Conf-950788--1

SAND95-0095C

The Development and Application of Massively Parallel Solid Mechanics Codes

Mike McGlaun, Allen Robinson and James Peery

Sandia National Laboratories, Albuquerque, New Mexico, U.S.A.

1. INTRODUCTION

Computational physicists at Sandia National Laboratories have moved the Eulerian CTH code, and the arbitrary-Lagrangian-Eulerian ALEGRA code to distributed memory parallel computers. CTH is a three-dimensional solid mechanics code used for large-deformation, shock wave analysis [1,2]. ALEGRA is a three-dimensional arbitrary-Lagrangian-Eulerian solid-mechanics code used for coupled large-deformation, shock and structural mechanics problems [3]. This paper discusses our experiences moving the codes to parallel computers, the algorithms we used and our experiences running the codes.

2. PARALLEL COMPUTERS

We moved our CTH and ALEGRA codes to massively parallel computers because we needed their enormous memories and processor speeds to analyze large, three-dimensional problems. The memory requirements for our codes scale inversely as the cube of the zone size. For example, if we halve the element size in each direction, then the memory requirement increases by a factor of eight. These codes use explicit, time-stepping integration schemes so the time step scales inversely as the mesh size. For example, if we halve the element size, then the code decreases the time step in half. Therefore, the Floating Point OPerations (FLOPs) scale as the fourth power of the mesh size. Since the FLOP requirements increase faster than the memory requirements, simply increasing the memory on existing supercomputers is not a good solution to running larger problems because the run time quickly becomes excessive.

Our codes run on distributed-memory parallel computers that are constructed from multiple compute nodes, as shown in Figure 1. Each compute node has a central processing unit (CPU), memory and may have special hardware such as a vector processor or input/output processor. A high speed communication network connects the compute nodes. This computer model fits many parallel computers including Sandia's 1840 compute node Intel Paragon, 1024 compute node nCUBE2, networks of workstations, and even shared-memory parallel computers like the Cray Y-MP.

We designed our codes to easily switch the inter-node communication software. We have run with PVM3, MPI, and vendor specific message passing interfaces. We use the small size, high-performance Sandia/University of New Mexico Operating System (SUNMOS) [4]. It maximizes the user memory, provides the needed functionality, and provides very high speed communications.

This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000.

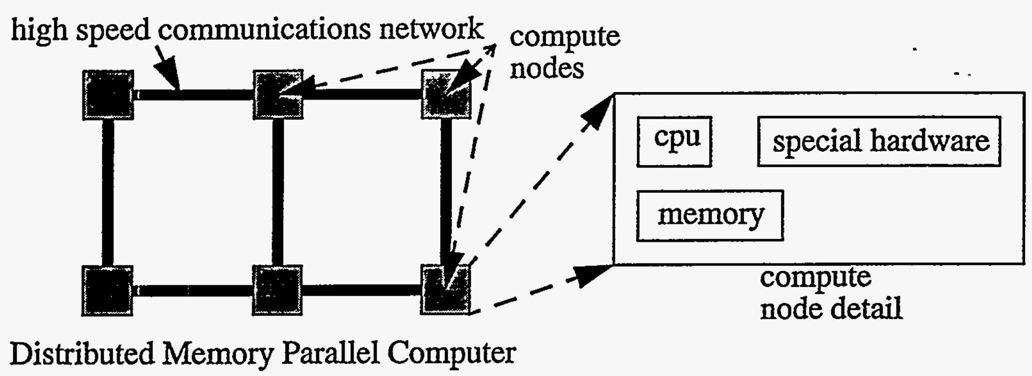
high speed communications network
compute nodes
cpu
special hardware
memory
compute node detail
Distributed Memory Parallel Computer

FIGURE 1: DISTRIBUTED MEMORY PARALLEL COMPUTER

3. THE CTH AND ALEGRA CODES

CTH models multidimensional, multi-material, large deformation, shock wave physics using a finite-volume numerical technique [1,2]. The massively parallel version of CTH supports only three-dimensional rectangular meshes. A rectangular mesh, as shown in Figure 2, is constructed from parallel lines. ALEGRA models multidimensional, multi-material, transient solid mechanics physics using a finite element numerical technique [3]. Three and two dimensional arbitrary connectivity meshes, as shown in Figure 2, are available. Arbitrary connectivity meshes allow an arbitrary number of elements to share a common element node.

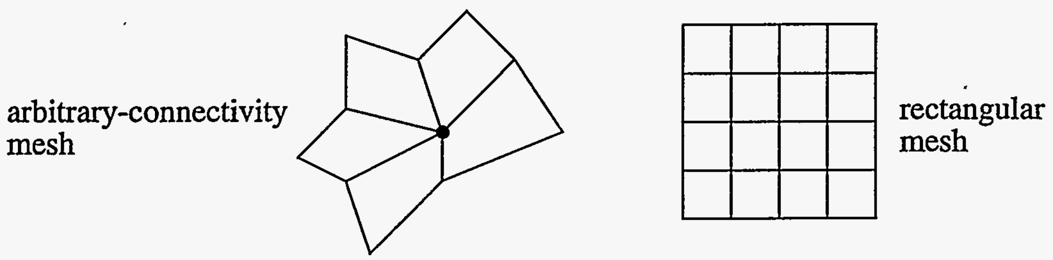

FIGURE 2: ARBITRARY-CONNECTIVITY AND RECTANGULAR MESH

We use the Single Program Multiple Data (SPMD) model where the same program runs on all compute nodes but each compute node has a different data base. For example, we run 1840 identical copies of our code on the Paragon but each copy uses a different data base. The programs communicate using explicitly passed messages.

3.1 Mesh Decomposition

The data base of a large, three-dimensional problem is too large to fit on any single compute node. We address this problem by breaking the data base into several sub-data bases. In particular, we break the entire mesh into several sub-meshes. This is shown in Figure 3. Each compute node gets a different sub-mesh. The compute node's data base is the sub-mesh's data base. We decompose the mesh so that each sub-mesh has approximately the same number of elements. Assuming equal work per element, equal sub-meshes result in the same amount of work for each compute node.

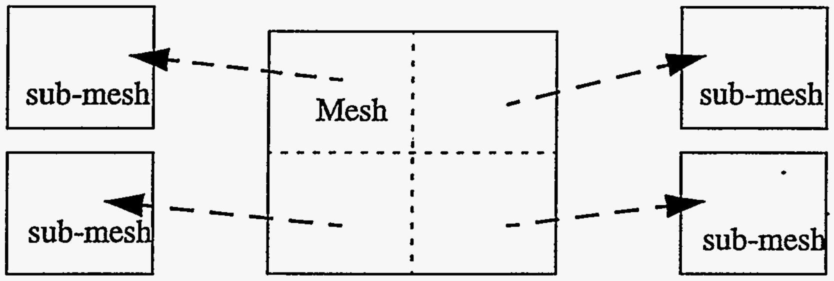

FIGURE 3: MESH AND SUB-MESHES

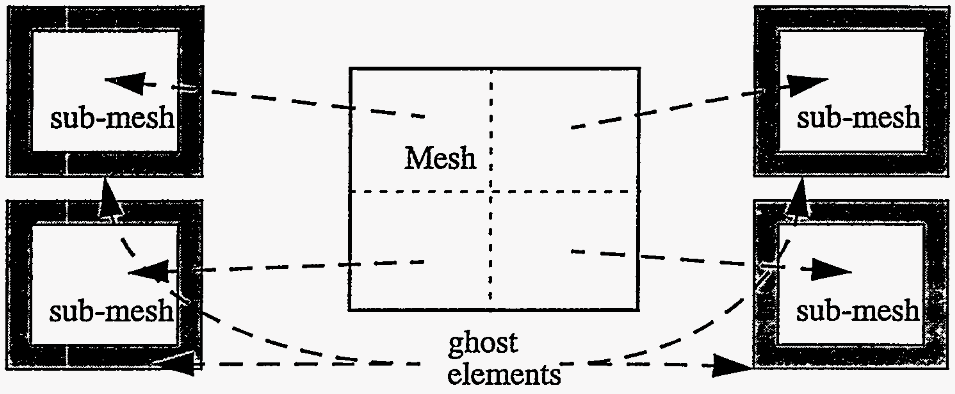

FIGURE 4: GHOST ELEMENTS ON SUB-MESH BOUNDARY

CTH uses one layer of ghost elements around each sub-mesh perimeter for boundary conditions, as shown in Figure 4. The additional ghost elements represent a parallel processing cost that can be quite large for compute nodes with a small number of elements. For example, assume a compute node holds 10 x 10 x 10 = 1,000 elements. Then $10^3 - 8^3 = 488$ elements (almost half) are boundary elements. If the compute node holds 100 x 100 x 100 = 1,000,000 elements, then $100^3 - 98^3 = 58,808$ elements (about 6%) are boundary elements. We use a sub-mesh that fills the compute node's memory to minimize the number of ghost elements. Minimizing the number of boundary elements also minimizes the amount of data passed between compute nodes.

ALEGRA does not use ghost elements. The element nodes on sub-mesh boundaries are replicated between compute nodes. This is shown in Figure 5.

Good mesh decomposition is important to minimize the memory requirements, balance the work on the compute nodes and minimize the data passed between compute nodes. Rectangular meshes are relatively easy to decompose. Arbitrary connectivity meshes are much more difficult to decompose. We decompose the meshes using Sandia's Chaco [5] library of routines.

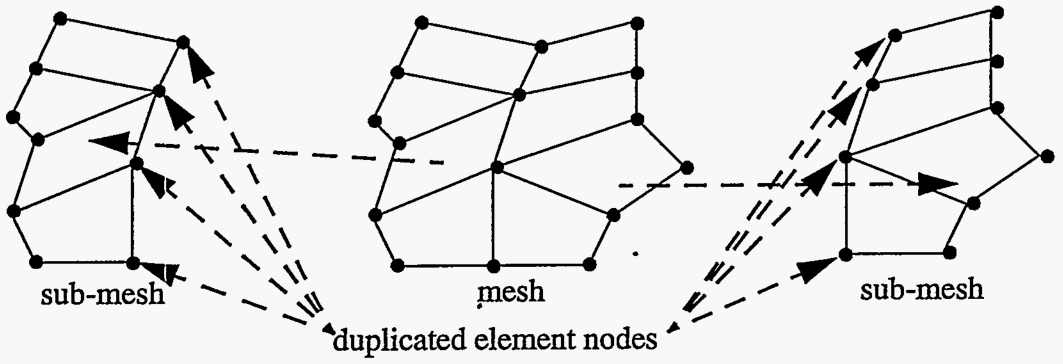

FIGURE 5: ARBITRARY CONNECTIVITY MESH DECOMPOSITION

3.2 Algorithm Modifications

We must modify the solution algorithm for parallel processing. The modifications are relatively simple for our codes because the governing equations are hyperbolic and we use explicit numerical techniques. Extending most of the numerical algorithms to parallel computers is simple because only an element's neighbors influence an element.

One algorithm that must be modified in CTH is calculating the gradient of a field. The gradient uses the cell value and the values of the nearest neighbors. In the parallel version, some of the nearest neighbor values are in neighboring sub-meshes. We solve this problem by sending the field values from the neighboring sub-meshes to the ghost elements before calculating the gradient. This process is shown in Figure 6. We then calculate the gradient in the sub-mesh interior. There are no major changes to the solution algorithms used with uni-processor computers. The major change is transmitting data between sub-meshes before the gradient operation.

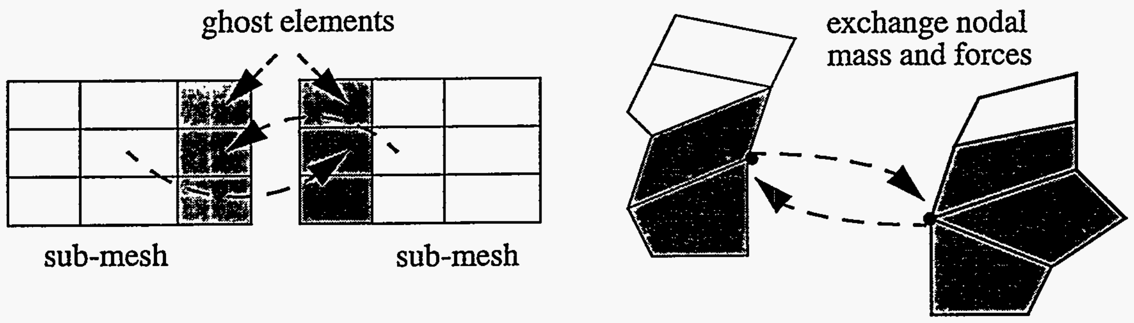

FIGURE 6: SENDING FIELD VALUES TO GHOST ELEMENTS

We must perform a similar modification in ALEGRA. We need to sum the forces and masses on a node to calculate the nodal acceleration. The nodal forces and masses are calculated from the elements attached to the node. First, we sum the forces and masses for the elements in the sub-mesh. Next, if the node is duplicated in another sub-mesh, then the sub-meshes exchange summed forces and masses, as shown in Figure 6. The total force and mass are calculated by summing the exchanged forces and masses.

We must modify our algorithm to calculate fluxes between elements. We use the van Leer algorithm [6] which is a monotone, second-order accurate algorithm. It uses donor element value and the gradients on the sides of the donor. This algorithm is more difficult to extend to parallel computers because it uses both nearest neighbor and next nearest neighbor values. The parallel CTH version of the algorithm uses a two step process. The first step calculates the gradients, passes them to neighboring sub-meshes, and stores them as part of the mesh data base. Using this form of the data base, the solution algorithm now uses the donor cell values and the nearest neighbor values. The second step calculates the flux. The parallel ALEGRA version uses a different approach by copying the field values into a working array even if the values lie in two sub-meshes, as shown in Figure 7.

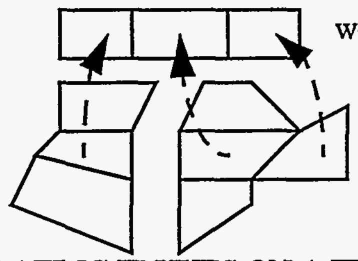

FIGURE 7: CALCULATING FLUXES ON A FINITE ELEMENT MESH

We need to calculate global information such as the calculation time step. The time step algorithm uses the smallest time step in the global mesh. The parallel algorithm first calculates the smallest time step for each sub-mesh. Then an efficient global algorithm calculates the minimum value to be used by all sub-meshes.

3.3 Files

Massively parallel calculations can generate enormous data bases that can result in long input/output times and enormous files. We address this problem by performing parallel output and compacting the files. The restart files contain all the information required to restart the calculations. Our codes write a restart file for each computational node. We perform no compaction of this file. We compress our graphics file. We identify the largest and smallest values in a field, such as pressure. We then linearly or logarithmically map the values into sixteen bit integers. This bins the values into 65,536 values. We then perform run-length encoding. This typically reduces the file size by two orders of magnitude.

Post-processing the compressed files is still very challenging. We can merge the data bases to form a single post-processing file. The data base may be too large to fit into the post-processing computer's memory. We can then analyze only a fraction of the data base by reading a part of the data base (zooming into a region of interest), plotting a subset of the materials, or sampling the plot file, e.g. read every other point.

3.4 Results

The massively parallel version of CTH is being routinely used on the Paragon. The Paragon ran a complex, ten-material problem with explosives and elastic-plastic material models at a rate of approximately 280 Kcell-cycles/second. A single processor of the Cray Y-MP runs a coarser zoned version of the same problem at about 10 Kcell-cycles/second. Recent optimization efforts may increase the speed by a factor of two.

We recently completed a series of eight-million cell CTH calculations of the comet Shoemaker-Levy 9's impact on Jupiter. Figure 8 shows a Jupiter fire ball 55 seconds after comet impact. The outline of the state of New Mexico indicates the size of the fire ball

[7].

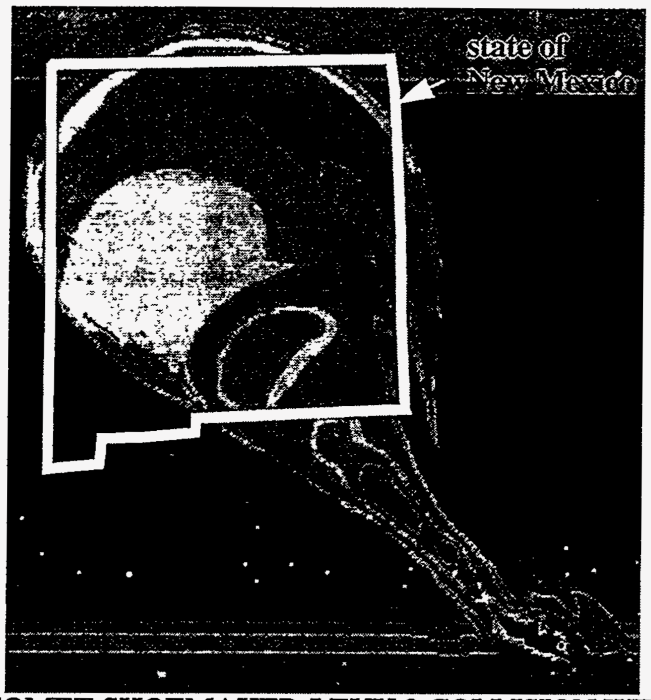

FIGURE 8: COMET SHOEMAKER-LEVY 9 COLLISION WITH JUPITER

4. REFERENCES

[1] McGlaun, J. M., S. L. Thompson, M. G. Elrick, 1990, “CTH: A Three-Dimensional Shock Wave Physics Code,” Int. J. Impact Engng., Vol 10, pp 351-360, 1990
[2] McGlaun, J. M., F. J. Zeigler, S. L. Thompson, M. G. Elrick, 1988, “CTH User's Manual and Input Instructions,” Sandia National Laboratories Report SAND88-0523, Sandia National Laboratories, Albuquerque, NM
[3] Peery, J. S., Budge, K. G., Wong, M. K. W., Trucano, T. G., 1993, “RHALE: A 3D MMALE Code for Unstructured Grids,” Proceedings of the Winter ASME Meeting, New Orleans, LA
[4] Maccabe, A. B., McCurley, K. S., Riesen, R. Wheat, S. R., 1994, “SUNMOS for the Intel Paragon: A brief user's guide,” In *Proceedings of the Intel Supercomputer User's Group. 1994 Annual North America Users' Conference*
[5] Hendrickson, B. and Leland, R, 1993, “The Chaco User's Guide,” Sandia National Laboratories Report SAND93-2339, Sandia National Laboratories, Albuquerque, NM
[6] van Leer, B., 1977. “Towards the Ultimate Conservative Difference Scheme IV. A New Approach to Numerical Convection.” *J. Comp. Phys.* Vol. 23, pp. 276-299
[7] Crawford, D. A., M. B. Boslough, T. G. Trucano, and A. C. Robinson, “The Impact of Comet Shoemaker-Levy on Jupiter,” Shock Waves (1994) 4:47-50