A NEW PARALLEL ALGORITHM FOR CONTACT DETECTION IN FINITE ELEMENT METHODS*

Bruce Hendrickson† Steve Plimpton Steve Attaway Courtenay Vaughan
David Gardner

Abstract

In finite–element, transient dynamics simulations, physical objects are typically modeled as Lagrangian meshes because the meshes can move and deform with the objects as they undergo stress. In many simulations, such as computations of impacts or explosions, portions of the deforming mesh come in contact with each other as the simulation progresses. These contacts must be detected and the forces they impart to the mesh must be computed at each timestep to accurately capture the physics of interest. While the finite-element portion of these computations is readily parallelized, the contact detection problem is difficult to implement efficiently on parallel computers and has been a bottleneck to achieving high performance on large parallel machines. In this paper we describe a new parallel algorithm for detecting contacts. Our approach differs from previous work in that we use two different parallel decompositions, a static one for the finite element analysis and dynamic one for contact detection. We present results for this algorithm in a parallel version of the transient dynamics code PRONTO-3D running on a large Intel Paragon.

1 Introduction

Transient dynamics models are often formulated as finite element simulations on Lagrangian meshes. Unlike Eulerian meshes which remain geometrically fixed as the simulation proceeds, Lagrangian meshes can be easily fitted to complex objects and can deform as objects change shape during a simulation. Prototypical phenomena that are modeled in this way include car crashes, and metal forming and cutting for manufacturing processes. Commonly–used commercial codes that simulate these effects include LS–DYNA3D, ABACUS, and Pam-Crash. PRONTO-3D is a DOE code of similar scope that was developed at Sandia [11].

A complicated process such as a collision or explosion involving numerous complex objects requires a large number of mesh elements to model accurately. The underlying physics of the stress-strain relations for a variety of interacting materials must also be included in the model. Running such a simulation for thousands or millions of timesteps can be very computationally intensive, and so is a natural candidate for the power of parallel computers.

The finite-element (FE) portion of the computation within a single timestep can be parallelized straightforwardly. In an explicit timestepping scheme, each mesh element interacts only with the neighboring elements it is connected to in the FE mesh topology. If each processor is assigned a small cluster of elements then the only interprocessor communication will be the exchange of information on the cluster boundary with a handful of neighboring processors. A variety of algorithms and tools have been developed that optimize this assignment task. For PRONTO-3D we use a software package called Chaco [4] which partitions the FE mesh so that each processor has an equal number of elements and interprocessor communication is minimized. In practice, the resulting FE computations are highly load–balanced and scale efficiently (over 90%) when large meshes are mapped to thousands of processors. The chief reason for the scalability is that the communication required by the FE computation is *local* in nature.

It is important to note that because the mesh connectivity does not change during the simulation (with a few minor exceptions), a *static* decomposition of the elements is sufficient to insure good performance. To achieve the best possible decomposition, we partition the FE mesh as a pre–processing step before the transient dynamics simulation is run. Similar FE parallelization strategies have been used in other transient dynamics codes [6, 8, 9, 10].

In most simulations there is a second major computation which must be performed each timestep. This is the detection of *contacts* between unconnected ele-

---

*To appear in Proc. High Performance Computing '96

†Sandia National Labs, Albuquerque, NM 87185-1110. Email: [bah,sjplimp,swattaw,ctvaugh,drgardn]@cs.sandia.gov.

and Cray T3D. An important aspect of our approach is that we use a different decomposition for contact detection than we use for the finite element calculation. This allows us to optimize each portion of the code independently. For contact detection we use a dynamic technique known as recursive coordinate bisection (RCB) to generate the decomposition anew at each timestep. We find several advantages to this approach. First, and foremost, since each processor ends up with the same number of contact nodes and surfaces, we can achieve nearly perfect load balance in the on–processor contact detection calculation. Second, the cost of performing an RCB decomposition is minimal if it begins with a nearly–balanced starting point. We use the result from the previous timestep, which will always be close to the correct decomposition for the current timestep. Third, the local and global communication patterns we use in our algorithm are straightforward to implement and do not require any complicated analysis of the simulation geometry. The price we pay for these advantages is that we must communicate information between the FE and contact decompositions at every timestep. Our results indicate that the advantage of achieving load balance greatly outweighs the cost of maintaining two decompositions.

We have recently become aware of independent work [6] which has some similarity to our approach. Like our technique, this approach uses a different decomposition for the contact detection than for the finite element analysis. In their method, they decompose the contact surfaces and nodes by overlaying a regular, coarse 3–D grid on the entire simulation domain. The coarse grid is then divided along one dimension into slices and each processor is responsible for contact detection within a slice. While this approach is likely to perform better than a static decomposition, the implementation described in [6] suffered from load imbalance and did not scale to large numbers of processors.

In the next section we provide some background material that will help explain our algorithm in §3. This is followed in §4 by some performance results from simulations using PRONTO-3D.

2 Background

Our contact algorithm involves a number of unstructured communication steps. In these operations, each processor has some information it wants to share with a handful of other processors. Although a given processor knows how much information it will send and to whom, it doesn't know how much it will receive and from whom. Before the communication can be performed efficiently, each processor needs to know about the messages it will receive. We accomplish this with the approach sketched in Fig. 2.

```
(1) Form vector of 0/1 denoting who I send to
(2) Fold vector over all $P$ processors
(3) $nrecvs$ = vector($q$)
(4) For each processor I have data for,
        send message containing size of the data
(5) Receive $nrecvs$ messages with sizes coming to me
(6) Allocate space & post asynchronous receives
(7) Synchronize
(8) Send all my data
(9) Wait until I receive my data
```

Figure 2: Parallel algorithm for unstructured communication for processor q.

In steps (1–3) each processor learns how many other processors want to send it data. In step (1) each of the P processors initializes a P–length vector with zeroes and stores a 1 in each location corresponding to a processor it needs to send data to. The fold operation [2] in step (2) communicates this vector in an optimal way; processor q ends up with the sum across all processors of only location q, which is the total number of messages it will receive. In step (4) each processor sends a short message to the processors it has data for, indicating how much data they should expect. These short messages are received in step (5). With this information, a processor can now allocate the appropriate amount of space for all the incoming data, and post receive calls which tell the operating system where to put the data once it arrives. After a synchronization in step (7), each processor can now send its data. The processor can proceed once it has received all its data.

The recursive coordinate bisectioning (RCB) algorithm we use was first proposed as a static technique for partitioning unstructured meshes [1]. Although for static partitioning it has been eclipsed by better approaches, RCB has a number of attractive properties as a dynamic partitioning scheme which have been exploited by Jones and Plassmann [7]. The subdomains produced by RCB are geometrically compact and well–shaped. The algorithm can also be parallelized in a fairly inexpensive manner. And it has the attractive property that small changes in the geometry induce only small changes in the partitions. Most partitioning algorithms do not exhibit this behavior.

The collection of points we want to divide equally among P processors is the combined set of N contact surfaces and nodes as shown in Fig. 3 for a 2–d example. For this operation we treat each surface as a

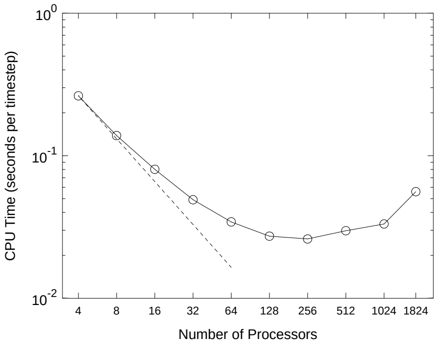

Figure 7: Average CPU time per timestep to crush a container with 7152 finite elements on the Intel Paragon. The dotted line denotes perfect speed–up.

Fig. 8 shows performance on a scalable version of the crush simulation where the container and surface are meshed more finely as more processors are used. On one processor a 1875–element model was run. Each time the processor count was doubled, the number of finite elements was also doubled by halving the mesh spacing in a particular dimension. Thus all the data points are for simulations with 1875 elements per processor; the largest problem is 480,000 elements on 256 processors.

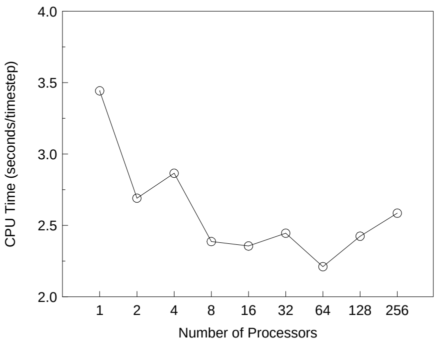

Figure 8: Average CPU time per timestep on the Intel Paragon to crush a container meshed at varying resolutions. The mesh size is 1875 finite elements per processor at every data point.

In contrast to the previous graph, we now see excellent scalability. A breakdown of the timings shows that the performance of the contact detection portion of the code is now scaling as well or better than the FE computation, which was our original goal with this work. In fact, since linear speed–up would be a horizontal line on this plot, we see apparent super–linear speed-up for some of the data points! This is due to the fact that we are really not exactly doubling the computational work each time we double the number of finite elements. First, the mesh refinement scheme we used does not keep the surface–to–volume ratio of the meshed objects constant, so that the contact algorithm may have less (or more) work to do relative to the FE computation for one mesh size versus another. Second, the timestep size is reduced as the mesh is refined. This actually reduces the work done in any one timestep by the serial contact search portion of the contact algorithm (step (6) in Fig. 5), since contact surfaces and nodes are not moving as far in a single timestep. More generally, the number of actual contacts that occur in any given timestep will not exactly double just because the number of finite elements is doubled.

5 Conclusions

The chief advantages of the parallel contact detection algorithm we have proposed are as follows:
(1) The contact surfaces and nodes are nearly perfectly spread across processors, ensuring that the contact detection is load–balanced.
(2) The RCB decomposition technique takes advantage of the fact that the partitioning does not change dramatically from one timestep to the next.
(3) The parallel code can use the same single–processor routine used in the original serial code to perform the actual work of contact detection.

The chief disadvantage of our method is that we must communicate data back–and–forth between the FE and RCB decompositions each timestep. In practice we observed this to be a very minor cost. Almost all of the time in the parallel contact detection was spent performing the RCB decomposition and in the on–processor contact detection effort. There is also a memory cost in our method for the contact surface and node data to be duplicated by the processors that store it in the RCB decomposition. This has not been a major bottleneck for us because the duplication is only for surface elements and because we are typically computationally bound, not memory bound, in the problems that we run with PRONTO–3D.

Acknowledgements

We benefited from helpful discussions about the parallel contact algorithm with David Greenberg and Rob Leland. Martin Heinstein provided insight

into the sequential contact detection algorithm in PRONTO–3D.

References

[1] M. J. Berger and S. H. Bokhari, *A partitioning strategy for nonuniform problems on multiprocessors*, IEEE Trans. Computers, C-36 (1987), pp. 570–580.

[2] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors: Volume 1*, Prentice Hall, Englewood Cliffs, NJ, 1988.

[3] M. W. Heinstein, S. W. Attaway, F. J. Mello, and J. W. Swegle, *A general–purpose contact detection algorithm for nonlinear structural analysis codes*, Tech. Rep. SAND92–2141, Sandia National Laboratories, Albuquerque, NM, 1993.

[4] B. Hendrickson and R. Leland, *The Chaco user's guide: Version 2.0*, Tech. Rep. SAND94–2692, Sandia National Labs, Albuquerque, NM, June 1995.

[5] B. Hendrickson, S. Plimpton, S. Attaway, C. Vaughan, and D. Gardner, *A new algorithm for parallelizing the detection of contacts in finite element simulations*. In preparation.

[6] C. G. Hoover, A. J. DeGroot, J. D. Maltby, and R. D. Procassini, *Paradyn: Dyna3d for massively parallel computers*, October 1995. Presentation at Tri–Laboratory Engineering Conference on Computational Modeling.

[7] M. Jones and P. Plassman, *Computational results for parallel unstructured mesh computations*, Computing Systems in Engineering, 5 (1994), pp. 297–309.

[8] G. Lonsdale, J. Clinckemaillie, S. Vlachoutsis, and J. Dubois, *Communication requirements in parallel crashworthiness simulation*, in Proc. HPCN'94, Lecture Notes in Computer Science 796, Springer, 1994, pp. 55–61.

[9] G. Lonsdale, B. Elsner, J. Clinckemaillie, S. Vlachoutsis, F. de Bruyne, and M. Holzner, *Experiences with industrial crashworthiness simulation using the portable, message-passing PAM-CRASH code*, in Proc. HPCN'95, Lecture Notes in Computer Science 919, Springer, 1995, pp. 856–862.

[10] 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.

[11] L. M. Taylor and D. P. Flanagan, *Update of PRONTO–2D and PRONTO–3D transient solid dynamics program*, Tech. Rep. SAND90–0102, Sandia National Laboratories, Albuquerque, NM, 1990.