CONF-9604117--1
SAND96-0431C

Shock Physics Code Research at Sandia National Laboratories; Massively Parallel Computers and Advanced Algorithms[1]

by J. M. McGlaun, J. S. Peery and E. S. Hertel
Sandia National Laboratories
Albuquerque, New Mexico, 87185-0819

1. Abstract

Shock physics researchers at Sandia are working in two areas: massively parallel computing and improved solution algorithms. Our goal is predictive modeling of large, three-dimensional problems. We will discuss the goals, rationale and status of this work.

1.1 Massively Parallel Computer Research

Massively parallel computers present both tremendous opportunities and challenges. They can have much more memory and speed than traditional supercomputers. For example, Sandia's Paragon computer has 1840 i860 CPU's, a peak performance of 147 GFLOPs and 37 Gbytes of memory. Intel will install the Accelerated Strategic Computing Initiative (ASCI) TFLOP computer at Sandia in 1996. It will have 9072 Pentium Pro CPU's, a peak performance of 1.8 TFLOPs and 594 Gbytes of memory. Both of these computers are orders of magnitude larger than traditional supercomputers and offer tremendous opportunities to predictively model very large, complex systems.

There are two challenges in developing software for these computers. First, we must write the code to use a distributed mesh. To do this, we must decompose the global mesh into numerous submeshes. Each compute node has its own submesh and copy of the code. The code explicitly passes information to neighboring submeshes as needed. Second, we must write the codes using different data structures and algorithms than we used in the past for vector computers. Massively parallel computers are built from commodity CPU's with hierarchical (cache) memories. Vector computers, such as CRAYs, have single-level memories with very fast bandwidth to memory. Cache memories have a very high bandwidth to the first level of memory (cache), but the bandwidth to main memory is too slow to keep the CPU busy. To obtain high performance, we are designing the codes to perform as much work as possible on data in cache before writing it back to memory.

1.2 Algorithm Research

We are focusing our algorithm research on better approximations to the governing equations and mesh management. Our ALEGRA [1] code uses finite-element methods, an arbitrary-connectivity mesh and an Arbitrary-Lagrangian-Eulerian (ALE) formulation where the mesh can move to improve the accuracy. For example, part of the mesh may move with the body

[1] This work was performed at Sandia National Laboratories supported by the U.S. DOE under contract DE-AC04-94AL85000.

(Lagrangian mesh), another part may be space-fixed (Eulerian) and another part may move arbitrarily.

We are also developing h-adaptive meshing techniques for ALEGRA. H-adaptive meshes can dynamically refine or coarsen a mesh depending on some criteria. For example, a single hexahedral element may divide into eight hexahedra. We will use h-adaptive methods to resolve discontinuities such as shocks, material interfaces and shear flow. One of the biggest challenges is developing h-adaptive methods that run efficiently on massively parallel computers. Work must dynamically migrate between processors to load-balance the work over the computer.

2. Introduction

We want to design and test complex three-dimensional systems on computers. This requires predictive, three-dimensional codes. This paper will discuss some of our research in computational sciences and algorithms to achieve that goal.

Three-dimensional calculations require large meshes to resolve small scale phenomena. Large meshes require very high compute speeds. Only Massively Parallel (MP) computers offer the memory and processor speed needed. Unfortunately, MP computers come with new memory hierarchies that require rethinking the way we develop code.

The memory hierarchy change in commodity CPU's will influence your codes even if you do not run on MP computers. There is a corresponding change in computer hardware. Workstations are taking over the market. They use hierarchical memories rather than the memory architectures used in CRAY vector computers. Our solution algorithms must change to achieve high performance on workstations and MP computers.

We are also developing adaptive solution algorithms. They will move the mesh and change the mesh to reduce the calculation's global error.

Code development of this scale requires team efforts to both develop and validate the codes. The authors acknowledge the extensive contributions of both the CTH and ALEGRA teams.

3. Massively Parallel Computer Research

3.1 Why Do We Need Massively Parallel Computers for Shock Physics?

Shock physics codes need the enormous memories and compute speed of massively parallel computers to support the very large meshes required to resolve the small structures that arise in shock problems. The structures may be a shock wave, a vortex sheet (sliding), a material interface, a crack, the geometry of the object or a contact discontinuity. Our codes cannot resolve these structures at a level finer than a cell. We model many of these phenomena with first-order algorithms. As you refine the mesh, the error decreases linearly. For example, a code that uses artificial viscosity will *always* spread a shock over approximately four cells. Many other parts of the solution scheme are second-order accurate so their error decreases as the square of the mesh size. The first-order algorithms' errors dominate the total error and determine the mesh resolution needed.

3.1.1 Processor Speed Versus Memory

The processing speed needed for shock physics codes should increase faster than the memory size increases because the Floating Point Operations (FLOPs) scale up faster than the memory requirements. This implies you should not simply add more memory to a busy system and try to run larger problems.

The memory needed for a three-dimensional mesh increases as the cube of the mesh refinement.

$$Memory \approx (mesh\_refinement)^3$$

For example, a region of space covered with one centimeter cells will require eight times as much memory as the same space covered with two centimeter cells. However, the number of floating point operations (FLOPs) will increase as the *fourth* power of the mesh refinement.

$$FLOPs \approx (mesh\_refinement)^4$$

The first three-fold increase in flops comes from the larger mesh. The fourth increase comes because the time step used with explicit time integration schemes decreases as the mesh refines. We must add processing speed *faster* than we add memory or the run times will quickly become be too long to be useful.

$$Memory \approx (mesh\_refinement)^3$$
$$FLOPs \approx (mesh\_refinement)^4$$
$$FLOPs \approx (Memory)^{4/3}$$

Equation 1. Speed and Memory Scaling Relations

3.1.2 Memory and Performance of MP Computers

Massively parallel computers provide the needed combination of memory and speed. A typical workstation may have 0.256 Gbytes of memory and a peak speed of 0.23 GFLOPs peak processing power. Sandia's Paragon massively parallel computer has 37 Gbytes of memory and 140 GFLOPs peak processing power from 1840 i860 CPUs. The internode bandwidth is 200 Mbytes per second. (The internode bandwidth is the speed nodes can exchange messages.) DOE's ASCI TFLOP massively parallel computer has 594 Gbytes of memory and 1800 GFLOPs peak processing power from 9072 Pentium Pro CPUs. The internode bandwidth is 800 Mbytes per second. Shock physics codes need massively parallel computers to achieve TFLOPs on Tbytes.

Computer	GFLOP	Gbytes	bandwidth between nodes MB/s	# of CPUs
workstation	0.23	0.256	NA	1
Sandia's Intel Paragon	147	37	200	1840
DOE's TeraFLOP	1,800	594	800	9072

Table 1. Computer Statistics

3.2 Distributed mesh

We must design our shock physics codes to run effectively on massively parallel computers because MP computers have a different memory structure than traditional computers. All CPUs have access to all the memory on traditional shared memory processor (SMP) computers, such as CRAY vector computers and various vendors' workstations. The memory on a SMP has one global address space.

MP computers have a memory hierarchy with different access times to the different memories. A code must manage the access to the different memories to be efficient. Memory is associated with each compute node. (There may be multiple CPUs on a node, as on the ASCI TFLOP.) There will be fast access to the memory on the compute node. Much of the memory will be on the other compute nodes. The memory on another compute node may not be part of a specific node's address space. The access time to off-node memory will be much longer because the inter-node communication system must retrieve the memory. Therefore, too much inter-node communication can degrade the code's performance. We must carefully design our codes to run efficiently on these hierarchical memory systems.

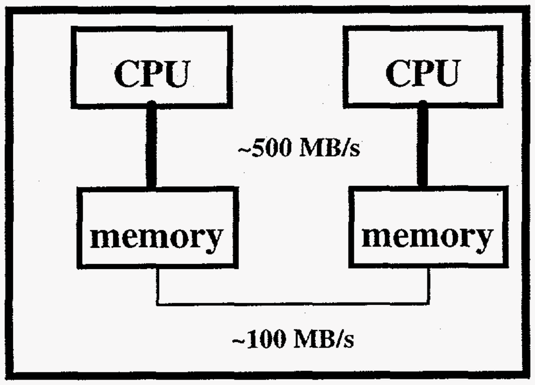

Figure 1. Hierarchical Memory and Communication Systems

Distributed meshes are an effective method for using MP computers. A distributed mesh breaks the global mesh into several nearly equal size submeshes. Each submesh is on a different compute node. The code solves the physics on its submesh.

3.2.1 Internal Boundaries

We introduce new internal boundaries when we break the global mesh apart. Most codes use some form of 'ghost' cells to store boundary condition data. We must add 'ghost' cells for the new internal boundaries. The ghost cells hold data that is on the interior of another submesh. For example, in Figure 2, the top, right-hand side of the ball in the ghost cell of sub-mesh #1 is actually a replication of the top, right-hand side of the ball in sub-mesh #2. The code updates the information in the ghost cells before the solution scheme uses the data. The code gathers the data into messages and passes it to a neighbor. We refer to this as *explicit message passing*.

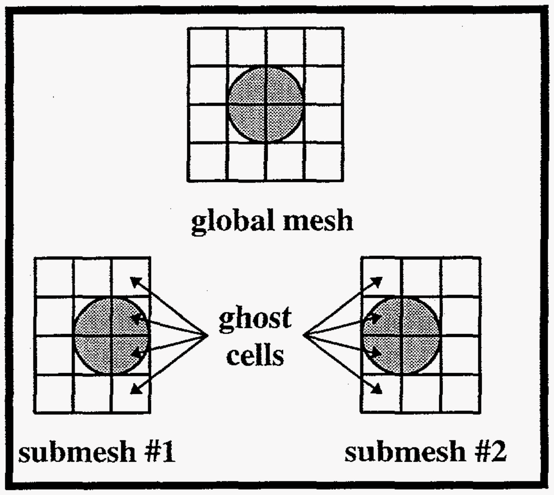

Figure 2. Global and Two Sub-Meshes

3.2.2 Mesh Decomposition

We need a tool to decompose a global mesh into the appropriate number of submeshes. The submeshes should be approximately the same size. This helps keep the same amount of work on each node and helps balance the work load. We refer to this as *static load balancing*.

The code should minimize communication between the submeshes to improve the efficiency. This is done by minimizing the 'surface area' of the new created internal boundaries. This is particularly important for arbitrary connectivity meshes used in finite-element codes because the connectivity is so complex. There are several mesh decomposition tools available. We use the Sandia developed CHACO [2] family of routines.

3.3 New Algorithms for New CPUs

3.3.1 Hierarchical Memories

Our shock physics codes need new core architectures because new memory architectures also appear on the compute node. The compute nodes used in workstations and MP computers have hierarchical (cache) memories. For example, a node may have two levels of memory. The first level is a 256 KByte cache memory with a GByte/sec bandwidth to the CPU's registers. The second level is a 128 MByte bank of memory with a 500 MByte/sec bandwidth to the cache.

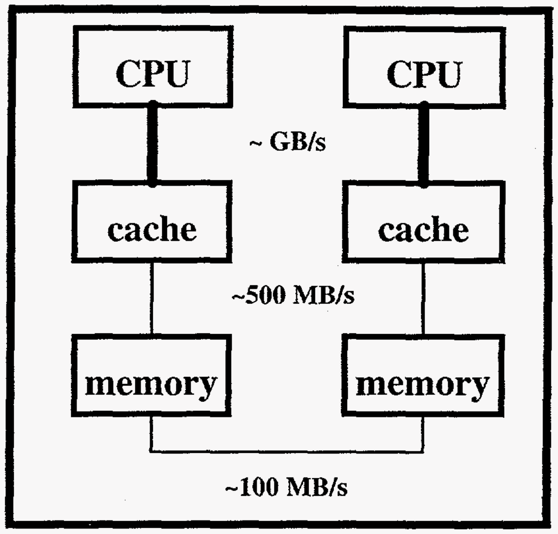

Figure 3. Cache and Main Memories

3.3.2 Cache Utilization

A code needs to perform most of its memory I/O from cache and not from the main memory to achieve high performance. The bandwidth between the CPU and cache can keep the CPU busy. But the bandwidth between the CPU and main memory cannot keep the CPU busy. If the data is not in the cache, then the CPU must wait while it reads the data from main memory into cache. Waiting for data degrades the performance.

3.3.3 New Code Architectures

We must restructure our algorithms for hierarchical memory computers. We originally developed the algorithms for vector memories on CRAY computers. Vector memories have very fast I/O to arrays. Existing algorithms may not run efficiently on hierarchical memories because they rely on the high bandwidth to memory. For example, our codes split the physics into several modest sized parts. The code would pass the database through each part of the physics. The modified data would overwrite the old data. There was a lot of I/O between memory.

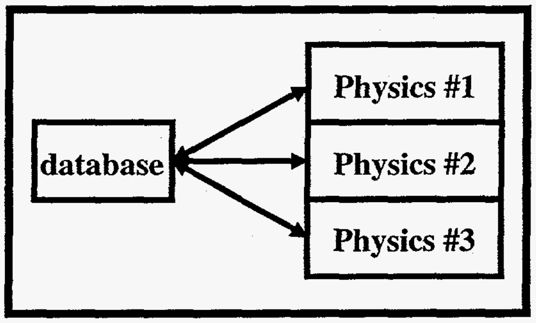

Figure 4. Old Code Architecture

We are going to a different code architecture for hierarchical memory computers. We break the data base into parts and keep the physics chunks as large a possible. We want to read some of the database into cache and perform as much of the physics as possible before writing it back out to main memory. Therefore, most of the CPU's I/O is to the cache rather than to main memory.

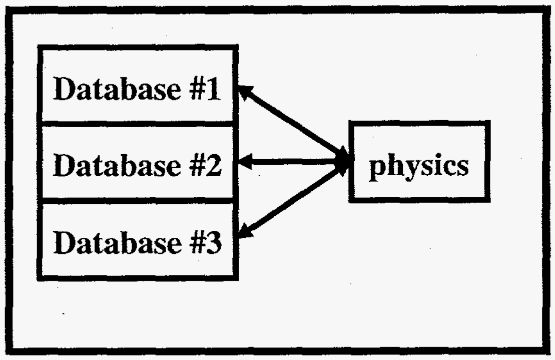

Figure 5. New Code Architecture

4. Algorithm Research

4.1.1 Arbitrary-Lagrangian-Eulerian Adaptive Algorithms

Our ALEGRA code uses an arbitrary-Lagrangian-Eulerian (ALE) mesh. A part of the mesh can move with the material (Lagrangian), be fixed in space (Eulerian) or anything in between (arbitrary). This allows us to run part of the mesh Eulerian and another part Lagrangian. We can also change the mesh from Eulerian to Lagrangian depending on the phenomena. The ALE capabilities allow the code to adapt the solution scheme to improve its accuracy.

4.1.2 Finite-Element Methods

ALEGRA use finite element rather than finite-difference or finite-volume algorithms. This allows us to use the large body of finite element technology. Finite element numerical techniques are also widely believed to give the best resolution of transient phenomena.

ALEGRA also uses arbitrary connectivity finite-element meshes. This allows us to use a body-fitted, finite-element mesh where the mesh boundaries coincide with the material interfaces. We can run the calculation Lagrangian or single-material ALE. A single-material ALE calculation has no multi-material element and requires the material interfaces remain Lagrangian but the material's interior can be non-Lagrangian. We can also use a fully multi-material mesh as in an Eulerian code. This gives us tremendous flexibility in modeling the behavior of complex three-dimensional structures.

4.1.3 H-Adaptive Methods

We are developing h-adaptive meshing schemes for ALEGRA. An h-adaptive mesh can subdivide an element into several elements to better resolve phenomena. We will use an error estimator to identify the elements with the largest error and refine those elements. For example, we may want to refine the mesh in the neighborhood of a shock to get 1 mm zoning and coarsen the mesh to a 1 cm mesh for the smooth flow behind the shock.

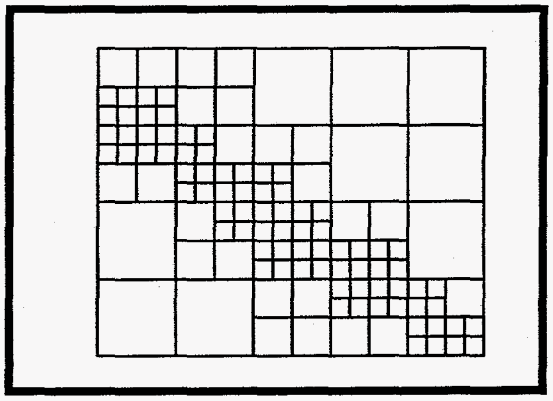

Figure 6. H-Adaptive Mesh

One of the biggest challenges is implementing h-adaptive methods on massively parallel computers. Static mesh decomposition techniques will not work as the mesh refines. A single element may transform into hundreds of elements. We will have to migrate work among compute nodes to keep the work load balanced and avoid exhausting the memory on the node.

5. Summary

We want a predictive, 3D modeling capability. The h-method will provide an adaptive algorithm that will use the computer resources to minimize the global error in the calculation. The MP computers will provide enough compute power and memory to effectively use h-adaptive methods on complex three-dimensional problems. The biggest challenges to achieving our goal are understanding the physics and understanding how to effectively use hierarchical memory computers.

6. References

[1] Peery, J. S., Budge, K.G., Wong, M. K., Trucano, T. G. "RHALE: A 3D MMALE Code for Unstructured Grids," Proceedings of the 1993 Winter ASME Meeting, New Orleans, LA

[2] Hendrickson, B. and Leland, R., "The CHACO Users Guide," Sandia National Laboratories Report SAND93-2339, 1993, Sandia National Laboratories, Albuquerque, NM