A HETEROGENEOUS GRAPHICS PROCEDURE FOR VISUALIZATION OF MASSIVELY PARALLEL SOLUTIONS*

Patricia J. Crossno
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico

Douglas D. Cline
Center for High Performance Computing
University of Texas System
Austin, Texas

Jeffrey N. Jortner
Massively Parallel Computing Research Laboratory
Sandia National Laboratories
Albuquerque, New Mexico

ABSTRACT

Scientific visualization is playing an increasingly important role in the analysis and interpretation of massively parallel CFD simulations due to the enormous volume of data that can be generated on these machines. In this paper we will describe the development of a visualization technique based on a parallel analogue to the *Marching Cubes* algorithm. The algorithm has been developed for Multiple-Instruction, Multiple-Data (MIMD) massively parallel computers and is designed to take advantage of the heterogeneous programming capabilities of the MIMD architecture. We examine several different configurations and conclude that for producing animations the best one, in terms of both frame generation time and disk usage, is to run the two applications heterogeneously and send the resulting geometry description directly to a workstation for rendering, thereby totally eliminating the use of files from the animation process.

INTRODUCTION

Massively parallel supercomputers offer scientists and engineers the ability to perform large-scale simulations at unprecedented speed. The term *large-scale* implies the solution of field equations in computational domains exceeding several million points of resolution. These simulations can be performed *in-core* due to the large distributed memories which reside on each processor. In CFD applications, the storage of three-dimensional field data may require several Gbytes of disk space. On massively parallel systems, these fields are stored on distributed file systems, with subdomain data spread over an ensemble of parallel disks. For visualization, these fields must be combined with respect to some global coordinate system in order to reconstruct the global computational domain. For distributed data sets on the order of five million mesh points or more, several hours of post-processing may be required for the reconstruction process to generate data files compatible with commercial visualization software systems.

In many MIMD implementations the tendency is to ignore the potential for running truly different application codes on the various processors. Although in a typical MIMD application, different instructions will be executing asynchronously on different processors, the same instruction set is loaded onto all the nodes allocated in the partition. MIMD machines have the capability for conducting *heterogeneous* programming. That is, each subset of processors executes different program instructions, with message passing providing the interface between the applications.We have implemented a CFD application that both generates the volume (flowfield) data and constructs an isosurface in parallel using different programs on different subsets of nodes.

MIMD HARDWARE CONFIGURATION

The heterogeneous analysis was carried out on the Intel iPSC/860, a 64 node hypercube configured with 32 Mbytes of memory in each of the lower 16 nodes, and 8 Mbytes of memory in each of the remaining nodes. The i860 has a 64-bit RISC processor, a floating point adder, and a floating point multiplier. The data cache for each node on the current machine is 8 Kbytes with a corresponding instruction cache of 4 Kbytes. Processors on this machine have a rated clock speed of 33 MHz for a peak performance of 66 MFLOPS in 32-bit arithmetic and 50 MFLOPS for 64-bit operations. Nearest neighbor communication can occur at 2.8 Mbytes/sec each direction for a total bandwidth of 5.6 Mbytes/sec in full duplex. Network latency is measured at 10 microseconds for message start-up with an addition of 10 microseconds of latency for each non-neighbor a message encounters along a given communication path. The system also contains four I/O nodes, each attached to two 760 Mbyte disks. These disks are the central component of Intel's *Concurrent File System* (CFS) which provide for parallel mass storage.

* Accepted for presentation at ASME Fluids Engineering Conference, June 20-24, 1993 Washington, D.C.

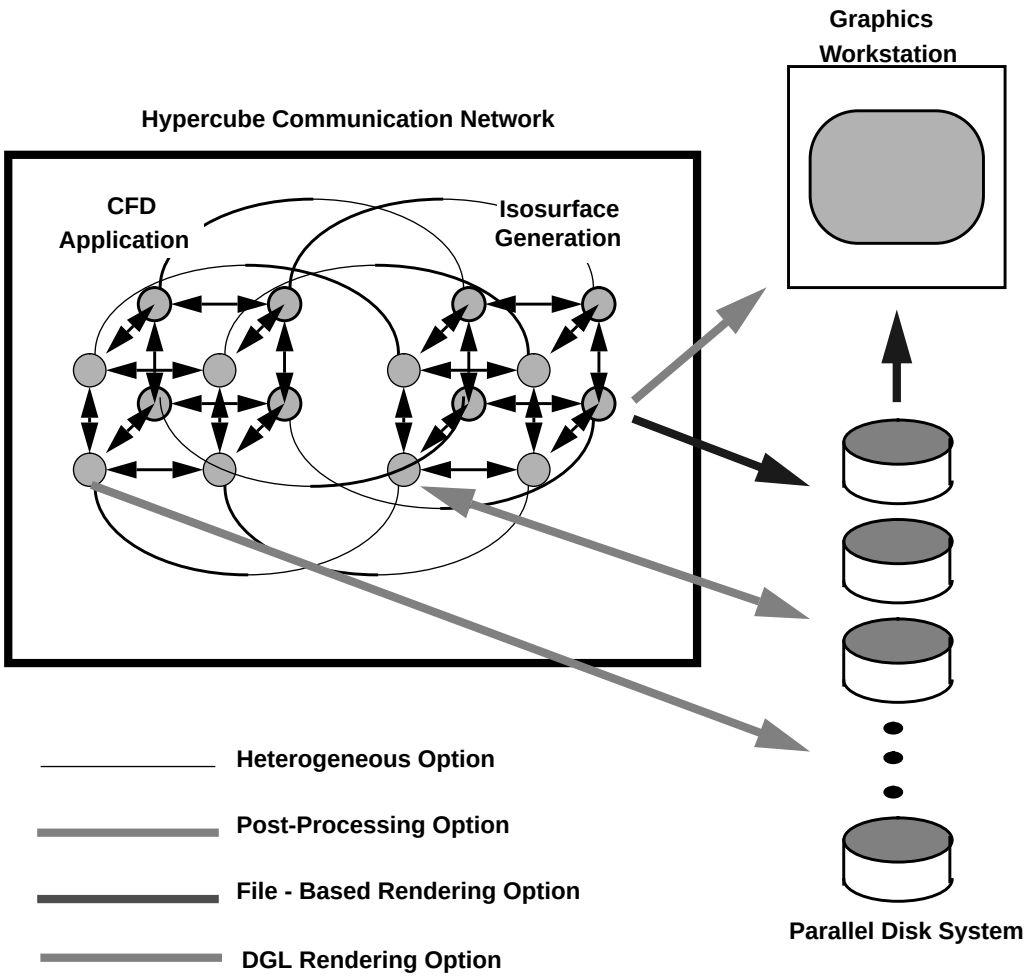

FIGURE 1. CONFIGURATION OPTIONS FOR 16 PROCESSOR MIMD HETEROGENEOUS DECOMPOSITION OF CFD AND GRAPHICS APPLICATIONS.

HYPERCUBE DECOMPOSITION

Our implementation does not typically use the entire machine, but rather utilizes two subcubes of equal size which are allocated as a single cube at runtime to enable message passing between the two applications. Each subcube loads and runs an entirely different executable. The heterogeneous configuration of the CFD code and isosurface generator is depicted in Figure 1. The lower subcube executes the CFD application that generates the field data, while the upper subcube thresholds the field data and stores the geometry files on the parallel disk system. Internode communication for each application is illustrated by dark arrows between adjacent processors. Subcube communication is denoted by the thin lines which connect the two applications. Because of the hypercube topology, processors in the two subcubes reside as *nearest-neighbors* along the communication channels. Thus the sending of CFD field sets to the isosurface generation code can be performed with minimal communication overhead. This also avoids the bottleneck inherent in a scheme which would use a front end to pass the volume data between the two applications.

After some number of time steps, each processor running the CFD code passes its sub-volume of data to the corresponding node in the upper half of the subcube along with a threshold value. The upper half then uses the marching cubes algorithm [Lorenson & Cline, 1987] to find the isosurface with that threshold within the volume. Geometry files generated by the marching cubes algorithm are then stored on the parallel disk system for subsequent display on a graphics workstation. Having sent its data, the CFD code continues execution on the lower subcube for the next time step. Thus, a pipeline is formed between the generation of the volume data, the generation of the isosurface, and the parallel disk system, thereby permitting concurrent operation. This capability is especially useful when a large number of volumes need to be rendered, such as when performing fluid flow animation. An alternative means of interfacing the two applications consists of having

the CFD code write field data directly to the parallel disk for subsequent post-processing by the isosurface generator.

There are a number of options for outputting the isosurfaces produced. Note that the application interface selected (heterogenous or file-based) is independent of the output option. There are two file-based output options from the isosurface generator, one to the CFS and one to the file system associated with the front end. Although the processors can write to the CFS in parallel and hence output the isosurface data more quickly, files on the CFS must be moved off of the CFS for viewing, which is a very slow operation. Consequently, writing to the CFS is best done only in those circumstances when the data can remain on the CFS, such as when the CFD code outputs volume data while running in the non-heterogeneous mode.

The geometry description created by the marching cubes algorithm may also be directly rendered using *Distributed Graphics Library* (DGL) commands. DGL is a Silicon Graphics' product which is available for a number of hosts. Intel has ported DGL to the iPSC/860. From a programming perspective, DGL commands are essentially identical to Silicon Graphics Iris (SGI) Graphics Library (GL) calls, except that lower level software sends the commands via ethernet to the SGI for execution and display. Using this approach, images may be directly displayed and recorded for animation, bypassing file output and the parallel disk system entirely.

Another configuration consists of replacing the DGL output module with a module which creates a socket connection between the Intel and a program running on the SGI. Triangle data is then sent over the socket to the SGI where the local program renders it using GL. Spaceball Technologies, Inc. produces an input device, the Spaceball, which permits simultaneous manipulation of all six degrees of freedom in a 3-dimensional space. The Spaceball connected to our SGI is used to control the viewpoint as the simulation progresses. One of the buttons on the Spaceball is used as a toggle to start and stop the simulation. After each timestep's isosurface is sent to the SGI, the Intel waits for a handshake before proceeding to the next timestep. The button disables the handshake, thereby permitting the researcher to fly around the isosurface and view interesting structures without fear of losing them in the next frame update. Additionally, another button on the Spaceball allows the researcher to take screen dumps of the simulation at the current viewpoint and save them to AVS compatible image files for later viewing.

The final output option uses the same output module on the Intel as the Spaceball configuration, but the program on the SGI is replaced with an *Application Visualization System* (AVS) coroutine. AVS is a commercial software product for scientific visualization which is supported on a number of workstation platforms. The coroutine establishes a socket connection with the Intel, receives the triangle lists, translates them into AVS's geometry format, and passes that geometry data on to another AVS module known as the geometry viewer. The geometry viewer then controls the rendering and viewing of the isosurfaces. It also provides a way to save geometry and scene files for later viewing. These files contain 3-dimensional descriptions of each object as well as the camera position, material characteristics, and lighting properties of the scene.

MIMD MARCHING CUBES ALGORITHM

The marching cubes algorithm approximates the isosurface with a set of triangles. Each neighborhood of 8 sample points, forming the 8 vertices of a cube, are operated on as a unit. The isosurface within a cube is found by comparing the values at the vertices with the surface value and using the results as an index into a table of surface triangulations for each configuration. The vertices of the triangles are then found by interpolation along the edges between the cube vertices. The evaluation of the cube "marches" through the volume, first by column, then by row, then by plane.

The adaptation of the algorithm to a MIMD environment is fairly straightforward due to its inherent data parallelism. The decomposition is accomplished by partitioning the volume into approximately equal-sized sub-volumes, which are distributed to the individual processors, each of which is asynchronously running an essentially serial version of marching cubes.

MIMD SOFTWARE ISSUES

The heterogenous implementation brings several software issues to light: having one node operate differently from the others, in effect as a host node; using global utilities that operate across all of the nodes; and using a broadcast mode for communicating information from one node to all of the other nodes. In the first case, a typical implementation will designate node zero as performing certain special functions (such as reading or writing to the front end). All of these references must be changed to refer to a base node which is assigned dynamically based on the cube size. The second issue requires more work to resolve. The Intel parallel libraries provide a number of global routines for functions such as synchronization, minimum, maximum, sum, product, AND, OR, and XOR. These routines require that all of the nodes in the allocated cube participate. Since half of the nodes are working on a separate problem, this is not possible. Therefore, these functions must be rewritten by the programmer to operate within subsets of the cube. The Intel does provide a mechanism for broadcasting to a subset of nodes, so the last problem is resolved by simply replacing the broadcast call with a call to the restricted broadcast routine.

EXAMPLE VISUALIZATIONS

As a first example of the visualization technique, we modified the shock hydrodynamics code, PAGOSA/MIMD [Gardner, Cline, and Vaughan, 1992; Gardner and Fang, 1992] to execute in a heterogeneous mode on the Intel iPSC/860. PAGOSA/MIMD is a three-dimensional, combined Eulerian/Lagrangian explicit code which models the advection of an ideal gas. PAGOSA/MIMD is written in FORTRAN-77 and incorporates domain decomposition and message-passing for MIMD, distributed memory parallel computers.

The PAGOSA/MIMD test problem consisted of the 3-D propagation of a spherical blast wave in the presence of an intervening

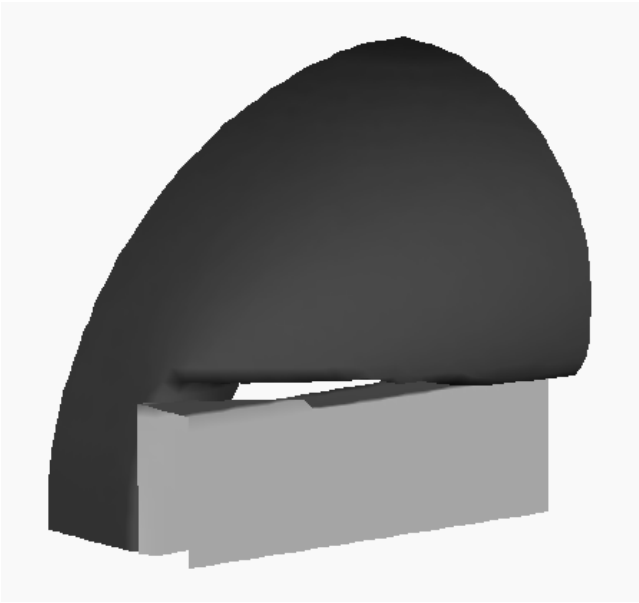

FIGURE 2: PAGOSA/MIMD EXAMPLE VISUALIZATION - BLAST WAVE WITH OBSTACLE USING 16 PROCESSORS

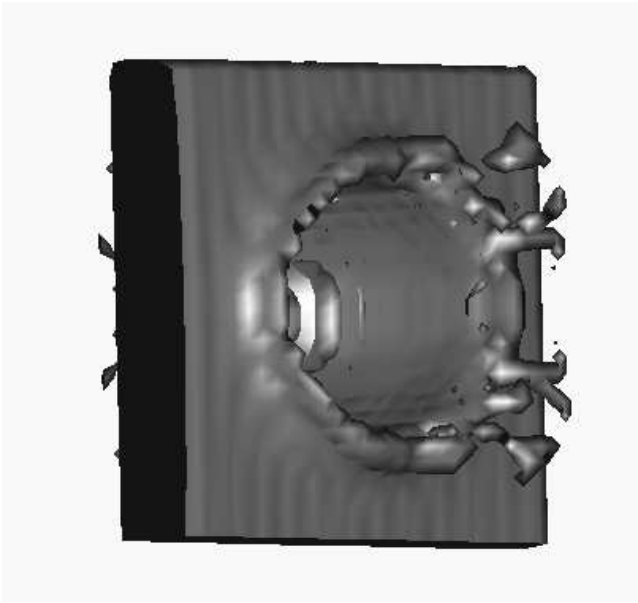

FIGURE 3: PCTH EXAMPLE VISUALIZATION - COPPER BALL IMPACTING STEEL PLATE USING 32 PROCESSORS

obstacle. Figure 2 is an example of a typical isosurface construction, created using the non-heterogeneous configuration in conjunction with the file-based rendering option, showing a blast wave impinging on an obstacle after 64 time steps. Within each application, each of which was run separately, the computational domain was distributed over 16 processors, configured as a 4 x 2 x 2 array. The global domain consisted of 32^3 points uniformly distributed in 3 dimensions over the ensemble of processors. The threshold value for the 3-D contour of the blast wave was 1.15 and the threshold for the barrier was .0001.

As a second example of the visualization technique, the shock hydrodynamic code PCTH [Gardner and Fang,1992; Robinson et al., 1991] was executed heterogeneously on the INTEL iPSC/860. PCTH is written in C++ for MIMD computers and is an explicit Lagrangian/Eulerian code designed to study the extreme deformation of solid materials due to high velocity impact. The present code models multiple materials, strong shocks, material strength and fracture.

The PCTH test problem consisted of the 3-D hydrodynamic deformation of a steel plate when impacted obliquely by a copper ball traveling at 4.5 km/sec (Figure 3). The figure was created using the heterogeneous configuration in conjunction with the AVS-based rendering option. The computational domain of the CFD application was distributed over 8 processors configured as a 2 x 2 x 2 array; the isosurface generator was similarly configured within its 8 processor subcube. The global domain consisted of 32^3 points uniformly distributed over the ensemble of processors within each subcube. Unlike the previous example, where a single volume is thresholded twice using different values to distinguish the two objects, the present example requires the construction of isosurfaces for two sets of volume data corresponding to the volume fractions of each of the materials present in the domain. The visualization also demonstrates the transportability of the present methodology to other parallel application codes.

TIMING ANALYSIS OF VISUALIZATION OPTIONS

In order to evaluate the comparative resource usage of each of our configurations, we performed an experiment wherein each visualization option described above was timed using the same volume data. In each case our timings represent the time necessary to take the simulation from volume creation in the CFD analysis to rendering the final image on the screen. As a test problem, we used PAGOSA/MIMD to visualize the blast wave problem depicted in Figure 2 on a computational domain of 32^3 grid cells.

Sixteen processors of the Intel iPSC/860 were utilized as a fixed resource. For heterogeneous visualization, 8 nodes were assigned to the PAGOSA/MIMD application, while 8 nodes were responsible for constructing the isosurfaces. In contrast, for file-based options, all 16 processors participated in the blast wave simulation, with distributed files for the volume data written to the CFS. Similarly, all 16 processors were utilized in the isosurface construction, reading the previously saved volume data on the CFS. Since the processors are treated as a fixed resource, pipelining of the two applications is not possible, and the two applications are timed sequentially. The timings were averaged over 10 timesteps of the simulation ranging from step 60 to step 70. Additionally we measured storage requirements for both the volume data obtained from PAGOSA/MIMD and the triangle list produced by the isosurface application. The outputs from both applications are stored without compression in a binary file format. The results of this study are summarized in Table 1.

TABLE 1: PERFORMANCE TIMINGS FOR RENDERING 32^3 FIELD DATA (SEC.)

Option	Description of Measurement	Avg. Time per Timestep	Total Time per Timestep	Total Disk Space Usage per Step (MB)
Heterogeneous with 8 processors assigned to each application	Time to render solution with DGL	2.91	2.91	0
Heterogeneous with 8 processors assigned to each application	Time to render solution with GL using socket connection and Spaceball	3.04	3.04	0
Heterogeneous with 8 processors assigned to each application	Time to render solution with socket connection and AVS	4.47	4.47	0
PAGOSA/MIMD with CFS output, Marching Cubes with CFS input, file output to front end with AVS (16 proc.)	Time for CFD code Time for Isosurface code AVS display time	2.20 6.13 3.00	11.33	0.47
PAGOSA/MIMD with CFS output, Marching Cubes with CFS input & CFS output (16 proc.)	Time for CFD code Time for Isosurface code Copy to front end AVS display time	2.20 3.56 10.40 3.00	19.16	0.47
PAGOSA/MIMD with CFS output, Marching Cubes with CFS input & DGL output (16 proc.)	Time for CFD code Time for Isosurface code, rendering and display using DGL	2.20 5.77	7.97	0.21

The thicker line in Table 1 delineates the heterogenous options from the file-based, post-processing options. The disk space usage consists of two contributions. For each timestep. 0.21 MB are required to store the field volumes created by the PAGOSA/MIMD application. The isosurface reconstruction requires an additional 0.27 MB each timestep to store the list of triangles required to construct the isosurface.

The performance of the heterogeneous application options clearly exceed the performance of the file-based options. This performance can be measured both in terms of saving time as well as disk space resources. In general, it can be seen that file based output from the isosurface construction is poor and suffers from serious bottlenecks. Outputting to a single file on the front end creates a bottleneck since all of the data must be serially channeled through a single node which does the writing to the file. As a second file-based alternative, geometry files may be written on the parallel disk system by having each processor concurrently write to separate sections of the same file. This eliminates the bottleneck associated with local memory constraints. However, in order to render the data contained in the geometry files, they must be first copied to the front end. In Table 1, for the 32^3 domain, this process took as long as 10.4 seconds per timestep. Compared to other timings in Table 1, this post-processing step is *very* slow. If geometry data is sent directly to the graphics workstation using a socket connection, the overall performance in the post-processing mode is the best of all procedures. Approximately 0.27 MB of disk space are saved each timestep when the flowfield data are post-processed using the socket connection.

CONCLUSIONS

Our principal result is a tremendous reduction in disk storage costs and time savings. As an example of the magnitude of this savings, consider just the size of the volume data files created by a CFD application which creates a single volume per timestep for a 30 second animation of a 64^3 domain. For a 32 node decomposition, the number of bytes across all of the sub-volumes (including overlapping points between processors and assuming 4-byte values) would be 32 x 18 x 18 x 34 x 4 = 1410048 bytes per frame. Assuming a frame rate of 30 frames per second (standard NTSC rate), the volume data for the entire animation would require 1.27 Gbytes to store the volume data alone. This is not a large volume or a long animation sequence. A volume of 256^3 would require 65.2 Gbytes of storage for the same sequence length. For CFD

applications which use volume fractions, multiply the storage costs by the number of materials.

Additional storage savings are possible if the images are rendered directly to the screen. However, these savings are more difficult to calculate because the size of the geometry file depends on the surface threshold selected and the data values present in the volume. Conceivably, a threshold could be selected which would not find a surface and the file would be of zero length. To provide some sense of the size of a typical geometry file, the file used to render Figure 2 is 0.26 Mbytes in length.

Each stage of the pipeline from volume creation to rendered image can be viewed as a filter which reduces the amount of data. For instance, once a volume is thresholded, unless the original volume data is saved, only the selected isosurface can be rendered. Our work and our results have been geared towards doing animations quickly and efficiently, so the loss of the volume and isosurface data is not an issue. However, for those who need to preserve intermediate results, our various options for the applications interface and the isosurface output permit researchers to save the data at each stage, assuming that they have sufficient disk space to do so. The advantage of this filtering process is the potential to view simulations which have been previously too large for conventional workstations to visualize. As massively parallel computers become more available, these type of large-scale simulations will be even more routine, making the visualization procedures we have outlined here increasingly important.

FUTURE WORK

There are a couple of enhancements we would like to make to the current visualization system. First, we would like to improve the load balance between the CFD code and the isosurface code by removing the restriction that the CFD nodes be one-to-one with the isosurface nodes. As the physics of the CFD applications increase in complexity, the processor resources should be concentrated in the CFD portion of the code. Second, we would like to move the rendering of the isosurface onto the parallel platform to further reduce the volume of data being moved off of the parallel machine. As the size of simulations increase, the data reduction resulting from the isosurface filter will be insufficient to keep the socket connection from creating a serious bottleneck. Further filtering down to a pixel representation will be necessary

Also the present work raises the question of whether it is computationally more efficient to visualize CFD applications using the present heterogeneous scheme, or whether the computational resources would be more effectively utilized by distributing the application over twice the number of nodes, and calling the graphics procedure as a subroutine. Many conflicting factors can tilt the balance towards preferring one technique over the other. Competing memory requirements between the CFD application and the graphics application may make the subroutine approach infeasible due to the limited size of the local memories. On the other hand, in the heterogenous approach, load imbalance results from the differences between the time required by the CFD application to generate a volume and the time spent by the isosurface application to threshold it.In the timings presented in Table 1, the hydrodynamic application code runs faster than the isosurface generation code. Even though timings for the multi-material application were not included in our experiment, we have observed in our use of that application that this load imbalance is reversed and the isosurface nodes are largely idle. This is due to the more extensive modelling of the multiple materials in this application. Further experimentation and comparisons between the heterogenous approach and a subroutine approach will be required in order to assess the tradeoffs.

ACKNOWLEDGMENTS

This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under Contract DE-AC04-76DP00789. The PAGOSA/MIMD development work was supported under the Joint DoD/DOE Munitions Technology Development Program, and sponsored by the Office of Munitions of the Secretary of Defense.

We would also like to acknowledge the work of both the PAGOSA/MIMD and PCTH development teams whose members include: David R. Gardner, Allen C. Robinson, H. Eliot Fang, Courtenay T. Vaughan, Arlo Ames, Dino Pavlakos, Phil Campbell, Ray Bell, James Peery, Mike Wong, and Diane A. Verner.

REFERENCES

Gardner, D. R., Cline, D., and Vaughan, C. T., 1992, “Implementation of a Single-Material Version of PAGOSA on MIMD Parallel Computers,” SAND 92-0640, Sandia National Laboratories, Albuquerque, NM.

Gardner, D. R., and Fang, H., 1992, “Three-Dimensional Shock Wave Physics Simulations on Massively Parallel Supercomputers,” *Proceedings of the 1992 Summer Computer Simulation Conference*, Reno, Nevada, pp. 537-541.

Lorenson, W., and Cline, H., 1987, “Marching Cubes: A High Resolution 3D Surface Construction Algorithm,” *Computer Graphics (ACM Siggraph Proceedings)*, Vol. 21, No. 4, pp. 163-169.

Robinson, A. C., Ames, A. L., Fang, H. E., Pavlakos, D., Vaughan, C. T., and Campbell, P. M., 1992, “Massively Parallel Computing, C++ and Hydrocode Algorithms,” *Proceedings on Computing in Civil Engineering, 8th Conference in conjunction with A/E/C/ Systems ‘92*, TCCP/ASCE, Dallas, TX.

Robinson, A. C., Vaughan, C. T., Fang, H. E., Diegert, C. F., and Cho K. -S., 1991, “Hydrocode Development on the NCUBE and the Connection Machine Hypercubes,” *Shock Compression of Condensed Matter 1991*, pp. 289-292.