_{c_i} = \bar{u}_{c_i} + (\nabla u)_{c_i} \cdot [x - x_{c_i}, y - y_{c_i}] \tag{5}$$

Linear piecewise polynomials are shown in Fig. 1 for a one-dimensional mesh. By the use of centroidal coordinates, we have that

$$\int_{c_i} u(x,y)_{c_i}\, \mathrm{d}a = \bar{u}_{c_i} \int_{c_i} \mathrm{d}a = (\bar{u}A)_{c_i} \tag{6}$$

From Eqs. (5) and (6), it is clear that any function that varies linearly in space with the correct integral average in a cell has precisely this integral average value at the centroid of the cell. In other words, we can refer to the solution unknowns as *pointwise* values of the solution located at the centroid of each control volume. More generally, pointwise values of the solution unknowns can be located at vertices of the mesh (which need not coincide exactly with centroids of the control volumes). This produces linear distributions of the form

$$u(x,y)_{c_i} = u_{v_i} + (\nabla u)_{c_i} \cdot [x - x_{v_i}, y - y_{v_i}] \tag{7}$$

This complicates the discretization of time terms appearing in Eq. (4) since the integral average of Eq. (7) involves the solution gradient $(\nabla u)_{c_i}$. For steady-state calculations, the time derivative can be approximated by

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_{c_i} u\, \mathrm{d}a \approx \frac{\mathrm{d}}{\mathrm{d}t} (u_i A_{c_i}) \tag{8}$$

without sacrificing spatial accuracy or conservation. This technique is similar to the mass-lumping approximation in finite elements.

Because the piecewise polynomials are discontinuous from cell to cell, along a cell boundary two distinct values of the solution can be obtained. To resolve this nonuniqueness, the Euler flux is supplanted by a numerical flux function $\bar{f}(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n})$, which when given these two solution states $\boldsymbol{u}^+$ and $\boldsymbol{u}^-$ produces a single unique flux. The numerical flux function is derived from exact and/or approximate solutions to the Riemann problem of gas dynamics. In our computations, we use the approximate solver developed by Roe.[4] Approximating Eq. (4) by piecewise polynomials and a numerical flux function we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t} (\bar{u}A)_{c_i} + \int_{\partial c_i} \bar{f}(\boldsymbol{u}^+, \boldsymbol{u}^-, \boldsymbol{n})\, \mathrm{d}l = 0 \tag{9}$$

To complete the discretization of the flux integral, we assume ∂c_i to be composed of straight line segments and perform a midpoint quadrature evaluation where (ξ_e, η_e) denotes the midpoint of each edge segment:

$$\frac{\mathrm{d}}{\mathrm{d}t} (\bar{u}A)_{c_i} + \sum_{e \in \partial c_i} \bar{f}[\boldsymbol{u}^+(\xi_e,\eta_e), \boldsymbol{u}^-(\xi_e,\eta_e), \boldsymbol{n}(\xi_e,\eta_e)] l_e = 0 \tag{10}$$

An important task in this solution process is the calculation of the piecewise linear solution distribution in each control volume given solution unknowns at vertices of the mesh. In the case of linear distributions, we insist that the linear functions be exact whenever the true solution varies linearly over the support of the cell discretization (distance-one neighbors of the mesh). To accomplish this task, we use numerical approximations to the exact Green-Gauss formula

$$\int_{\Gamma} \nabla u\, \mathrm{d}a = \int_{\partial\Gamma} u\boldsymbol{n}\, \mathrm{d}l \tag{11}$$

for some convenient path $\partial\Gamma$ surrounding c_i. For linear functions, the gradient is constant in Γ

$$\int_{\Gamma} \nabla u\, \mathrm{d}a = (\nabla u)_{\Gamma} A_{\Gamma}$$

Using pointwise values of the unknowns at vertices of the mesh, we choose a path connecting distance-one neighbors of the mesh (see Fig. 2). A trapezoidal quadrature formula for the integration of the right side of Eq. (11) guarantees that we can perform exact linear reconstruction (noniteratively) whenever the function varies linearly over the support of the reconstruction. Using some simple algebraic manipulations (see, e.g., Ref. 5), the trapezoidal integration about the path $\partial\Gamma_i$ can be rewritten as

$$(\nabla u)_{c_i} = \frac{3}{A_{\Gamma}} \sum_{\alpha} \frac{1}{2} (u_\alpha + u_0) \boldsymbol{n}_\alpha l_\alpha \quad \forall \alpha \in \{j,k,l,m,n,o\}$$

where $\boldsymbol{n}_\alpha$ is the normal of the control volume associated with the edge of the centroidal dual. For example, when $\alpha = j$, then $\boldsymbol{n}_\alpha$ and l_α are the normal and length of edge (j', k'). Note that this formula is similar in form to Eq. (10). Therefore, we can use the directed edge computation strategy developed for the flux summation.

To compute solutions with steep gradients and/or discontinuities without generating spurious oscillations, we must also insure that the reconstruction does not introduce new data extrema. This is accomplished by a slope limiting process. (See

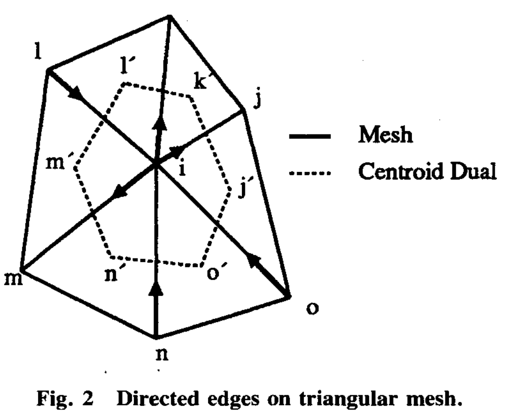

Fig. 2 Directed edges on triangular mesh.

Barth and Jespersen[1] for a discussion of the slope limiting method used in this implementation.)

To summarize the previous discussion, the solution process consists of three basic calculations:

1) Gradient calculation in each control volume: given solution unknowns, construct monotone piecewise linear polynomials for use in Eq. (11).

2) Flux evaluation on each edge: consider each control volume boundary ∂c_i to be a collection of edges from the tessellation. For each edge, perform a flux quadrature consistent with linear functions.

3) Evolution in each control volume: collect flux contributions in each control volume and evolve in time using any time-stepping scheme, i.e., Euler explicit, Euler implicit, Runge-Kutta, etc.

III. Why a Mesh-Vertex Scheme?

When using a finite volume formulation such as the one described in Sec. II, one must choose an appropriate control volume. One could use the triangles of the mesh, as shown in Fig. 3. Other tessellations of the problem domain can be constructed as well. Figure 4 shows a *dual* of the triangular mesh constructed by connecting centroids of adjacent triangles.

When one uses the triangles (faces) of the mesh for the control volumes, we call this a cell-centered scheme. Alternatively, a mesh-vertex scheme is when one uses the faces of the dual grid as control volumes.

There are several reasons to choose a mesh-vertex scheme instead of a cell-centered scheme. To see this, note that the three basic calculations just listed are either performed across edges (calculation 2) or within cells (calculations 1 and 3). One important property of the dual is that interior edges of a planar mesh and its dual are one to one. Therefore, an equal number of flux computations must be performed in both the mesh-vertex and the cell-centered schemes. But for a typical triangular mesh, there are approximately twice as many triangular control volumes as control volumes in the dual. Recall that calculations 1 and 3 are performed once for each control volume. The mesh-vertex scheme therefore needs half as much computation for these two operations.

When working on a dual tessellation, one can also gain improvements in accuracy. The average number of sides of the dual cells of a triangular mesh is six. The resulting numerical approximations are, therefore, significantly more isotropic with respect to wave orientation. Recent results by Lomax[6] indicate a dramatic improvement in phase error distributions on regular hexagonal control volumes as opposed to triangles or quadrilaterals. Finally, work by Roe[7] indicates an improvement in truncation error when using a mesh-vertex scheme instead of a cell-centered scheme.

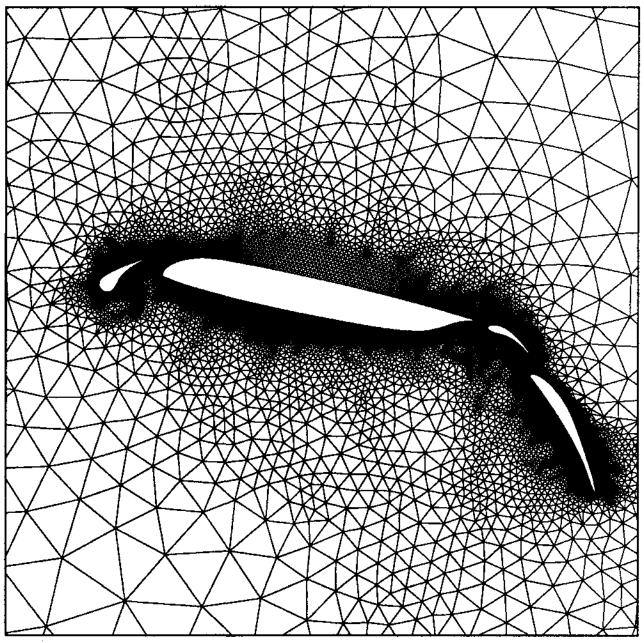

Fig. 3 Closeup of mesh about four-component airfoil with extended flaps.

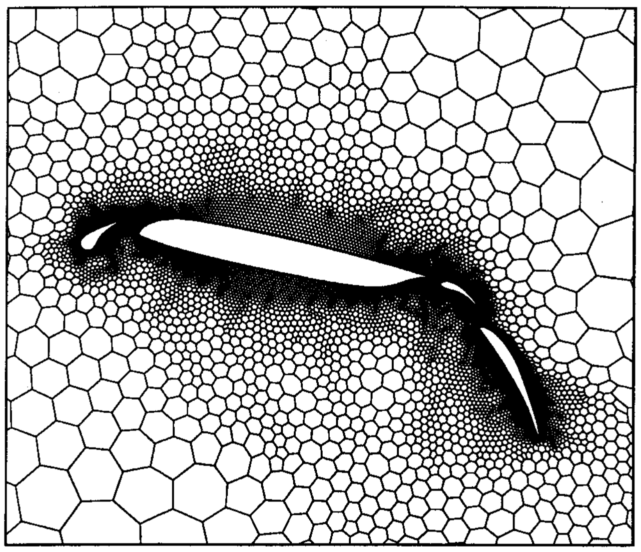

Fig. 4 Centroid dual constructed by connecting adjacent triangle centroids.

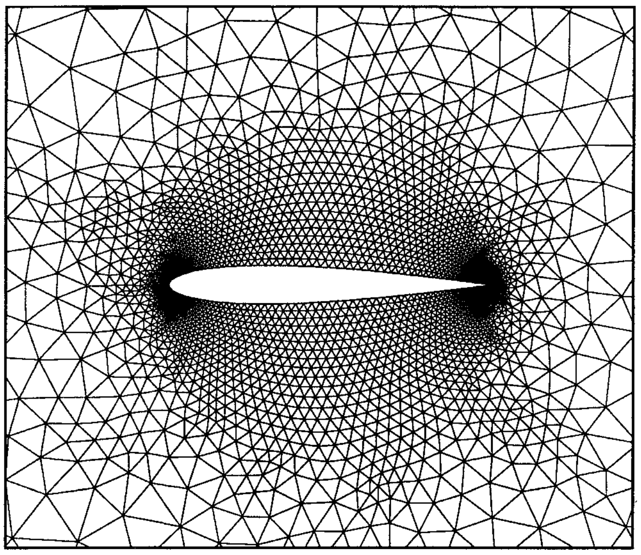

Fig. 5 Unstructured grid around NACA 0012 airfoil.

IV. Comparison with ARC2D

In this section, we validate the present parallel implementation of the Euler solver with the well-established ARC2D code.[8] The ARC2D code solves the Euler equations on structured meshes using an implicit time-advancement scheme with second-order-accurate spatial differencing. Computations were performed for Euler flow about a NACA 0012 airfoil at $M_\infty = 0.63$ and angle of attack $\alpha = 2.0$ deg. A 193 × 33 structured C-mesh was used for the ARC2D code and compared with a triangular mesh (see Fig. 5) containing approximately 3500 vertices and 6800 triangles. Steady-state solutions were obtained with both codes. The convergence history of the unstructured mesh code is shown in Fig. 6. This figure plots the L_2 norm of the stationary discretization vs the number of steps

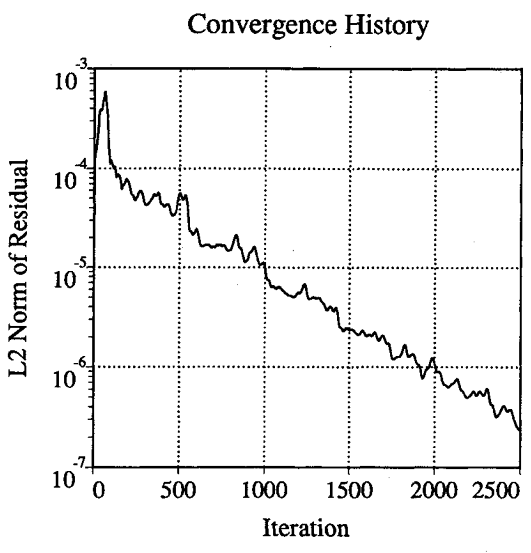

Fig. 6 Reduction in residuals as a function of the number of iterations.

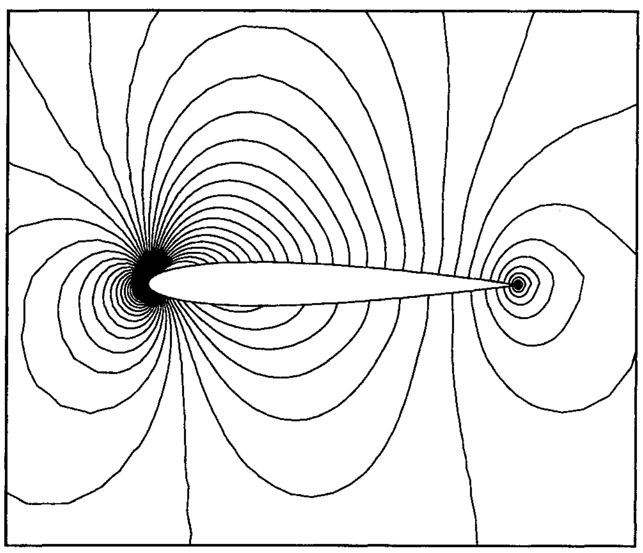

Fig. 8 Mach number contours: $M_\infty = 0.63$; $\alpha = 2.0$ deg.

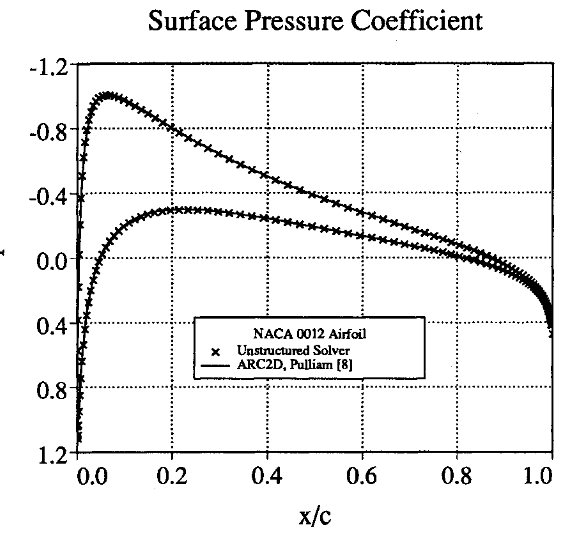

Fig. 7 Comparison of surface pressure computed by ARC2D and unstructured Euler code.

taken using a four-stage Runge-Kutta time advancement. The ARC2D surface pressure coefficient distributions is compared to the unstructured mesh code in Fig. 7. For purposes of fair comparison, the point-vortex correction used in ARC2D to simulate the effect of placing far-field boundaries at infinite distance has been disabled. Calculations were performed on the meshes with far-field boundaries placed approximately six chord lengths from the airfoil. The agreement between the two codes is excellent. Mach number contours are shown in Fig. 8. Mach number contours provide a useful critique of the solution accuracy. Poor solution accuracy of this isentropic flow is usually manifested by the appearance of a viscous-like slip surface downstream of the trailing edge of the airfoil. The Mach number contours match ARC2D contours (not shown) very well with no indication of a slip surface.

The reader is reminded that the parallel algorithm described in this paper performs computations identical to the code developed for vector processing in Ref. 1. Further validation of the high accuracy achievable with the present code for compressible Euler flows including shock waves can be found in this report.

V. Vertex-Based Partition

Given the aforementioned arguments for using a mesh-vertex scheme instead of a cell-centered scheme, one now must decide how to distribute the edge and vertex data structures to the processors of a massively parallel computer. The problem can be partitioned to assign one edge to each processor or one vertex to each processor. Choosing a vertex-based assignment is optimal for communication and computation.

Flux computations for the vertex-based decomposition are identical to those preformed in the edge-based scheme but are computed by processors associated with vertices. Each edge of the mesh joins a pair of vertices and is associated with one edge of the control volume (see Fig. 2). Consider vertices i and j. Suppose that they are assigned to processors p and q, respectively. Either processor p or q can compute the flux through the shared edge of the control volume (k',j'). In general, to determine which processor will perform the flux calculation, one can direct the edges of the mesh. When there is a directed edge from i to j, the flux across (k',j') is computed by p. Processor q (holding vertex j) sends its conserved values to p, flux is computed, and the result accumulated locally. Finally, the result is sent from p to q to be accumulated negatively.

Edge direction eliminates redundant flux calculations but can result in load imbalances if one vertex has many more outward pointing edges than the other vertices. We define outdegree to be the number of edges directed out from a vertex. Recently, Chrobak and Eppstein[2] have shown that there exists a linear time algorithm for orienting the edges of any planar graph such that no vertex has an outdegree greater than 3. Therefore, no processor needs to compute the flux across more than three edges of the control volume. This is critical on a single instruction multiple data (SIMD) computer since all of the processors perform the same computation at the same time. Directing the edges in this manner is an optimal load balancing.

For nonplanar graphs, bounding the outdegree by a constant is not possible. It should be possible to bound the outdegree to be equal to one half of the maximum degree of the graph. In general, tighter bounds are not known for nonplanar graphs.

The partitioning scheme is illustrated in Figs. 9. Figure 9a shows a list of vertices from 1 to V and a list of edges from 1 to E, where E and V are the number of edges and vertices in a triangular mesh. This represents an edge-based partition in which one edge is assigned to each processor. The first V of these processors contain data associated with the vertices. The remaining E-V processors do not hold any vertex information. During calculation 2, E-V processors must get data from two other processors, perform a flux computation, and then send results back to the same two processors. Recall that on a SIMD computer all processors do the same thing at the

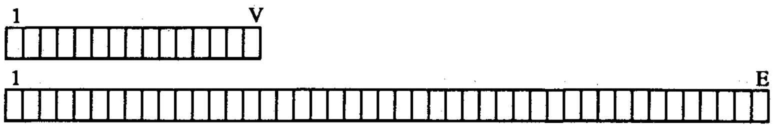

Fig. 9a Edge-based partition.

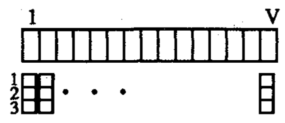

Fig. 9b Vertex-based partition.

same time. Therefore, there are a total of $4E$ communications required by an edge-based distribution of the data.

Figure 9b shows a list of vertices from 1 to V and a small array of length 3 below it. This represents a vertex-based partition where each vertex is assigned to one processor. Also, there are up to three incident edges (corresponding to outward directed edges) of each vertex stored in the array. During calculation 2, each processor loops over its three outward directed edges. For each of these edges, every processor does the following: it gets vertex data from one other processor, performs a flux computation, and then sends results back to the same processor. Every processor has half of the information needed for the flux computation stored locally in the vertex-based partition. For example, in Fig. 2, the processor holding vertex i will receive data from the processors holding vertices j, k, and m and send data to the processors holding vertices n, o, and l. After the processor holding vertex i computes the flux through (j',k'), (k',l'), and (m',n'), it sends results to the processors holding vertices j, k, and m and receives results from the processors holding vertices n, o, and l. Therefore, in the vertex-based partition, each processor performs two communications for each of its three outward directed edges, for a total of $2E$. This is half the number of messages required by an edge-based partition.

One should note that if there are many more processors in the computer than vertices in the mesh then an edge-based approach provides a way to exploit more parallelism since typically there are up to three times more edges than vertices in triangular discretizations of two-dimensional space.

Now, we prove that organizing the computation by vertices rather than by edges requires asymptotically the same amount of work. It is clear for calculations 1 and 3 that the vertex-based partition requires no redundant calculations since these computations are done for each vertex of the mesh and we have assigned one processor to each vertex. However, calculation 2 must be performed once for each edge of the control volume. It is obvious that an edge-based partition is efficient for this computation, but as we have shown, it is expensive in terms of communication. Recall that two vertices of the mesh share a control volume edge and that we direct the edges of the mesh to determine which processor performs the flux computation. We claim that there are approximately three times as many edges as vertices in a triangular discretization of 2-space. Therefore, if each processor associated with a vertex performs three flux calculations, then the vertex-based partition requires approximately the same amount of work. We can prove this but first we need to define some terms in graph theory.

Let G be a graph. Let the symbols v and ϵ denote the number of vertices and edges of a graph. A graph is *planar* if it can be drawn in the plane so that its edges intersect only at their ends. A *planar graph* partitions the rest of the plane into a number of connected regions; the closure of these regions are called *faces*. Let F be the set of faces and let ϕ be the number of faces of a planar graph. Further, let $f \in F$ and let $d(f)$ be the degree of the face. The degree of a face is the number of vertices incident to the face. Finally, let $\{F_b \subseteq F | f \in F_b, d(f) > 3\}$. We prove the following:

Claim: For a two-dimensional mesh of triangular elements,

$$\epsilon = 3v - 6 - \sum_{f\in F_b} [d(f) - 3]$$

Proof: Let G be a planar graph. Euler's formula for planar graphs[9] states that

$$v - \epsilon + \phi = 2 \tag{12}$$

For any planar graph, each edge is shared by two faces. Therefore, the sum of the degree of all of its faces is equal to twice the number of edges:

$$\sum_{i=1}^{\phi} d(f_i) = 2\epsilon \tag{13}$$

We can subtract 3 from each term in the summation and 3ϕ from the right to get

$$\sum_{i=1}^{\phi} [d(f_i) - 3] = 2\epsilon - 3\phi \tag{14}$$

or

$$\sum_{f\in F_b} [d(f) - 3] = 2\epsilon - 3\phi \tag{15}$$

Rearranging Eq. (12) and substituting into Eq. (15) yields

$$\sum_{f\in F_b} [d(f) - 3] = -\epsilon + 3v - 6 \tag{16}$$

Solving for ϵ yields

$$\epsilon = 3v - 6 - \sum_{f\in F_b} [d(f) - 3] \tag{17}$$

□

Therefore, if the number of boundary vertices is small relative to the number of interior vertices, then, asymptotically, there are three times as many edges as vertices in a triangular mesh.

VI. Fast Communication

When computing on a massively parallel computer, such as the Connection Machine CM-2, it is often more important to determine which processor gets which data than to determine which processor performs which computations. This is due to the high cost of communication relative to computation.

A feature of the communication within the flow solver here is that the communication pattern, although irregular, remains static throughout the duration of the computation. We take advantage of this by using a mapping technique developed by Hammond and Schreiber[10] and the Fast-Graph package developed for the CM-2 by Dahl.[11] The former is a highly parallel graph mapping algorithm that assigns vertices of the grid to processors in the computer such that the sum of the distances that messages travel is minimized. The latter is a software facility for scheduling completely general communications on the Connection Machine. The user specifies a list of source locations and destinations for messages and enables one to fully utilize the communication bandwidth of the machine.

We have incorporated the mapping algorithm and the communication compiler into the flow solver running on the CM-2 and have realized a factor of 30 reduction in communication time compared to using naive or random assignments of vertices to processors and the router.

VII. Timing and Results

We compare the performance of the unstructured flow solver on 8K processors of a Connection Machine CM-2 with one processor of a Cray-YMP. Note that this is one-eighth of each of the full machines. The code on the CM-2 is an implementation of the vertex-based partitioning of the mesh-vertex scheme with piecewise linear reconstruction and four-stage Runge-Kutta integration used for evolution in time. The code is written in *lisp, and timings were done using a Sun4/490 front end. The calculations in the comparison are all done in 32-bit arithmetic since the CM-2 we used did not have 64-bit hardware. The geometric calculations for edge lengths, edge normals, control volume areas, etc., were all precomputed in 64-bit arithmetic on the CM-2 and stored as 32-bit values. This was necessary to obtain accurate gradients used in linear reconstruction and probably indicates the need for 64-bit precision for this type of computation. This will be especially important for viscous flow calculations where the control volume areas will be orders of magnitude smaller. The 64-bit calculations on the CM-2 were computed in software, and the initialization was not timed as part of the benchmark on either machine.

The calculations performed on the Cray-YMP are identical to the ones performed on the CM-2 except that the data are distributed to the processors using an edge-based partitioning scheme. It is written in Fortran, and all computations are in 64-bit arithmetic. Also, the clock period on the YMP is 6 ns. We used the flow tracing package *perftrace* to analyze the Cray code to determine the floating point usage. There are approximately 300 floating point operations per edge, per iteration of the flow solver. The code is vectorized over edges of the dual grid and utilizes gather scatter. The shortest vector has length 500. The main part of the code consumes 97% of the time and sustains 150 Mflops.

The test case we used has 15,606 vertices, 45,878 edges, 30,269 faces, 4 bodies, and 949 boundary edges. Note that, since there are approximately twice as many vertices as processors on the CM-2, two vertices are assigned to each processor. (The code runs at virtual processor ratio of 2.) Part of the grid is shown in Fig. 3.

The flow was computed at a Mach number of 0.1 at 0-deg angle of attack relative to the mesh; 100 time steps of the code on one processor of a dedicated YMP took 39 s. On the CM-2, the same computation took 45 s.

VIII. Conclusions

A novel vertex-based partitioning of the problem is introduced that minimizes the computation and communication costs associated with distributing the computation to the processors of a massively parallel computer. We have shown that each of the three primary operations (calculations 1–3) can be carried out using the directed graph edges without generating redundant computation or communication. Finally, we have shown that the performance of this unstructured computation on 8K processors of the Connection Machine CM-2 compares favorably with one processor of a Cray-YMP—45 s vs 39 s for 100 iterations of the flow solver.

Acknowledgments

This work was partially supported by the Defense Advanced Research Projects Agency via Cooperative Agreement NCC 2-387 between NASA and the University Space Research Agency. The Connection Machine CM-2 and the Cray-YMP used in this research are located at NASA Ames Research Center. We also wish to thank Dennis Jespersen for supplying the mesh generator to us.

References

[1]Barth, T. J., and Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, 27th Aerospace Sciences Meeting, Reno NV, Jan. 1989.

[2]Chrobak, M., and Eppstein, D., "Planar Orientations with Low Out-Degree and Compaction of Adjacency Matrices," *Theoretical Computer Science*, Vol. 84, July 1991, pp. 243–266.

[3]Godunov, S. K., "A Finite Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dyanmics," *Matematicheskie Sbornik,* Vol. 47, 1959.

[4]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics,* Vol. 43, 1981.

[5]Barth, T. J., "On Unstructured Grids and Solvers," *Computational Fluid Dynamics,* Lecture Series 1990-03, Von Kármán Inst., Belgium, March 1990.

[6]Lomax, H., private communication, 1990.

[7]Roe, P. L., "Error Estimates for Cell-Vertex Solutions of the Compressible Euler Equations," Inst. for Computer Applicatiions in Science and Engineering, Rept. 87-6, Hampton, VA, 1987.

[8]Pulliam, T. H., "Euler and Thin Layer Navier Stokes Codes: ARC2D, ARC3D, Notes for Computational Fluid Dynamics User's Workshop," Univ. of Tennessee Space Inst., Tullahoma, TN, March 1984.

[9]Bondy, J. A., and Murty, U. S. R., *Graph Theory with Applications,* Elsevier, New York, 1976.

[10]Hammond, S. W., and Schreiber, R., "Solving Unstructured Grid Problems on Massively Parallel Computers," *Proceedings of Society of Photo-Optical Instrumentation Engineers 1990 International Symposium on Optical and Optoelectric Applied Science and Engineering,* edited by F. Luk, International Society for Optical Engineering, Bellingham, WA, 1990.

[11]Dahl, E. D., "Mapping and Compiled Communiation on the Connection Machine System," *Proceedings of the Fifth Distributed Memory Computer Conference,* edited by D. W. Walker and Q. F. Stout, IEEE Computer Society, Los Alamitos, CA, 1990.