# Naval Research Laboratory

Washington, DC 20375-5320

# A Survey of Three Scalable Hydrodynamics Codes

MARK H. EMERY

*Center for Computational Physics Developments*
*Laboratory for Computational Physics and Fluid Dynamics*

June 3, 1996

19960627 015

DTIC QUALITY INSPECTED 1

# REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget. Paperwork Reduction Project (0704-0188), Washington, DC 20503.

| 1. AGENCY USE ONLY (Leave Blank) | 2. REPORT DATE<br><br>June 3, 1996 | 3. REPORT TYPE AND DATES COVERED | |
|---|---|---|---|
| **4. TITLE AND SUBTITLE**<br><br>A Survey of Three Scalable Hydrodynamics Codes | | | **5. FUNDING NUMBERS**<br><br>RP01433<br>PE - 62633N |
| **6. AUTHOR(S)**<br><br>Mark H. Emery | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**<br><br>Naval Research Laboratory<br>Washington, DC 20375-5320 | | | **8. PERFORMING ORGANIZATION REPORT NUMBER**<br><br>NRL/FR/6440--96-9817 |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**<br><br>Office of Naval Research<br>Arlington, VA 22217-5660 | | | **10. SPONSORING/MONITORING AGENCY REPORT NUMBER** |
| **11. SUPPLEMENTARY NOTES** | | | |
| **12a. DISTRIBUTION/AVAILABILITY STATEMENT**<br><br>Approved for public release; distribution is unlimited. | | | **12b. DISTRIBUTION CODE** |

**13. ABSTRACT (Maximum 200 words)**

Massively parallel computer architectures offer greater memory capacity and faster computational speeds through the cooperative effort of hundreds or thousands of individual processors. This makes possible the solution of problems previously impractical with conventional vector supercomputers. This report presents a survey and partial assessment of several different hydrodynamic codes with a focus on scalability; i.e., codes that are amenable to implementation on high-performance parallel computer architectures. Issues, such as decomposing the physical problem into naturally parallel parts, load balancing the computation across multiple processors, effectively communicating data between processors, and manipulation and display of large sets of data are discussed from the perspective of three different hydrodynamic codes.

| 14. SUBJECT TERMS | | 15. NUMBER OF PAGES |
|---|---|---|
| Scalability<br>Hydrodynamic codes | Parallel architectures<br>Parallel computing | 18 |
| | | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFIED | UNCLASSIFIED | UL |

NSN 7540-01-280-5500

# DISCLAIMER NOTICE
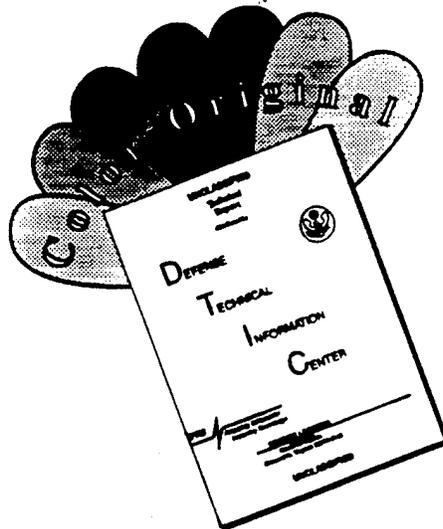
THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF COLOR PAGES WHICH DO NOT REPRODUCE LEGIBLY ON BLACK AND WHITE MICROFICHE.

# CONTENTS

# A SURVEY OF THREE SCALABLE HYDRODYNAMICS CODES

## INTRODUCTION

Presented here is a survey and partial assessment of several different hydrodynamic codes with a focus on scalability; i.e., codes that are amenable to implementation on high performance parallel computer architectures. Massively parallel architectures offer greater memory capacity and faster CPU speeds through the cooperative effort of hundreds or thousands of individual processors. Each processor works on its own portion of the problem in parallel with the others and communicates its results with other processors when necessary. This makes possible the solution of problems previously thought to be impractical with conventional vector supercomputers. However, exploiting scalability in physical problems brings with it a new set of problems to be solved. Issues, such as decomposing the physical problem into naturally parallel parts, load balancing the computation load across multiple processors, communicating data effectively between processors, and manipulating and displaying large sets of data, become important [1].

There are two principal strategies for parallel computing: the single-instruction, multiple-data approach (SIMD) and the multiple-instruction, multiple-data (MIMD) approach. In the SIMD approach, processors perform their operations in unison under the control of a master processor, thus they are particularly effective on problems where the parallelism arises through the parallel structure of the data. Computational problems with data irregularities or irregular computational requirements lead to a loss of efficiency. SIMD machines operate very efficiently on nearest neighbor transfer of information; however, general data exchange operations, as required for unstructured grids, are very inefficient. Special routers or precompilers have been developed in an attempt to circumvent this problem. These machines are not practical for more general types of applications where the grid topology changes every few time steps (remeshing, h-refinement, etc.). Examples of SIMD architecture with distributed memory are the Connection Machines—CM-1, CM-2, and CM-200, and the MASSPAR-1.

For MIMD, processors operate independently on their data, possibly using different instructions, with coordination between the processors handled by the explicit passing of messages and synchronization hardware. Most of the algorithmic considerations designed for vector machines are applicable to many MIMD machines since, in many current designs, each node of a MIMD system is a vector processor. Because MIMD computers store their code locally, they are capable of running with many input-output (I/O) nodes by designating certain processors as I/O nodes. The software development for MIMD machines is, in many instances, more complex than that for SIMD. The Intel series of machines—the Cray T3D, IBM SP1 And SP2, and the CM-5—are examples of MIMD architecture.

Recent achievements in microprocessor technology and the capability to produce inexpensive high-speed processors have led to hardware developments that outpace software developments. The Department of Defense (DoD) has taken significant steps to accelerate the exploitation of scalable, parallel, high-performance computing systems to solve complex computational problems of critical importance to the DoD. One of the motivations of this effort is the realization that code development must keep pace with the evolving computational environment while focussing on cost effectiveness.

These efforts encompass a broad spectrum of research endeavors. This report focuses on a small subset of these efforts in the area of hydrodynamics—specifically, computational fluid dynamics— and deals with several computational models and their potential for scalability that would be applicable to the solution of underwater explosive bubble dynamics. This problem is characterized by large material distortions that severely limit the applicability of a pure Lagrangian method. The report focuses on three general-purpose computational fluid dynamics (CFD) (or hydrodynamic) codes that can model highly dynamic conditions and shock propagation.

CFD is the accurate numerical solution of the equations describing fluid and gas motion and the related use of digital computers in fluid dynamics research. CFD is used for basic studies of fluid dynamics, for engineering design of complex flow configurations, for predicting the interactions of chemistry with fluid flow for combustion, propulsion, and explosive reactions, for interpreting and analyzing experimental data, and for extrapolation into regimes that are inaccessible or too costly to study. CFD encompasses all velocity regimes and scales of interest to the DoD. The physics to be considered may entail additional force fields, changes of phase, changes of chemical composition, interactions among multiple phases of heterogeneous flows, and coupling to surface physics and microphysics. CFD has no restrictions on the geometry and motion of boundaries defining the flow.

Rapid advances in hardware and software on today's modern, parallel computers have provided unique opportunities for cost effective CFD calculations. Although these architectures tend to require significant restructuring and rewriting of the existing program, the opportunity for great improvements in efficiency does exist. An additional incentive to port software to these parallel computers lies in their modest cost as compared to the easier-to-use general supercomputers like the Cray.

There are as many numerical models in CFD as there are problems to be solved. In this report, we restrict our consideration to three models. As noted above, this is somewhat driven by the problem to be solved (underwater explosive dynamics) and the requirement that this report be finite in length.

## NUMERICAL MODELS

CFD has advanced to the point where the flow field about arbitrary, complex, three-dimensional (3-D) bodies can be solved given enough memory and CPU time. The first step in the solution process is the construction of an appropriate grid to represent the computational domain of interest. The criterion for delineating the various models is the methodology used to represent the grid in complex geometries. Grid methods are broadly defined as unstructured or structured.

For unstructured grids, the computational domain is subdivided into an irregular set of computational cells, usually tetrahedra in 3-D. This method is possibly optimal in terms of flexibility, degrees of freedom, and ability to "conform" the grid to a complex shape. A number of algorithms are known for automatic unstructured grid generation [1,2]. However, the unstructured grid approach is not well suited for either SIMD or MIMD parallelism because of the large demands placed on both computer memory and CPU time. One of the features of the unstructured grid is that the number of elements surrounding an interior node is not necessarily constant; thus additional data have to be stored to get information on the neighbors, and additional CPU time is used to transfer the information between elements and nodes. A number of schemes have been developed to minimize this difficulty [3,4].

For structured grids, the computational domain is divided into an assembly of quadrilateral or hexahedral cells. Here, each interior node is surrounded by the same number of elements; thus, this methodology is ideally suited for MIMD or SIMD parallelism. Typical structured grid approaches are unsuited for complex geometries because of the need to balance computational speed against cell resolution near the complex shape. Numerical algorithms such as CIA [5] and virtual-cell embedding (VCE) [6] take advantage of the large memory available with parallel architectures and improved volume resolution about the complex geometry without much of a sacrifice in computational memory or speed. With these algorithms, it is possible to construct an acceptable grid to regions of general shape.

There are several additional considerations with respect to grid generation when developing algorithms for MIMD machines. To maximize the advantage of a multiprocessor system, each processor should be assigned an equivalent amount of computation to minimize idle time. This is referred to as load balancing. Basically, the problem is to map the computational blueprint for the discretized equations onto the blueprint for the network of the massively parallel computer in such a way that the overall cost of the simulation is minimized. The domain should be decomposed in such a way as to minimize the amount of interprocessor transfer time relative to microprocessor performance. This is equivalent to minimizing the ratio of the surface area to the volume of the computational domain assigned to each processor. The more complex the griding becomes, the more difficult it is to minimize this ratio.

The three codes discussed in this article are: *FEFLOIC* [7,8], an implicit, 3-D, incompressible flow code using unstructured meshes; *PCTH* [9,10], an explicit, 3-D, compressible, strong shock wave code using an Eulerian griding scheme; and *FAST3D-VCE* [6,11], an explicit, 3-D, monotone, positivity preserving hydrodynamics code using a structured, rectilinear Eulerian grid with cell subdivision near geometrically complex bodies.

## FEFLOIC

This code is a classic example of a 3-D, finite element, unstructured grid flow solver [7,8]. The incompressible Navier-Stokes equations are discretized in the arbitrary Lagrangian-Eulerian (ALE) form. The resulting linearized matrix system is solved sequentially using a projectionlike method [12,13]. The large systems of equations in the two-step process are solved iteratively with a preconditioned conjugate gradient algorithm. The preconditioning is achieved through linelets [14]. This model has been applied to unsteady aerodynamic flows and turbulent separating flows over complex structures such as submarines.

This code is also an example of how the algorithm and the problem to be solved control the choice of computer architecture, SIMD or MIMD. As noted above, the unstructured grid offers good flexibility with few degrees of freedom and involves straightforward adaptive meshing strategies for dynamic resolution in transient problems. The mesh generation employs the advancing front technique [15] that requires different tasks to be performed in different subdomains. As noted above, the implicit flow solver uses linelets as a preconditioner to achieve improved convergence rates. Both the mesh generation technique and the implicit nature of the flow solver necessitate MIMD architecture. The use of SIMD architecture would require that linelets be used in all subdomains (or not at all). This would eliminate the possibility of retaining an implicit flow solver in regions such as a boundary layer (where the cell sizes are small) and an explicit flow solver where the cell sizes are large. In an unstructured mesh, the communication pattern is quite irregular, and SIMD architectures are efficient only for communication between nearest neighbors. Note that mapping techniques [16] can be used to assign vertices of the unstructured mesh to processors, thus minimizing

communication costs stemming from the irregular communication pattern; however, this is not realistic in situations where adaptive remeshing is taking place.

*FEFLOIC* has been ported to a MIMD environment such as the Intel DELTA prototype. A description of the problem to be addressed (surfaces, initial condition, etc.) are typically formed from a CAD-like description, and the sequence of events required to perform a run on the MIMD machine is as follows [8]:

- generate the mesh either on a serial or parallel machine;
- partition the mesh by using a load balancing algorithm with a serial or parallel machine [17];
- integrate the flowfield on a parallel machine; and
- display the results on a supergraphics workstation.

The algorithm is parallelized through domain decomposition. To be efficient, the workload in the subdomains must be balanced and executed without incurring substantial overhead. The workload in each subdomain is summed and any imbalance in the surplus/deficit of this workload between the element and its neighbors is computed. Elements are then added to subdomains with a deficit, and the process is repeated until a balance is achieved [17].

Once the domain has been partitioned, each subdomain is assigned to a processor. There is one layer of elements overlapped between adjacent subdomains. The information at the vertices associated with this layer of elements are communicated between processors. The interprocessor communication is achieved by adding one extra layer of elements. When information from neighboring domains is required, a central library of subroutines is invoked. Details of the parallel implementation are available in Ref. 8.

Figure 1 shows the scalability of the code [8]. This shows the CPU time taken per-point-per-timestep ($\tau$) for steady flow over a NACA 0012 airfoil. For constant problem size (2992 points), $\tau$ decreases linearly as the number of processors increases from one to four but then deviates from ideal scaling as the number of processors increases because of increased communication time. For a grid of 98,022 points and 194,993 elements, $\tau$ decreases linearly from 124 $\mu$s to 32 $\mu$s, as the number of processors is increased from 32 to 100. Figures 2(a) and (b) show the grid and the domain decomposition for the airfoil study. Figures 3(a) and (b) show the domain decomposition and pressure contours for flow past a DARPA-2 submarine model.

## PCTH

*PCTH* is a software system used for modeling multidimensional, multimaterial, large deformation, strong shock-wave physics [9,10]. *PCTH* is the scalable version of the CTH shock-physics code and is under active development at Sandia National Laboratory. The conservation equations for mass, momentum, and energy are cast into a finite volume format. A two-step solution scheme is used on a fixed Eulerian mesh. The first step is a Lagrangian step in which the cells distort to follow the material motion, and the second step is a remesh step where the distorted cells are remapped back onto the Eulerian mesh. Three-dimensional rectangular geometry is available with *PCTH*. This code is used for solid dynamics problems such as armor design, penetrator design, and explosive component design and can model alloys, ceramics, glasses, geological materials, and energetic materials [18].
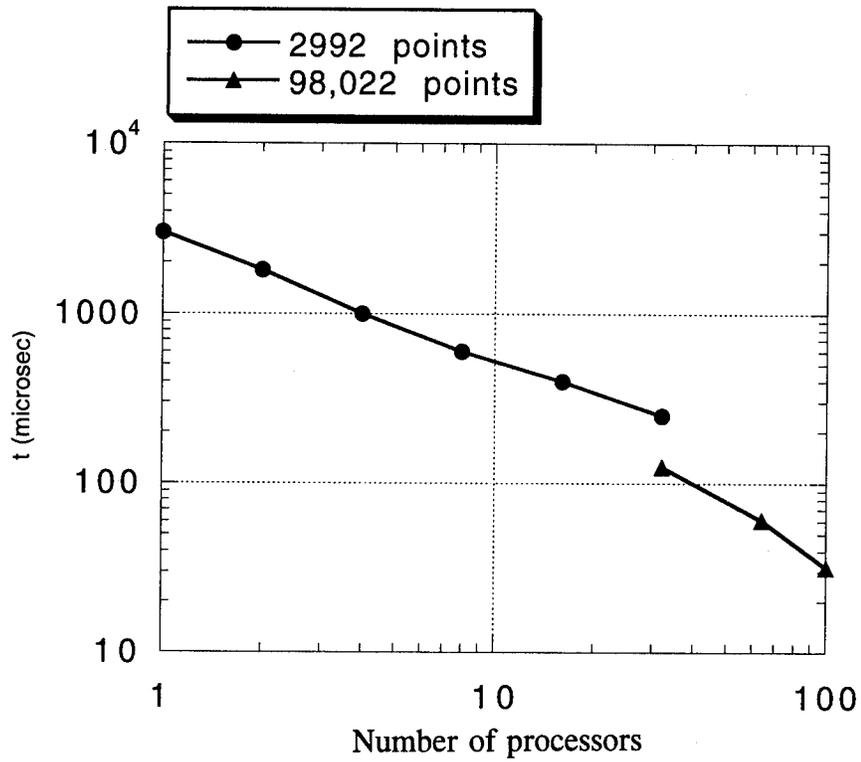
Fig. 1 — Performance of the implicit flow solver *FEFLOIC* from Ref. 8 (Used with permisssion)
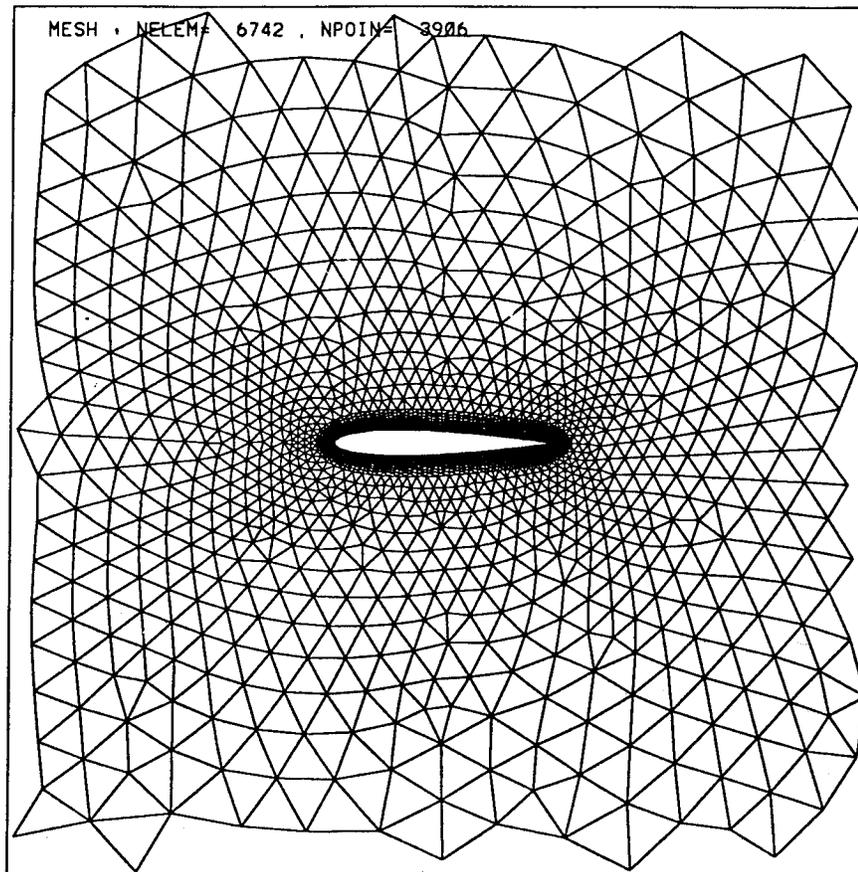


Fig. 2(a) — Grid for simulation of flow past a NACA0012 airfoil
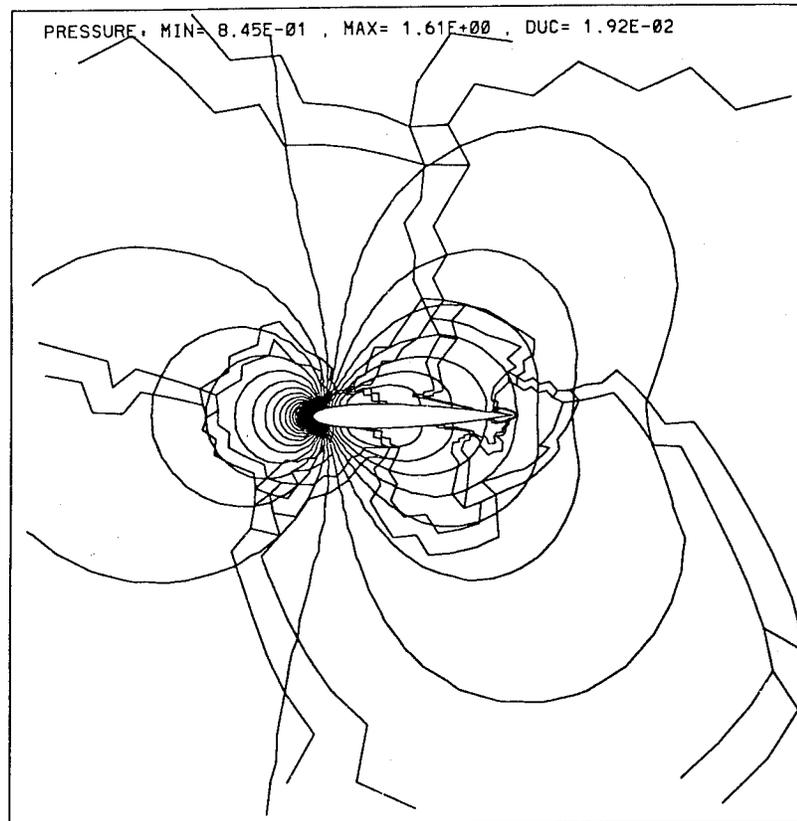
Fig. 2(b) — Pressure contours and domain decomposition for the same problem. From Ref. 7.
(Used with permisssion)

*PCTH* uses second-order accurate algorithms to reduce dispersion and dissipation and incorporates a high-resolution interface reconstruction scheme to prevent numerical breakup and distortion of material interfaces. Up to ten materials and void can occupy a computational cell. Several material models are available: analytic equations of state for the nonlinear behavior of materials in the high-pressure regime, elastic-perfect plastic with material softening, programmed explosive burn, and a history variable model, the Jones-Wilkins-Lee analytic model for explosive reaction products, and fracture.[19]

*PCTH* is designed for distributed memory parallel architectures (MIMD). The mesh is broken into several subdomains. Each subdomain is sent to a different processor. The problem scales to different size parallel computers by dividing the mesh into a different number of subdomains. Explicit message-passing is used to communicate between processors. The communication software can be modified easily and is easily structured to use either vendor provided communication routines or PVM3 message passing routines. *PCTH* is written in the $C_{++}$, ANSI Fortran77, and C languages. The code requires the standard UNIX tools LEX and YACC for input parsing, and the code generated by these software tools is modified automatically by the *PCTH* Makefile system [18]. *PCTH* runs on several distributed memory parallel machines including the Intel Paragon, Intel Delta, nCUBE2, networked HP's workstations, and networked Sun workstations.

*PCTH* runs approximately five times faster on the 1024 nCUBE2 massively parallel computer than on the single processor Cray YMP and approximately 30 to 60 times faster on the 1840 processor Intel Paragon than on the single processor Cray YMP. Recent simulations on 256 nodes of the Paragon include the formation of a shaped charge and the impact of the charge on a rod of high explosive PBX-9501 and the explosive welding of a copper tube to a steel plate [18].
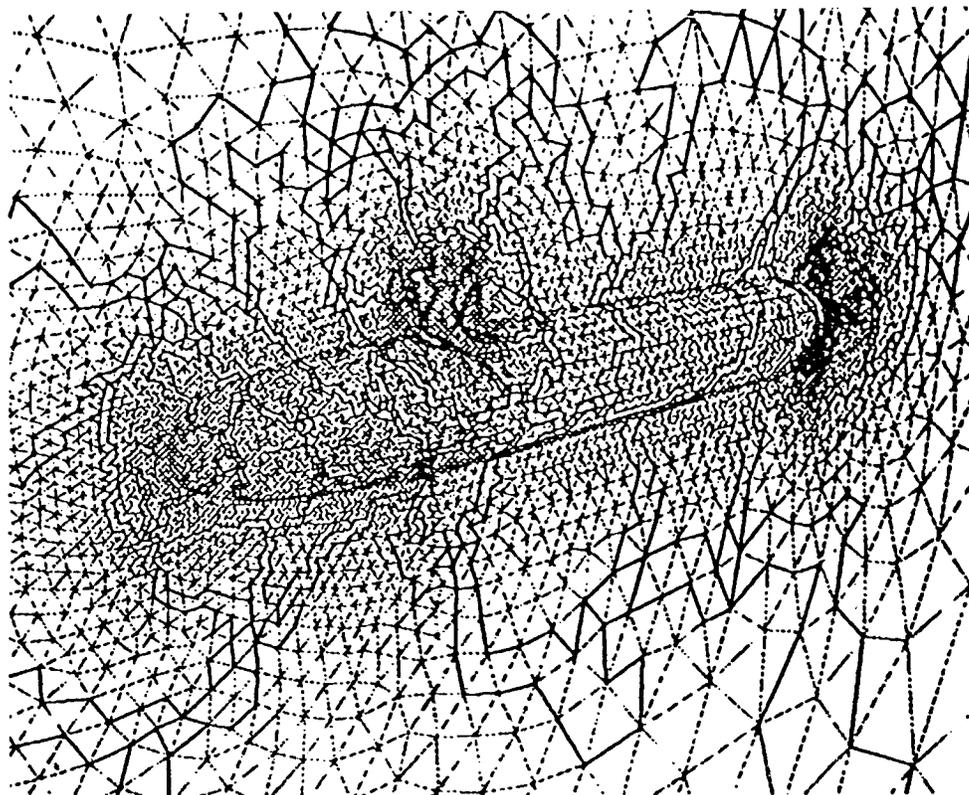
Fig. 3(a) — The mesh and domain decomposition for the DARPA-2 submarine problem
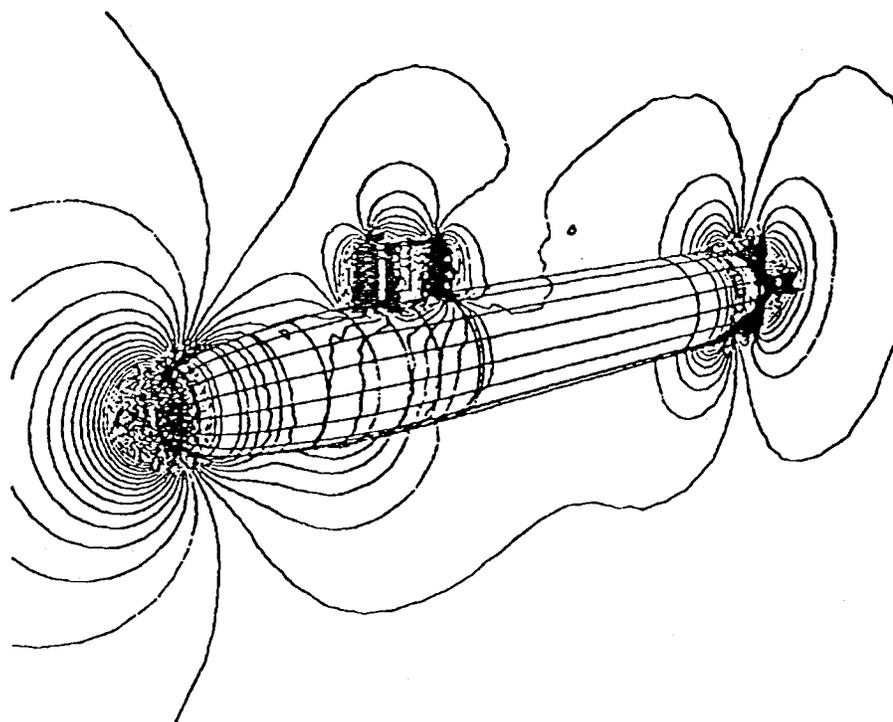


Fig. 3(b) — Pressure contours of flow past a DARPA-2 submarine model, from Ref. 8.
(Used with permission.)

**FAST3D-VCE**

The parallel *FAST3D* CFD code with the VCE grid generator for complex geometry uses a globally structured grid [6,11]. *FAST3D* is a compressible fluid dynamics flow-solver applicable to problems such as noise generation in supersonic jets and reactive flow in ram accelerators, as well as low speed flows such as the unsteady airwake over a complex Navy ship superstructure [11]. The multiphase version of *FAST3D* is being used to model wave-breaking over complex geometries and underwater explosion and bubble dynamics. *FAST3D* was designed to be scalable since these applications require large grid sizes as well as long run times to collect unsteady flow field data. *FAST3D* solves the unsteady 3-D transport equations using the flux-corrected transport (FCT) [20] algorithm with the VCE method [6] for complex geometries. The FCT algorithm is a high-order, monotone, conservative, positivity preserving algorithm. Thus when a convected quantity such as density is initially positive, it remains positive, and no new maxima or minima are introduced due to numerical errors in the convection process. Monotonicity is achieved by introducing a diffusive flux and later correcting the calculated results with an antidiffusive flux modified by a flux limiter. In the *FAST3D* model, the FCT algorithm solves the 3-D flow equations by using a direction-split, time-split method, essentially creating a series of 1-D equations, which the FCT algorithm time integrates. For multidimensional problems, *FAST3D* is implemented typically in an Eulerian fashion. In addition, a structured, orthogonal, rectilinear mesh is used for 3-D problems. In turn, this would require extremely fine grids for accurate solution around complex bodies; however, the VCE method circumvents this problem.

The VCE method improves the accuracy around complex bodies without sacrificing speed or memory [6]. While the VCE method uses a structured, rectilinear grid, the cells with the VCE method may be fully outside the body, fully inside the body, or partially inside the body. It is the partially obstructed cells—those cells "cut" by the surface of the body—that are given special treatment. To compute the fluxes correctly in these partially obstructed cells, the unobstructed face areas and volumes are needed. These cells are subdivided into a number of smaller subcells with which the areas and volumes can be calculated to arbitrary accuracy. Using the VCE method, only those cells next to the body are subdivided. The term "virtual" is used since the subcells embedded within a cell are not stored in memory and, therefore, are not integrated in the flow solution. CPU time is not sacrificed appreciably since only those cells next to the body require special treatment. The information computed from the subcells (computed once and stored) is stored in a list at the location indicated by a pointer stored in the parent cell. The division into subcells can be made so fine that the body is essentially smooth without staircasing. VCE has been shown to provide accurate results when tested on a series of 2- and 3-D problems [6].

The full 3-D, time-dependent model *FAST3D-VCE* has been implemented in parallel on the Intel iPSC/860 Touchstone computer [11]. Each node of the Touchstone is based on the Intel i860 high-performance chip that has vector-processing capability. The Touchstone architecture is a hypercube with distributed memory and MIMD data flow. Since memory is distributed across the nodes, internode communication becomes of paramount importance. The orthogonal, structured grid, FCT algorithm is based on operator-spliting techniques that enable the 3-D flow field to be solved as a series of 1-D integrations along the three independent coordinate directions. This multidimensional data structure, coupled with the VCE technique for complex geometries, is particularly amenable to parallelization. As with the previous cases, the choice of computer architecture was driven somewhat by the problem to be solved and the method of solution. In this case, there was the desire to preserve much of the original code, retain the long vector features, interface with the simpler to program vector computers to do the initialization for complex geometries, and be able to modify resolution during a computer run. To meet these criteria, an efficient running transpose (RT) data structure was

developed, and a system of virtual nodes was incorporated to simplify the transpose on varying numbers of physical nodes [11].

The fluid equations are integrated forward in time on a computational domain which is an Nx X Ny X Nz rectilinear, structured grid. The 3-D flow field is partitioned across the nodes so that each node's subset contains its share of full-length vectors at each stage of the full 3-D transpose. In a typical simulation, the blocks are on the order of tens to hundreds of kilowords. These blocks are swapped with their replacement blocks in counterpart nodes and are transmitted as a single large message at each stage of the RT. This approach minimizes communications overhead and entirely eliminates interprocessor communications from the fluid geometry and fluid dynamic integration sections of the code. This does require an efficient RT algorithm because of the size of the large data blocks being transmitted, unlike the standard static block domain decomposition that requires only that the data along the block surface interaction be transferred.

There are several reasons for invoking this methodology [11]:

- overhead for communications startup is minimal;
- the transpose proceeds at near optimum data rates on a hypercube architecture;
- the data blocks are very large and there are few transfers per time step;
- the data structure maintains its full-length vector features at each processor, improving optimization; and
- the data structure is exactly the same for both scalable and serial machines, allowing for preservation of much of the original code.

The parallel performance characteristics indicate that the benefits of this data structure greatly outweigh the minimal overhead costs associated with the running transpose. The CPU time for the hydrodynamics operations scale nearly as the inverse of the number of nodes. This confirms that a high degree of scalability has been achieved. The RT scales at a slower rate. (See Fig. 4.) In practice, with a variety of architectures and sizes, the running transpose accounts for less than 10% of the total CPU time. Standard benchmark calculations indicate that 32 nodes of the Intel iPSC Touchstone is equivalent to one Cray C90 processor and five Power Challenge SGI nodes. The code has been used to model the flow characteristics over ship superstructures. The DDG51 destroyer geometry is shown in Fig. 5. This geometry was obtained from a CAD package that constructed the shell of the ship with a series of plates. Figure 6 shows the typical unsteady flow pattern down the centerline of the DDG51 destroyer [21].

## ADDITIONAL CONSIDERATIONS

As the above examples showed, the algorithm and the problem to be solved can control the choice of computer parallel architectures—SIMD or MIMD. Several other issues should also be considered when developing scalable software from existing serial code. Strong consideration should be given to keeping restructure of the original code to a minimum. There are several benefits in this approach:

- it minimizes the interactive cycle time between the user and the simulation;
- it allows for rapid and efficient verification/validation with serial code;
- it enhances the rapid portability of the code to various architectures; and
- it allows for easier and more rapid revision/modification of the code to incorporate new algorithms and technologies.

Time for FCT plus overhead
Time for RT

t (secs/timestep)

Number of processors
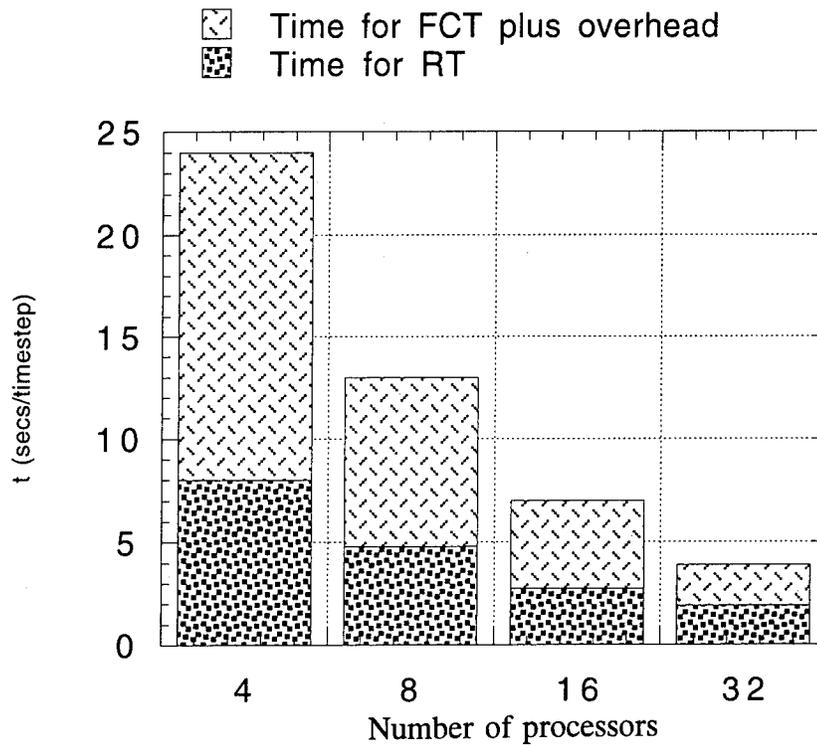
Fig. 4 — Bar chart shows the timing for *FAST3D* for a 96 × 64 × 64 grid. The total length represents the total time taken for each time step. The lower portion represents the time for the running-transpose. From Ref. 11. (Used with permission).
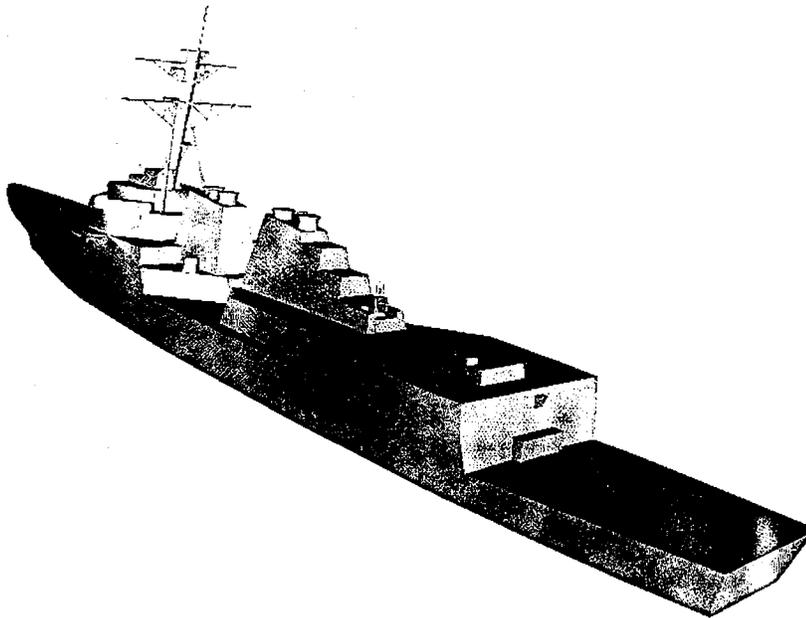
Fig. 5 — The DDG51 Destroyer geometry
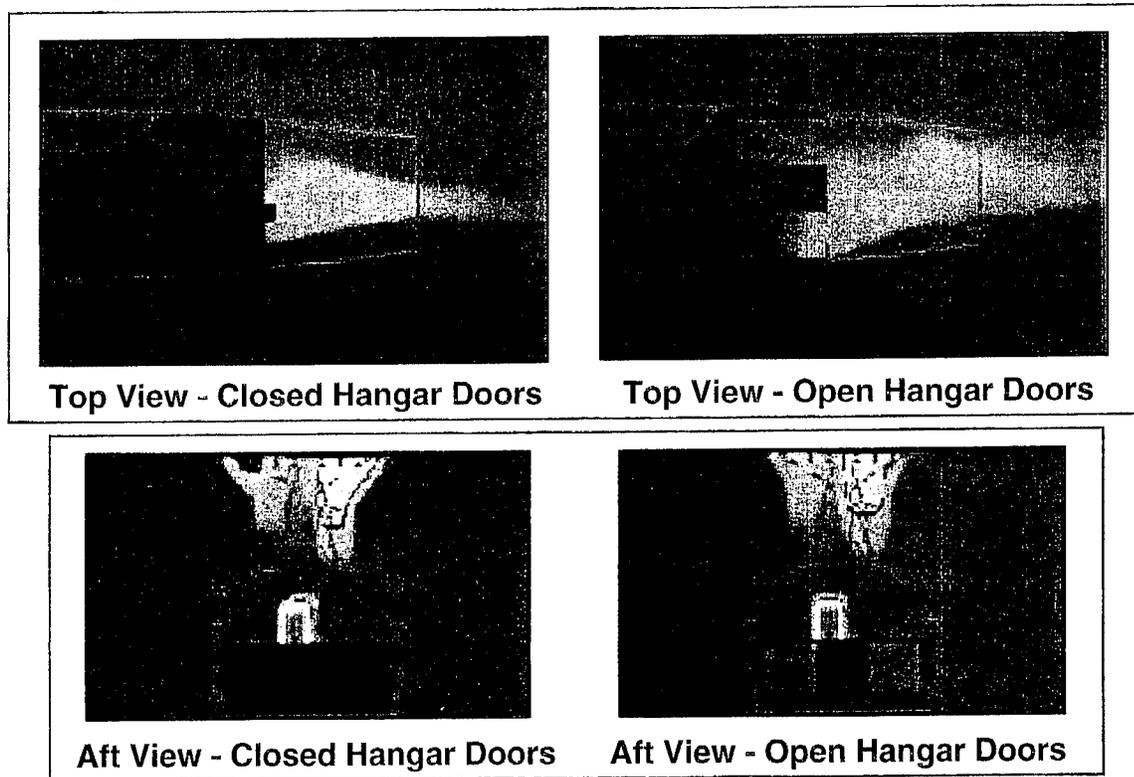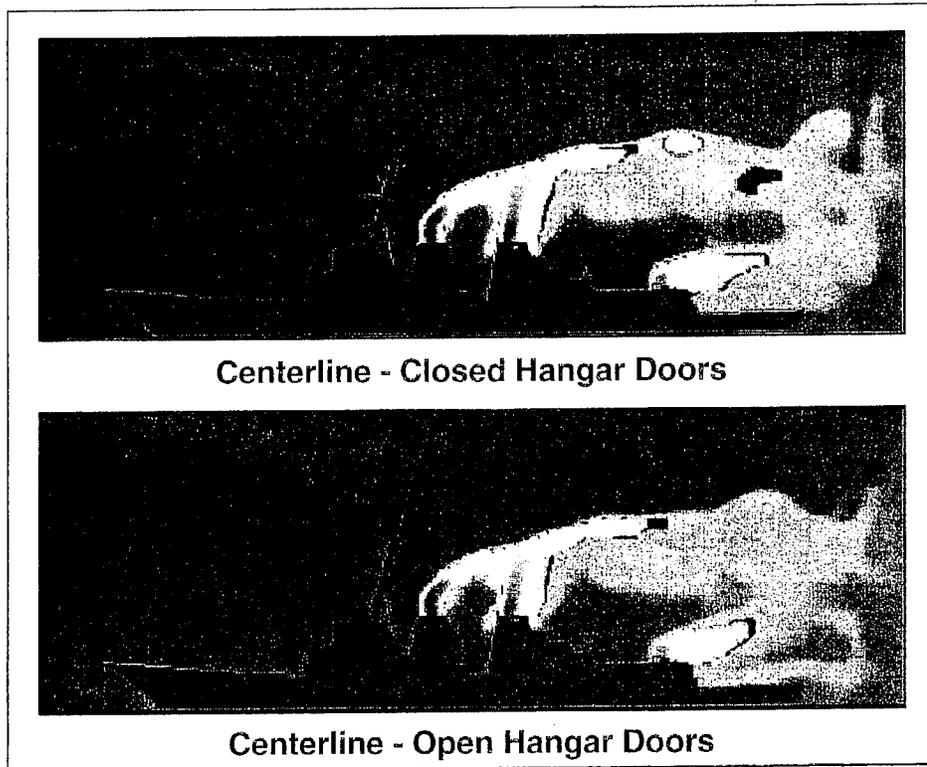
Fig. 6 — Plots of the stack gas distribution for the cases with the hanger doors both open and closed. From Ref. 21. (Used with permission.)

There are minimal debugging facilities on most scalable architectures, which implies that the code developer should follow certain precautions. Interactive, real-time graphics capability is a necessity. It is almost impossible to develop and debug codes of the size discussed here without such a capability. The user/developer should write periodically all primary data, including complex geometry information, from a selection of cross-sectional planes to a file on a workstation. All three planes, or portions thereof, could be viewed without making the files unmanageable. Displays of the evolving flow could be displayed on terminals for both diagnostic and debugging purposes.

Many geometries are extremely complex and require CAD packages that construct the shell(s) of the object(s) being modeled with a series of plates. The ability to rapidly grid complex geometries can entail initial reliance on serial architectures. Thus, it is more efficient to keep the data structures of both the serial and parallel versions of the code the same.

Significant thought must be given to the overall data structure. Typically, very large and very long production calculations stem from scalable code development [22]. In addition, interaction with other users of the same machines may necessitate reducing the number of nodes available during heavy use periods. The data structure should be designed in such a manner as to facilitate check-point restarting, provide fault tolerance for very long run times, allow for changes in the number of physical nodes used in different segments of a single simulation, and ensure that dump-restart files are compatible with other serial processors required for initializing complex geometry cases and postprocessing the simulation results [11].

## ACKNOWLEDGMENTS

## REFERENCES

1. W.L. Camp and S.J. Plimpton, "MIMD Massively Parallel Methods for Engineering and Science Problems," Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, 1993.

2. P.L. George, *Automatic Mesh Generation* (J. Wiley & Sons, New York, 1991).

3. D. Roose, J. DeKeyser, and K. Lust, "Run-Time Load Balancing of CFD Applications on Adaptively refined Grids," Parallel CFD '94 Proceedings, May 16-19, Kyoto Research Park, Japan.

4. S.L. Johnson and K.K. Mather, "Data Structures and Algorithms for the Finite Element Method on a Parallel Supercomputer," *Int. J. Numer. Meth. Eng.* **30**, 881 (1990).

5. S.L. Krist, J.L. Thomas, W.L. Sellers, and S.A. Kjelgaard, "An Embedded Grid Formulation Applied to a Delta Wing," 28th Aerospace Sciences Meeting, Reno, NV, AIAA Paper 90-0429 (1990).

6. A. M. Landsberg, T. R. Young, and J. P. Boris, "An Efficient, Parallel Method for Solving Flows in Complex Three-Dimensional Geometries," 32nd Aerospace Sciences Meeting, Reno, NV, AIAA Paper 94-0413 (1994).

7. R. Ramamurti and R. Lohner, "A Parallel Implicit Incompressible Flow Solver Using Unstructured Meshes," NRL Memorandum Rept. 6410-93-7178, Naval Research Laboratory, Washington, DC, 1993.

8. R. Ramamurti, W. Sandberg, and R. Lohner, "Evaluation of a Three Dimensional Finite Element Incompressible Flow Solver, 32nd Aerospace Sciences Meeting, Reno, NV, AIAA Paper 94-0756 (1994).

9. D.R. Gardner and H.E. Fang, "Modeling Shock Wave Physics problems with PCTH on the Paragon," MPCRL Research Bulletin, 4(1), Massively Parallel Computing Research Lab., Sandia National Laboratory, Sandia, NM, January 1994.

10. J.M. McGlaun, S.L. Thompson, and M.G. Elrick, "CTH: A Three-Dimensional Shock Wave Physics Code," *Int. J. Impact Eng.* 10, 351 (1990).

11. T.R. Young, A.M. Landsberg, and J.P. Boris, "Implementation of the FULL 3D FAST3D (FCT) Code Including Complex Geometry on the Intel iPSC/800 Parallel Computer," SCS Simulator Multiconference, San Diego (1993).

12. J. Donea, S. Giuliani, H. Laval, and L. Quartapelle, "Solution of the Unsteady Navier-Stokes Equations by a Fractional Step Method," *Comp. Meth. Appl. Mech. Eng.* 30, 53 (1982).

13. A.J. Chorin, "Numerical Methods of the Navier-Stokes Equations," *Math. Comp.* 22, 745 (1968).

14. D. Martin and R. Lohner, "An Implicit Linelet-Based Solver for Incompressible Flows," 30th Aerospace Sciences Meeting, Reno, NV, AIAA Paper 92-0668 (1992).

15. R. Lohner and P. Parikh, "Three-Dimensional Grid Generation by the Advancing Front Method," *Int. J. Num. Meth. Fluids* 8, 1135 (1988).

16. E.D. Dahl, "Mapping and Compiled Communication on the Connection Machine," Proceedings of the Distributed Memory Computing Conf V, IEEE Computer Society Press, April, 1990.

17. R. Lohner, R. Ramamurti, and D. Martin, "A Parallelizable Load Balancing Algorithm," 31st Fluid Dynamics Meeting, Washington, DC, AIAA Paper 93-0061 (1993).

18. A.C. Robinson, "Shock Physics," MPCRL Research Bulletin, Massively Parallel Computing Research Lab., Sandia National Laboratory, Sandia, NM, February 1991.

19. E.S. Hertell, Jr, "CTH/PCTH Status," 13th Semiannual Meeting TCG I, Computational Mechanics and Materials Modeling, China, Lake, 1994.

20. J.P. Boris and D.L. Book, "Flux Corrected Transport 1. SHASTA, A Fluid Transport Algorithm That Works," *J. Comp. Phys.* 11, 38 (1973).

21. A.M. Landsberg, J.P. Boris, W. Sandberg, and T.R. Young, Jr, "Analysis of the Nonlinear Coupling Effects of a Helicopter Downwash with an Unsteady Airwake," AIAA Paper 95-0047, Reno, NV (1995).

22. DoD HPC Modernization Office, HPCM Pub 95-001, *Contributions to DoD Mission Success from High Performance Computing—March 1995,* edited by Leland Williams,.