Transient Solid Dynamics Simulations on the Sandia/Intel Teraflop Computer

Stephen Attaway[*], Ted Barragy[†], Kevin Brown[*], David Gardner[*], Bruce Hendrickson[*], Steve Plimpton[*] and Courtenay Vaughan[*]

Abstract

Transient solid dynamics simulations are among the most widely used engineering calculations. Industrial applications include vehicle crashworthiness studies, metal forging, and powder compaction prior to sintering. These calculations are also critical to defense applications including safety studies and weapons simulations. The practical importance of these calculations and their computational intensiveness make them natural candidates for parallelization. This has proved to be difficult, and existing implementations fail to scale to more than a few dozen processors. In this paper we describe our parallelization of PRONTO, Sandia's transient solid dynamics code, via a novel algorithmic approach that utilizes multiple decompositions for different key segments of the computations, including the material contact calculation. This latter calculation is notoriously difficult to perform well in parallel, because it involves dynamically changing geometry, global searches for elements in contact, and unstructured communications among the compute nodes. Our approach scales to at least 3600 compute nodes of the Sandia/Intel Teraflop computer (the largest set of nodes to which we have had access to date) on problems involving millions of finite elements. On this machine we can simulate models using more than ten-million elements in a few tenths of a second per timestep, and solve problems more than 3000 times faster than a single processor Cray Jedi.

1 Introduction

Transient dynamics simulations are among the most widely used engineering calculations. The industrial application which consumes more time on Cray vector supercomputers than any other is crash simulations, a prototypical transient dynamics calculation[7]. Other industrial applications include simulations of metal forging,

[*]Sandia National Labs, Albuquerque, NM.
[†]Intel Corporation, Beaverton, OR.

powder compaction prior to sintering and other processes involving high stresses and strains. These calculations are also critical to defense applications including safety studies and weapons simulations. A number of commercial and government solid dynamics codes have been developed including DYNA, PamCrash and ABACUS. Sandia also has a long history of research and code development in this area, headlined by the PRONTO code suite. PRONTO is similar in scope to the commercial codes, but also includes smoothed particle hydrodynamics (SPH), which allows for simulations with very high strains (e.g., explosions) or coupled fluid/structure interaction problems. A discussion of some PRONTO simulations can be found Section 7. The practical importance of transient dynamics simulations, combined with their computational intensiveness would seem to make them natural candidates for parallelization. Unfortunately, this has proved to be quite difficult. For reasons discussed below, existing parallel implementations fail to scale to more than a few dozen processors. These disappointing results have convinced leaders in the solid dynamics community that parallel computing can not yet make a significant impact in this field[2].

In Section 2 we describe the functionality and structure of PRONTO. In Section 3 we explain why transient dynamics simulations have been difficult to parallelize. Our parallelization strategy is sketched in Section 4 and some further performance enhancements are described in Section 5. The performance of the code on some scalable problems is discussed in Section 6. A discussion of applications enabled by parallel PRONTO follows in Section 7. Conclusions are drawn in Section 8.

2 What is PRONTO?

PRONTO is a three-dimensional, transient solid dynamics code which is used for analyzing large deformations of nonlinear materials subjected to high rates of strain[3]. Developed over the past 10 years, PRONTO is a production-level code used by over 50 organizations inside and outside Sandia. Input to the code includes an unstructured grid consisting of an arbitrary mixture of hexahedral elements, shell elements, rigid bodies and smoothed particles. PRONTO implements a Lagrangian finite-element method with explicit time integration and adaptive timestep control to integrate the equations of motion. The finite-element formulation uses eight-node, uniform strain hexahedral elements and four-node quadrilateral uniform strain shell elements. Either the Flanagan-Belytschko hourglass control scheme or an assumed-strain hourglass control scheme can be used to control element distortions. PRONTO contains a variety of complex, nonlinear material models, including elastic-plastic materials with various types of strain hardening. A critical feature of the code is a robust algorithm for detecting when one material surface contacts another, for example in an automobile collision when the bumper buckles into the radiator. Correctly identifying surfaces in contact requires sophisticated algorithms for searching the global set of finite-elements. In a complex simulation, the cost of contact detection alone can be more than 50% of the run time on a sequential machine. A PRONTO timestep has the following structure.

1. Perform finite element analysis to compute forces on elements.
2. Compute forces between smoothed particles.
3. Predict new locations of particles and grid elements.
4. Search for contacts between mesh elements, or between elements and particles.
5. Correct the locations by pushing back objects in contact.

Stages (1), (2) and (4) dominate the sequential run time. The contact search in stage (4) typically consumes 30-60% of the time, so a great deal of effort has been expended over the years to make this computation fast[4]. The result of this effort was the replacement in PRONTO of floating point operations with a faster approach involving sorting and searching in integer lists.

3 Why is Parallelization Difficult?

Parallelizing transient dynamics codes is challenging for several reasons. For PRONTO there is the obvious complexity of starting with a fully featured production code. All its functionality must be parallelized in a scalable way. Even more daunting is the inherent difficulty of parallelizing several key kernel operations which operate on different data sets. The first task is to parallelize the finite element (FE) portion of the code. This is conceptually straightforward: partition the elements among processors in a way that balances computation while minimizing communication[5]. But parallelizing contact detection (which is performed on only the surface mesh - not the volumetric FE mesh) is much harder. To our knowledge, no previous attempts at parallelizing contact detection have scaled to more than a few dozen processors[8, 9]. Since, in principle, on a given timestep any surface can contact any other, contact detection requires some kind of **global** search. As the geometry of the simulation evolves, this requires **dynamic** load balancing and **irregular** communication. Problems which exhibit any global, dynamic or irregular behavior are challenging to parallelize; contact detection exhibits all three. Parallelizing smoothed particle hydrodynamics (SPH) is also a challenging problem. Particles with time-dependent radii interact if they are geometrically near each other, and their density can vary greatly as the calculation proceeds, posing a load-balancing problem. Computing the physics of the SPH interactions also requires several stages of inter-processor communication within a timestep. The key difficulty in making a code like PRONTO perform well on a large parallel machine is that all of these computational kernels must be parallelized efficiently within the same timestep. And each of the kernels operates on a different data set (volumetric mesh, surface mesh, particles) whose spatial density is dynamically changing.

4 Our Parallel Implementation

We only sketch our parallelization strategy here. More details can be found in some of the references[1, 6, 10]. Most previous attempts to parallelize transient dynamics codes have relied upon a single decomposition of the mesh for both finite elements and contact detection. But these operations demand very different decomposition properties. The finite element analysis performs optimally only if each processor has the same number of elements and interprocessor boundaries are minimized. This decomposition can be generated once and used throughout the calculation. In contrast, contact detection and SPH depend upon geometric proximity, so a geometric decomposition is most appropriate. As the elements and particles evolve, the decompositions should change dynamically. The key idea behind our parallelization strategy is that we construct and maintain different decompositions for the different portions of the calculation. We choose appropriate decompositions to optimize performance of each phase: a graph-based static method for the finite element analysis generated by Chaco[5], and dynamic, geometric decompositions for contact detection and SPH. For the latter we use recursive coordinate bisection (RCB) which has a number of attractive properties for this application. The advantage of this approach is that we can achieve high performance in all phases of the calculation. The downside is that we need to communicate considerable information between the different decompositions which is expensive in both time and memory. But by carefully implementing the communication routines we can limit the run time cost, and solid dynamics calculations are not generally memory-bound. As our results will indicate, the advantages of multiple decompositions greatly outweigh the costs.

A timestep of parallel PRONTO has the following structure.

1. Perform finite element analysis to compute forces on elements.
2. Update the RCB decomposition of smoothed particles.
3. Compute forces between smoothed particles.
4. Predict new locations of particles and grid points.
5. Ship data to previous decomposition of the contact problem.
6. Update the RCB decomposition of the contact problem.
7. Search for contacts between mesh elements, or between elements and particles.
8. Communicate contact results back to finite element and SPH decompositions.
9. Correct the locations by pushing back objects in contact.

Our parallelization of PRONTO required about 15,000 lines of new code. In addition, much of the original PRONTO code was restructured for the parallel version to improve data locality on cache-based architectures.

5 Maximizing Performance

The goal of both serial and parallel PRONTO is to enable very large problems to run as quickly as possible. The dominant steps in the above outline are stages (1) and (7). (In this and the next section we focus on mesh-only problems though SPH computations can also be time consuming.) The fastest way to perform the global searches inherent in stage (7) is to do virtually no flops at all, but rather to use integer-based sort and search operations. Our calculations were performed on the 3600-node Sandia Teraflop computer. Each node of this machine has 128 Mbytes of memory and two 200 Mhz Pentium-Pro processors, each of which runs at 200 Mflops peak. We specially coded the kernel operations of the finite element computation to use the second processor for computation wherever possible. In practice the speed-up thus obtained is limited by memory bandwidth since the two processors share the same memory bus. We also reorganized some data structures to improve cache locality. These efforts improved the performance of the finite element computation from 40 Mflops per node to over 120 Mflops per node. For the contact computation, our algorithm already insures load-balance of the basic sort and search operations. We further optimized by altering the basic algorithm to avoid a global search on most of the timesteps. To accomplish this we occasionally perform a full search which stores all pairs of nearby surfaces. On subsequent timesteps we need only scan this list instead of searching the processor's entire domain. When the geometry has evolved enough that the lists could miss possible contacts, a new global search is triggered. This method requires extra memory for storing the lists, but it halved the overall contact computation time.

6 Performance

Depending on the physical problem being modeled, parallel PRONTO can run as a pure finite element computation without contacts, as finite elements with contacts, as pure SPH particles (no finite elements), or as coupled finite elements and SPH particles with contacts. Here we focus on the performance of the first two cases. In all of the performance numbers we present, we timed the outermost timestepping loop of PRONTO to determine CPU time per timestep. Problem setup time (which is constant independent of the number of timesteps simulated), was not included since it is insignificant in production-scale runs. We counted floating-point operations using hardware counters on the Pentium Pro chips. This hardware counts floating point divides, adds and multiplies as one flop each.

To test the performance of a pure finite element run of parallel PRONTO, we modeled a steel bar with hexahedral elements vibrating due to an oscillatory stress while being pinned at the ends. This simple problem was selected since it is easy to scale to different sizes. Strains induced between adjacent elements and the material's equation of state are modeled in the FE computation, but the bar does not bend enough to create contacts. We observed nearly 100% parallel efficiency in running this problem if we scaled the problem size (number of mesh elements) linearly with the number of processors. As mentioned above, the FE computational kernels run at about 120 Mflops/node. Interprocessor communication is only a few percent of the total run time. Other lower flop-rate overhead within the timestep (boundary conditions and time integration) takes about one half the CPU time regardless of the number of processors. Scaling the problem to the full Teraflop machine, we ran a 14.04 million element version of the beam problem on 3600 nodes at 224.9 Gflops (62.5 Mflops/node), requiring 0.166 CPU secs/timestep.

Our second benchmark is more interesting as it is prototypical of the problems for which PRONTO was designed. We simulated the crush of an idealized steel shipping container by an inclined wall, as shown in Fig. 1. As with the first benchmark, this computation is easily scaled due to its regular geometry. However, this calculation is considerably more complex. The crumpling of the folded surfaces is a stringent test of the contact algorithm's accuracy and performance. A symmetry plane was used so that only half the container was actually simulated. An elastic-plastic material model was used for the steel in both the can and wall. Within the contact algorithm, global searches were conducted about every five timesteps.

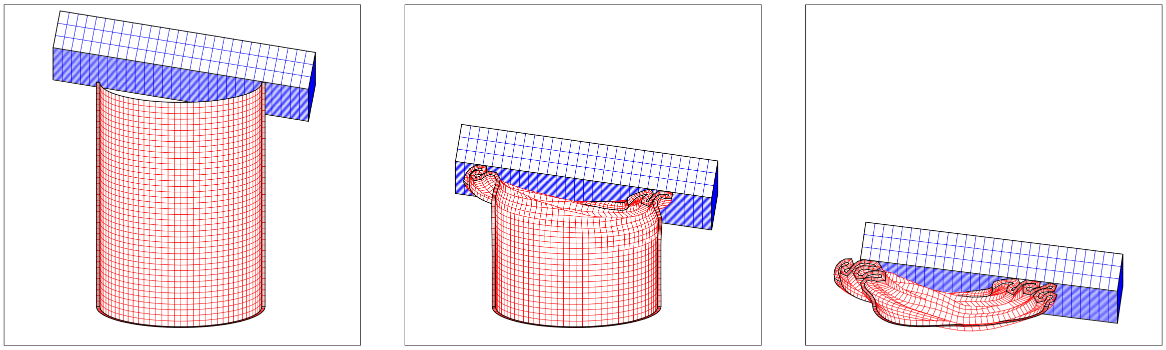

Figure 1: Crushing of idealized shipping container.

Parallel timings are shown in Fig. 2 for a set of small scaled simulations with 1875 elements/node. Every time the number of processors P is doubled, the mesh is refined in one of the three dimensions so that the number of mesh elements N also doubles. Thus the leftmost data points are for a 3750 element simulation running on 2 processors. The rightmost data points are for a 6.57 million element simulation on 3504 processors.

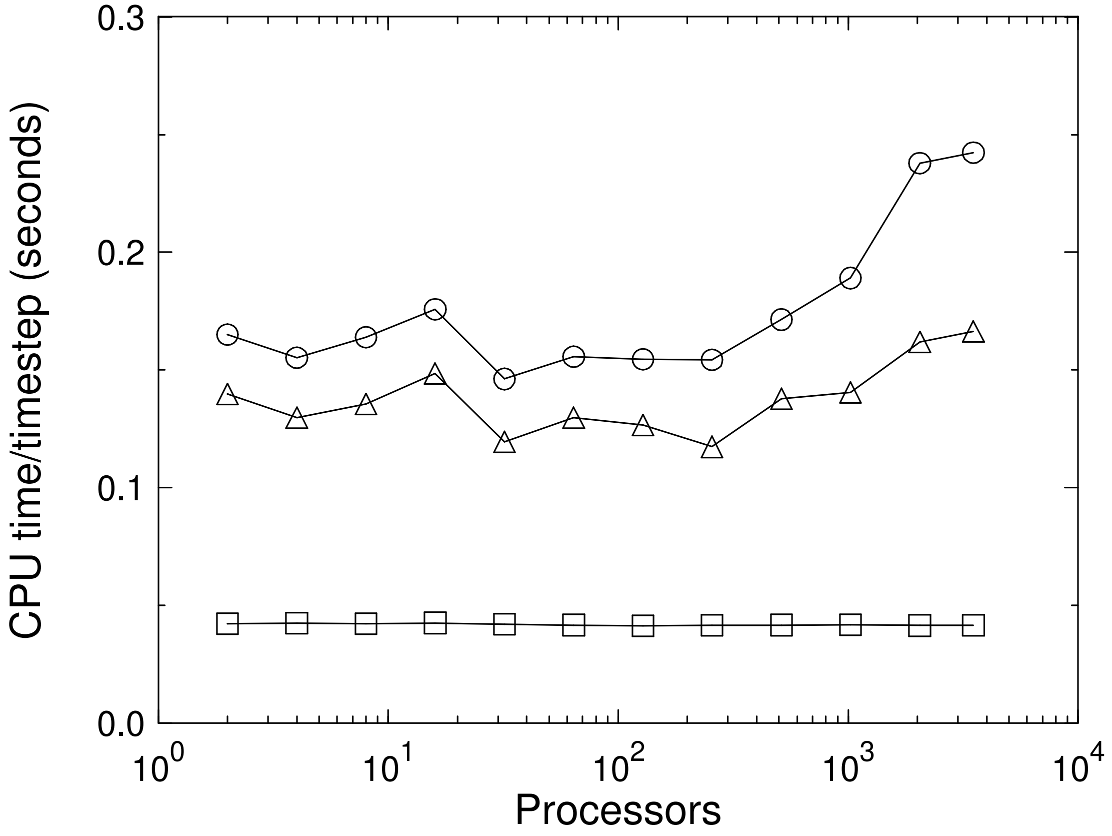

Figure 2: Scaled speedup for small container-crush problem.

The topmost curve is the total CPU time per timestep averaged over a 100 microsecond (problem time) run. On the small problems this is a few hundred timesteps; on the large problems it is several thousand, since the timestep size must shrink as the mesh is refined. The lowest curve is the portion of time spent in the FE computation. Contact detection is the time between the lowest and middle curves. Overhead is the time between the top two curves. We again see excellent scalability to very large N and P. Perfect scalability would be a horizontal line on this plot. The FE computation scales essentially perfectly. The contact detection time varies from one problem size to the next due to variations in surface-to-volume ratios of mesh elements as refinement is done in different dimensions, but is also roughly horizontal. The overhead time is also a constant portion of the total run time (i.e. scalable) as P increases until the P=2048 and P=3504 data points. The reason for the non-scalability here is that the overhead timing includes the cost to push-back contacts that are detected. This normally small computation becomes somewhat unbalanced in this problem on very large numbers of processors. The overall flop performance of parallel PRONTO on this problem is 76.05 Gflops on 3504 nodes of the Teraflop machine. Essentially all the flops are computed within the FE computation (lowest curve) which again runs at about 120 Mflops/node. The majority of the remaining CPU time is spent in the integer-based contact searches and sorts (no flops).

A set of larger simulations of the container crush was also performed where each run used a mesh with about 3800 elements/node. These timings are shown in the Fig. 3. As before, the upper curve is total CPU time per timestep. PRONTO again evidences excellent scalability, since all of the timing curves are roughly horizontal. The largest problem (rightmost data points) is a simulation of 13.8 million mesh elements on 3600 nodes of the Teraflop machine. It runs at a sustained rate of 120.4 Gflops or 33.4 Mflops/node.

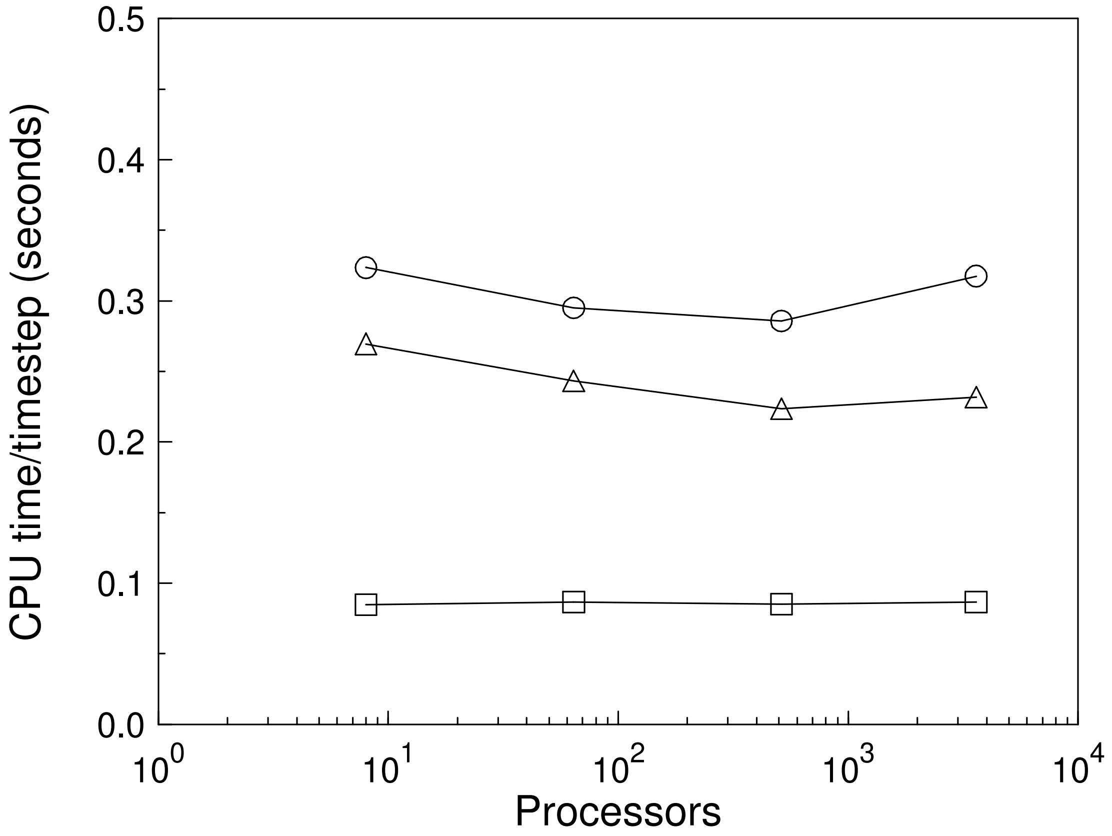

Figure 3: Scaled speedup for large container-crush problem.

7 Applications

Parallel PRONTO has been used to perform a range of calculations which were previously impractical or impossible. Here we briefly sketch three representative applications.

7.1 Application I: Airplane Crash Fuel Dispersal

In an airplane crash, fires fed by ruptured fuel tanks are a great threat to survivors and to hazardous cargo. The danger posed by such a fire depends critically on the dispersal pattern of the fuel. Parallel PRONTO is ideally suited for simulating these kinds of incidents since it can combine structural analysis for the plane with smoothed particle hydrodynamics for the fuel. Fig. 4 shows a simulation of an airplane wing striking a vertical pole. In the image on the left, the purple dots are SPH particles representing the resulting fuel cloud. The image on the right shows the damage to the wing itself. Note that the collision tears the wing. This particular example illustrates how pronto allows the surface to be adaptively redefined as portions of model experience failure. If the strain in a given element becomes too large, failure is simulated by deleting the element. Allowing the elements to be adaptively deleted requires the parallel contact algorithm to be capable of tracking and updating the changing contact surface as the problem progresses. This calculation was run on 128 nodes of the Teraflop computer using about 110,000 hexahedral and shell elements to model the structures and about 130,000 SPH elements to model the fuel. More detailed versions of this problem are being developed which will include the entire airplane and a soil model for impact. the current limitation lies in the tools to build the computational mesh. These calculations are being performed by John Pott at Sandia.

7.2 Application II: Shipping Container Integrity

A problem of great interest to the DOE is the integrity of shipping containers for transporting weapons and hazardous waste. Specifically, will the containers function properly in the event of a vehicular collision? An image of such a simulation of interest is depicted in Fig. 5, where the container is about to be crushed between two steel walls. This simulation involves more than 1.3 million elements, and includes both hexahedral and shell elements. The large number of elements is necessary to resolve critical small-scale structural details of the container. Studies of this model with parallel PRONTO are ongoing. This work is being performed by Jeff Gruda at Sandia.

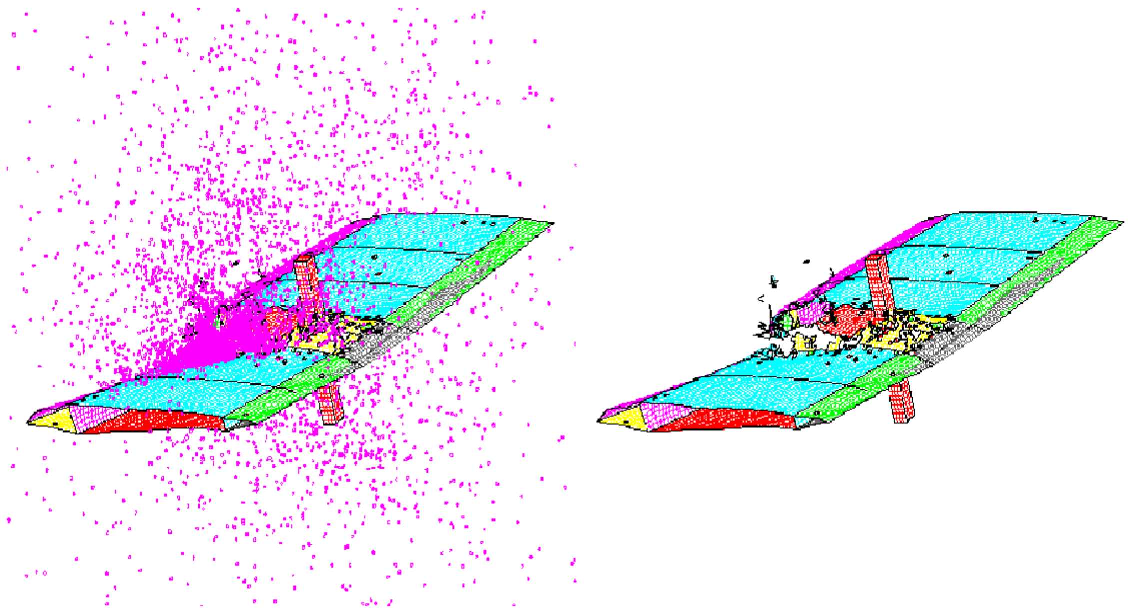

Figure 4: Simulation of wing hitting vertical pole.

7.3 Application III: Constitutive Models of Foams

Foams of various types are widely used to distribute impact forces or to absorb energy in collisions. The macroscopic properties of foams depend upon their fine-scale structure in a complex manner that is not well understood. Better constitutive models of foam properties can be obtained through simulations of small-scale behavior. Unfortunately, very large simulations are necessary to be able to compare computations to experiments. Until the parallelization of PRONTO, such simulations were impossible. This example illustrates how parallel PRONTO has enabled qualitatively new and different engineering studies.

Fig. 6 depicts a simulation of an open-cell foam, with cells about 1mm in diameter. A linear elastic material model was used, but the complex buckling and folding generates complex nonlinear behavior. The foam is being crushed from above by a fast moving plate.

As the picture reveals, there is some crush near the impacting plate, but much more on the opposing boundary. This is due to the reflection of stress waves off of the bottom plate. This behavior is consistent with experimental observations.

Each of the foam struts was modeled with multiple hexahedral elements, totaling more than 900,000. While one could use beam finite elements, the complex deformation patterns associated with large crush could cause the beam elements difficulty. In particular, the beam-on-beam contact would be very hard to detect. By using hexa-

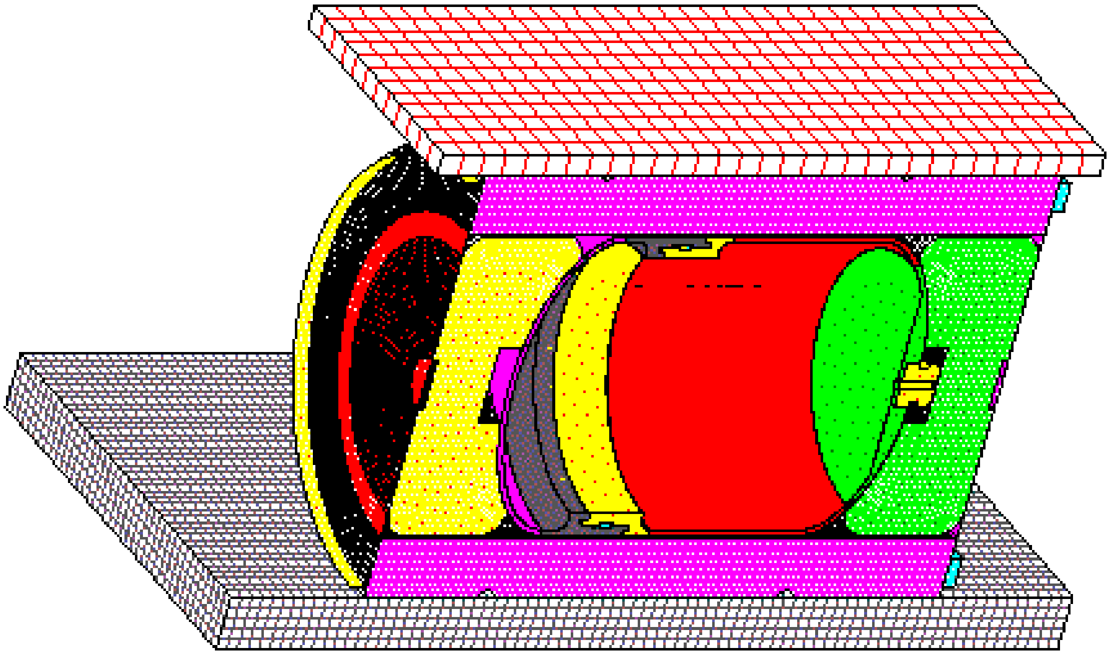

Figure 5: Simulation of shipping container crushed between steel walls.

hedral elements, we are able to model very complex contact conditions. The drawback to using hexahedral elements, aside from the number of elements required, is that a very small timestep is required to properly integrate the motion. The problem was run on 512 nodes of the Teraflop computer and required 8.8 hours of CPU time. Over 650,000 timesteps were used to integrate the motion in this problem. The complexity of the model and the physics can be appreciated in the close-up view shown in Fig. 7. The large number of finite elements comprising the struts are clearly visible, as is the complicated folding and contact patterns. The red regions are those with the highest stresses. This study is being conducted by Mike Neilsen and Stephen Attaway at Sandia.

8 Conclusions

We have successfully parallelized a large-scale production solid dynamics code with a novel algorithmic approach that utilizes multiple decompositions for different key segments of the computations. On our 3600-node Teraflop computer, parallel PRONTO runs complex finite element (FE) simulations with global contact searches at rates of up to 120 Gflops. The finite element kernel can run contact-free FE simulations at a

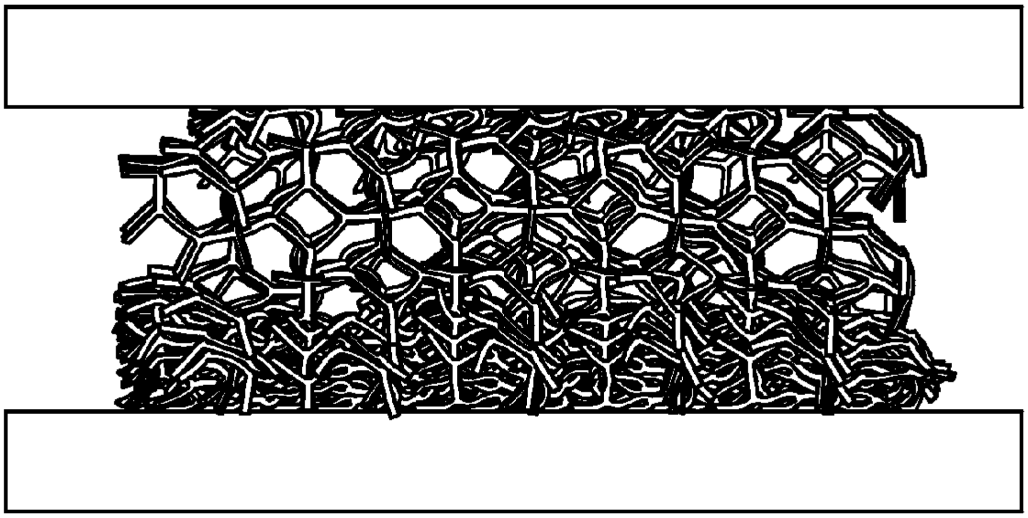

Figure 6: Simulation of partially crushed, open-cell foam.

rate of 225 Gflops. While these flop rates may not seem impressive when compared to other kinds of simulations or the peak rate of the Pentium Pro chips, some context may be useful. First, to be able to simulate a more than ten million element model in a few tenths of second per timestep is unprecedented for solid dynamics simulations, especially when full global contact searches are required. The key reason is our new algorithm for efficiently parallelizing the contact detection stage. To our knowledge scalability of this computation had never before been demonstrated on more than 64 processors. This has enabled parallel PRONTO to become the only solid dynamics code we are aware of that can run effectively on 1000s of processors. More importantly, our parallel performance compares very favorably to the original serial PRONTO code which is optimized for vector supercomputers. On the container crush problem, a Teraflop node (two Pentium Pro processors) is as fast as a single processor of the Cray Jedi. This means on 3600 nodes of the Teraflop machine we can now run simulations with tens of millions of elements over 3000 times faster than we could on the Jedi! This is enabling transient dynamics simulations of unprecedented scale and fidelity. Not only can previous applications be run with vastly improved resolution and speed, but qualitatively new and different analyses have been made possible.

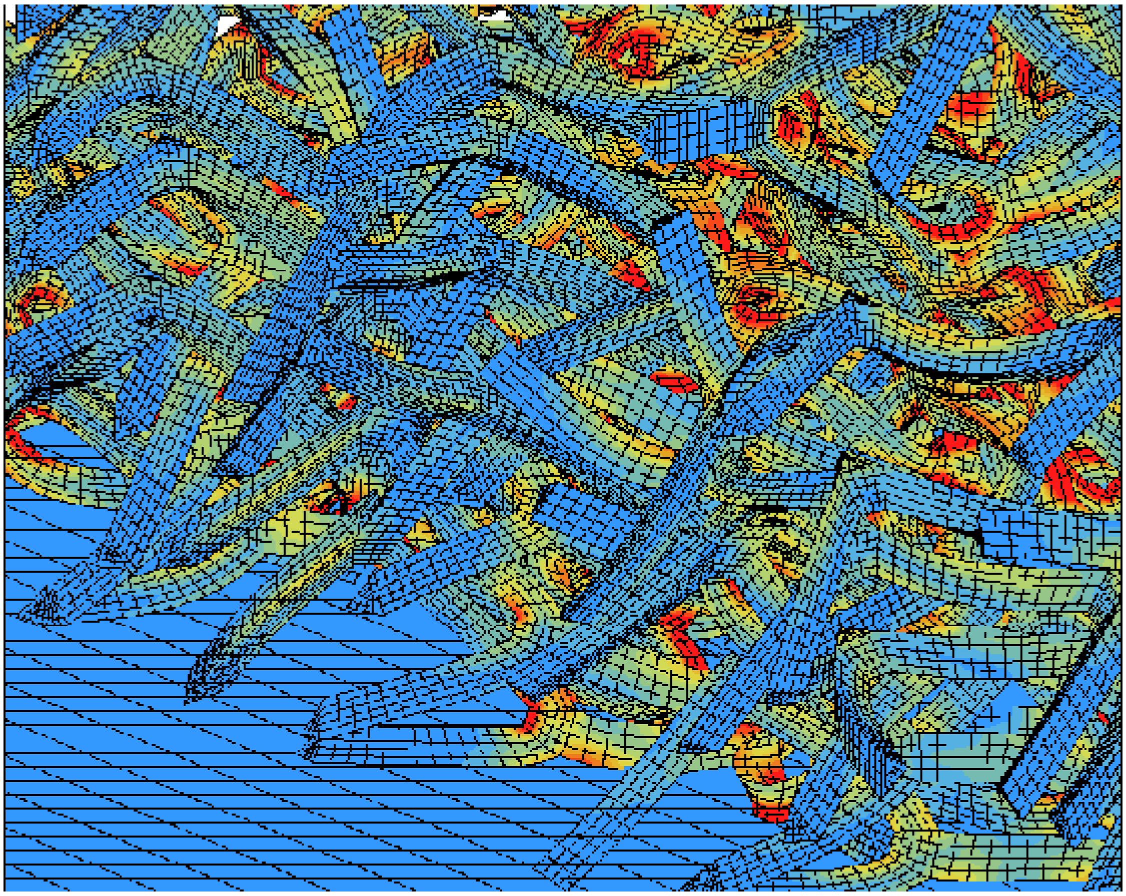

Figure 7: Close-up view of partially crushed, open-cell foam.

Acknowledgments

We are indebted to Gary Hennigan at Sandia for creating the NEMESIS toolkit which has been of great utility to us in pre-processing our multi-million element meshes for the Teraflop machine. Ben Cole of Intel has provided timely hardware and software support to our project. Tim Preston of Sandia helped us maintain a quality production-level code. Running our suites of test problems would not have been possible without the helpful scripts provided by Christi Forsythe and Mike Brayer of Sandia. This work was performed at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy (DOE) under contract DE-AC04-94AL85000. We gratefully Sandia/Intel Teraflop computer. We also received funding from the Joint DoD/DOE Munitions Technology Development Program for development of the contact algorithm.

References

[1] Stephen W. Attaway, Bruce A. Hendrickson, Steven J. Plimpton, David R. Gardner, Courtenay T. Vaughan, Martin W. Heinstein, James S. Peery. **Parallel Contact Detection Algorithm for Transient Solid Dynamics Simulations Using PRONTO3D**. Proc. International Mech. Eng. Congress & Exposition '96, November 1996.

[2] Ted Belytschko. Northwestern University, personal communication.

[3] L. M. Flanagan and D. P. Flanagan. **PRONTO3D: A Three-Dimensional Transient Solid Dynamics Program**. Tech. Rep. SAND87-1912, Sandia National Labs, Albuquerque, NM, March 1989.

[4] M. W. Heinstein, S. W. Attaway, J. W. Swegle and F. J. Mello. **A General-Purpose Contact Detection Algorithm for Nonlinear Structural Analysis Codes**. Tech. Rep. SAND92-2141, Sandia National Labs, Albuquerque, NM, May 1993.

[5] Bruce Hendrickson and Robert Leland. **The Chaco User's Guide: Version 2.0**. Tech. Rep. SAND94-2692, Sandia National Labs, Albuquerque, NM, June 1995.

[6] Bruce Hendrickson, Steve Plimpton, Steve Attaway, Courtenay Vaughan, David Gardner. **A New Parallel Algorithm for Contact Detection in Finite Element Methods**. Proc. High Performance Computing '96, April 1996.

[7] Mike Heroux. Cray Research, personal communication.

[8] C. G. Hoover, A. J. DeGroot, J. D. Maltby, R. D. Procassini. **Paradyn: DYNA3D for massively Parallel Computers**. Presentation at the Tri-Laboratory Engineering Conference on Computational Modeling, October 1995.

[9] J. G. Malone and N. L. Johnson. **A Parallel Finite Element Contact/Impact Algorithm for Nonlinear Explicit Transient Analysis: Part II - Parallel Implementation**. Intl. J. Num. Methods Eng. 37 (1994), pp. 591-603.

[10] Steve Plimpton, Steve Attaway, Bruce Hendrickson, Jeff Swegle, Courtenay Vaughan, David Gardner. **Transient Dynamics Simulations: Parallel Algorithms for Contact Detection and Smoothed Particle Hydrodynamics**. Proc. Supercomputing'96, November 1996.