Parallel Transient Dynamics Simulations: Algorithms for Contact Detection and Smoothed Particle Hydrodynamics

Steve Plimpton* Steve Attaway Bruce Hendrickson Jeff Swegle
Courtenay Vaughan David Gardner

February 5, 1998

Abstract

Transient dynamics simulations are commonly used to model phenomena such as car crashes, underwater explosions, and the response of shipping containers to high-speed impacts. Physical objects in such a simulation are typically represented by Lagrangian meshes because the meshes can move and deform with the objects as they undergo stress. Fluids (gasoline, water) or fluid-like materials (soil) in the simulation can be modeled using the techniques of smoothed particle hydrodynamics. Implementing a hybrid mesh/particle model on a massively parallel computer poses several difficult challenges. One challenge is to simultaneously parallelize and load-balance both the mesh and particle portions of the computation. A second challenge is to efficiently detect the contacts that occur within the deforming mesh and between mesh elements and particles as the simulation proceeds. These contacts impart forces to the mesh elements and particles which must be computed at each timestep to accurately capture the physics of interest. In this paper we describe new parallel algorithms for smoothed particle hydrodynamics and contact detection which turn out to have several key features in common. Additionally, we describe how to join the new algorithms with traditional parallel finite element techniques to create an integrated particle/mesh transient dynamics simulation. Our approach to this problem differs from previous work in that we use three different parallel decompositions, a static one for the finite element analysis and dynamic ones for particles and for contact detection. We have implemented our ideas in a parallel version of the transient dynamics code PRONTO-3D and present results for the code running on the Pentium-based Intel Teraflop machine at Sandia.

1 Introduction

Large-scale simulations are increasingly being used to complement or even replace experimentation in a wide variety of settings within science and industry. Computer models can provide a flexibility which is difficult to match with experiments and can be a particularly cost-effective alternative

*Sandia National Labs, Albuquerque, NM 87185-1111.
Email: [sjplimp,swattaw,bahendr,jwswegl,ctvaugh,drgardn]@sandia.gov.

when experiments are expensive or hazardous to perform. As an example, consider the modeling of crashes and explosions which can involve the interaction of fluids with complex structural deformations. The simulation of gas-tank rupture in an automobile crash is a prototypical example.

Transient dynamics codes are able to model the material deformations occurring in such phenomena, and smoothed particle hydrodynamics (SPH) provides an attractive method for including fluids in the model. There are several commercial codes which simulate structural dynamics including LS-DYNA3D, ABACUS, EPIC, and Pam-Crash. Of these, EPIC is the only one which includes an SPH capability [12]. PRONTO-3D is a DOE code of similar scope that was developed at Sandia National Laboratories [21, 4, 20] which includes both structural dynamics and SPH models.

A complicated process such as a collision or explosion involving numerous complex objects requires a mesh with fine spatial resolution if it is to provide accurate modeling. The underlying physics of the stress-strain relations for a variety of interacting materials must also be included in the model. Coupling of mechanical and hydrodynamic effects further increases the complexity. Running such a simulation for thousands or millions of timesteps can be extremely computationally intensive, and so is a natural candidate for the power of parallel computers.

Unfortunately, these kinds of simulations have resisted large-scale parallelization for several reasons. First, multi-physics simulations which couple different types of phenomenological models (e.g. structures and fluids), are often difficult to parallelize. A strategy which enables efficient parallel execution of one portion of the model may lead to poor performance of other portions. Second, the problem of contact detection, which is described in the next section, is central to structural dynamics simulations, but has resisted efficient parallelization. We believe our parallel algorithm for this computational task is the first to exhibit good scalability.

In this paper we describe the algorithms we have developed to enable an efficient parallel implementation of PRONTO-3D, coupling structural dynamics with SPH. Although there have been previous attempts to parallelize contact detection and SPH individually, to our knowledge PRONTO-3D is the first code to parallelize them together. A key idea in our approach is the use of different parallel decompositions for each of the three dominant computations that occur in a timestep: finite element analysis, SPH calculation, and contact detection. Although we must communicate information between the different decompositions, the cost of doing so turns out to be small, and thus is more than compensated for by the load-balance and high parallel efficiency we achieve in each individual stage. Our development effort has targeted large distributed-memory parallel supercomputers such as the Intel Paragon and Teraflop and the Cray T3D/E. As such, parallel PRONTO-3D and the algorithms we describe here are written in portable F77 and C with standard message-passing calls (MPI).

In the next section we present a brief overview of transient dynamics simulations including SPH. We also introduce our parallelization strategy and contrast it with previous approaches. We omit a detailed discussion of the numerics of either deformation modeling or SPH, since this can be found in the references and is beyond the scope of this paper. In §3 we highlight two fundamental operations which serve as building blocks for our parallel algorithms. These operations are used repeatedly in the parallelization of SPH in §4 and the contact detection task described in §5. In an earlier version of this paper [19] we discussed PRONTO-3D performance on the Intel Paragon. In §6 we present performance results on the new Pentium-based Intel Teraflop machine.

2 Overview

Transient structural dynamics models are usually formulated as finite element (FE) simulations on Lagrangian meshes [11]. Unlike Eulerian meshes which remain fixed in space as the simulation proceeds, Lagrangian meshes can be easily fitted to complex objects and can deform with the objects as they change shape during a simulation. The deformation of an individual mesh element is computed by calculating the material-dependent stresses and strains imparted by neighboring elements within the topology of the FE mesh. We term this calculation the FE portion of the transient dynamics simulation.

Smoothed particle hydrodynamics (SPH) models are gridless Lagrangian techniques used to model regions of extreme deformation where traditional FE meshes become too distorted and break down [16, 6, 18]. Essentially the material is modeled as a dense collection of particles. In this setting, the term "hydrodynamics" is somewhat of a misnomer, since in a transient dynamics code the particles can carry along stress and strain tensors to model the strength of solid materials as well as liquids. A group of nearby SPH particles is treated as a collection of interpolation points. Spatial gradients of various quantities are computed as a sum of pairwise interactions between a particle and its neighbors and this information is used in material equations-of-state to compute the forces on individual particles. We term this calculation the SPH portion of a hybrid particle/mesh transient dynamics simulation.

For illustration purposes, Fig. 1 shows a schematic of a simulation containing both FE meshes and SPH particles. This is a computation of an airplane wing striking a stationary pole. These are snapshots from simulations performed by John Pott at Sandia to understand fuel dispersal in airplane crashes. The wing and pole are modeled with finite elements; the fuel is modeled as SPH particles. The left snapshot shows the cloud of fuel surrounding the wing after impact; the right snapshot has the fuel particles removed to show details of the tearing of the wing.

In most transient dynamics simulations, there is a third major computation which must be performed every timestep. This is the detection of *contacts* between pairs of unconnected mesh elements and between SPH particles and mesh elements. For example, in Fig. 2, initial and 5 millisecond snapshots are shown of a simulation of a steel rod colliding with a brick wall. Contacts occur any time a surface element on one brick interpenetrates a surface element on another brick. These contacts impart forces to the impacting objects which must be included in the equations-of-motion for the interpenetrating elements. If SPH particles are included in the model, then contacts also occur whenever an SPH particle penetrates a surface element of a meshed object. Finding the set of all such mesh/mesh and particle/mesh contact pairs is termed the contact detection portion of the simulation.

To achieve high performance on a massively parallel machine with a hybrid particle/mesh transient dynamics code such as PRONTO-3D all three of the computational tasks outlined above must be effectively parallelized. We now discuss some of the issues that must be considered in such an effort. Consider a single timestep of the simulation as outlined in Fig. 3. In step (1), the FE portion of the timestep is performed. Within a single timestep in an explicit timestepping scheme, each mesh element interacts with only the neighboring elements it is connected to within the topology of the FE mesh. These kinds of FE calculations can be parallelized straightforwardly. This is also true for implicit schemes if iterative methods are used to solve the resulting matrix equation. In either case, the key is to assign each processor a small cluster of elements so that the only interprocessor communication will be the exchange of information on the cluster boundary with a handful of neighboring processors. 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 to processors is sufficient to insure good performance. A variety of algorithms and tools have been developed that optimize this assignment task. For PRONTO-3D we use a software package called Chaco [8] as a pre-processor to partition the FE mesh so that each processor has an equal number of elements and interprocessor communication is minimized. Similar FE parallelization strategies have been used in other transient dynamics codes [10, 14, 15, 17]. In practice, the resulting FE computations are well load-balanced and scale efficiently (over 90%) when large meshes are mapped to thousands of processors in PRONTO-3D. The chief reasons for this are that the communication required by the FE computation is *local* in nature and the quantity of communication required per-processor is roughly constant if the problem size is scaled with the number of processors.

In step (2), the SPH portion of the timestep is performed. In PRONTO-3D, individual SPH particles are spheres with a variable smoothing length or radius that can grow or shrink as the simulation progresses. The details of how SPH particles interact with each other and how physical quantities such as stresses and forces are derived from these interactions is beyond the scope of this paper, but the interested reader is referred to [4] for a description of the SPH formulation within serial PRONTO-3D. For parallelism considerations, the key concept is that two SPH particles interact if their spheres overlap. A single SPH particle may interact with several nearby neighbors and with a distant particle whose radius is large. Thus an efficient parallel implementation of SPH has two requirements: (1) the number of SPH particles per processor must be balanced, and (2) the spatial region owned by a processor must be geometrically compact to enable neighbors to be found quickly and with a minimum of communication. A static decomposition of particles to processors cannot meet both these demands, since the spatial density of SPH particles can change quite dramatically over time.

There have been a few implementations of parallel SPH calculations described in the literature [22, 23], although SPH is more commonly used for astrophysical simulations than in transient dynamics. The paper by Warren and Salmon gives an excellent overview of issues involved in SPH parallelization and discusses their implementation using an oct-tree approach. This method results in processors owning geometrically compact, although somewhat irregularly-shaped sub-domains. We have opted for a different approach in PRONTO-3D, which is to use recursive coordinate bisectioning (RCB) as a dynamic decomposition technique for the SPH particles. The technique is detailed in the next section, but in essence it assigns a simple rectangular sub-domain to each processor which contains an equal number of particles. We use this technique for several reasons in addition to the fact that it is simpler to implement. First it allows us to take advantage of the fast sorting and searching routines that are already part of serial PRONTO-3D that work within a rectangular sub-domain to optimize the neighbor finding on a single processor. It also produces a smaller boundary region with neighboring processors than does the oct-tree method. Finally, as we shall see in the next sections, the RCB technique is used for both the SPH and contact detection tasks and thus we can reuse the same routine for both purposes.

In step (3) of Fig. 3, the forces previously computed on mesh elements and SPH particles are used to advance their positions and velocities in time. This step is perfectly parallel within the context of the separate decompositions for the FE and SPH steps. In step (4), the element and particle interpenetrations resulting from step (3) must be detected. As illustrated in Fig. 4, several kinds of contacts can occur. In (a), mesh elements from distant regions of one object may come in contact as when a car fender crumples. In (b) elements on two separate objects can also contact, as in the bricks simulation above. Formally, a "contact" is defined as a geometric penetration of a *contact surfaces* (face of a mesh element) by a *contact node* (corner point of an element). Thus in the 2-D representation of (c), node B has contacted surface EF and likewise node F has contacted surface BC. In this context, the centers of SPH particles in PRONTO-3D are treated as additional contact nodes. Thus, in (d), particle B has contacted the top surface of the lower object, but particle A has not.

Although in any one timestep only a few contacts typically occur, checking for all such contacts requires a global search of the simulation domain and in practice can require 30-50% of the overall CPU time when PRONTO-3D runs on a vector machine like the Cray Y-MP. This is because, in principle, any contact node and contact surface can interpenetrate at some time during the simulation. Efficient schemes for spatially sorting and searching lists of contact nodes and surfaces have been devised to speed this computation in the serial version of PRONTO-3D [7].

Unfortunately, parallelizing contact detection is problematic. First, in contrast to the FE portion of the computation, some form of *global* analysis and communication is now required. This is because a contact node and surface pair can be owned by any two processors in the static mesh decomposition described above. Second, load-balance is a serious problem. The contact nodes associated with SPH particles are balanced by the RCB decomposition described above. However, the contact surfaces and nodes come from mesh elements that lie on the surface of meshed object volumes and thus comprise only a subset of the overall FE mesh. Since the static mesh decomposition load-balances the entire FE mesh, it will not (in general) assign an equal number of contact surfaces and nodes to each processor. Finally, finding the one (or more) surfaces that a contact node penetrates requires that the processor who owns the node acquire information about all surfaces that are geometrically nearby. Since the surfaces and nodes move as the simulation progresses, we again require a dynamic decomposition technique which results in a compact sub-domain assigned to each processor if we want to efficiently search for nearby surfaces.

Given these difficulties, how can we efficiently parallelize the task of contact detection? The most commonly used approach [14, 15, 17] has been to use a single, static decomposition of the mesh to perform both FE computation and contact detection. At each timestep, the FE region owned by a processor is bounded with a box. Global communication is performed to exchange the bounding box's extent with all processors. Then each processor sends contact surface and node information to all processors with overlapping bounding boxes so that contact detection can be performed locally on each processor. Though simple in concept, this approach is problematic for several reasons. For general problems it will not load-balance the contact detection because of the surface-to-volume issue discussed above. This is not as severe a problem in [17] because only meshes composed of "shell" elements are considered. Since every element is on a surface, a single decomposition can balance both parts of the computation. However, consider what happens in Fig. 2 if one processor owns surface elements on two or more bricks. As those bricks fly apart, the bounding box surrounding the processor's elements becomes arbitrarily large and will overlap with many other processor's boxes. This will require large amounts of communication and force the processor to search a large fraction of the global domain for its contacts.

An alternative approach, more similar in spirit to our work and which was developed concurrently, is described in [10]. This approach uses a different decomposition for contact detection than for the FE analysis. In their method, they decompose the contact surfaces and nodes by overlaying a fine 1-D grid on the entire simulation domain and mapping the contact surfaces and nodes into the 1-d "slices". Then a variable number of slices are assigned to each processor so as to load-balance the number of contact elements per processor. Each processor is responsible for finding contacts within its collection of slices which it can accomplish by communicating with processors owning adjacent slices. While this approach is likely to perform better than a static decomposition, its 1-D nature limits its utility on large numbers of processors. The implementation described in [10] suffered from some load imbalance on as few as 32 processors of a Cray T3D.

Our approach to parallel contact detection in FE simulations as implemented in PRONTO-3D has been described in [9, 3]. In this paper we generalize the discussion to include SPH particles. The key idea is that we use the same RCB approach described above to dynamically balance the entire collection of contact nodes and SPH particles at each timestep as part of step (4) in Fig. 3. Note however that this is a different decomposition than used in step (2) for the SPH particles alone. In other words, the RCB routine is called twice each timestep, with a different set of input data. The details of how contacts are detected within the context of this new decomposition are presented in §5.

Once contacts have been detected, restoring or "push-back" forces are computed in step (5) that move the interpenetrating mesh elements and particles back to non-penetrating positions. This adjustment is performed in step (6). Both steps (5) and (6) only work on the relatively small set of pairs of contact nodes (including SPH particles) and contact surfaces detected in step (4). Thus they consume only a small fraction of the timestep which is dominated by the computations in steps (1), (2), and (4).

In summary, we have outlined a parallelization strategy for transient dynamics simulations that uses 3 different decompositions within a single timestep: a static FE-decomposition of mesh elements, a dynamic SPH-decomposition of SPH particles, and a dynamic contact-decomposition of contact nodes and SPH particles. In the early stages of this project we considered other parallelization strategies that were less complex, such as balancing particles and mesh elements together across processors. Such approaches have the drawback that they do not load-balance one or more of the 3 computational kernels, an effect that is often fatal to scalable performance on large numbers of processors.

Our final decision to use 3 separate decompositions was motivated by two additional considerations. First, the only "overhead" in such a scheme is the cost of moving mesh and particle data between the decompositions. Once that cost is paid, we have a highly load-balanced decomposition in which to perform each of the three major computational stages within a timestep. For PRONTO-3D this data migration cost turned out to be small in practice. Whether it is *always* small is an open question, but we believe PRONTO-3D is representative of complex multiphysics codes. Second, in each of the three decompositions we end up reducing the global computational problem to a single-processor computation that is conceptually identical to the global problem, e.g. find all contacts or compute all SPH forces within a rectangular sub-domain. This allows the parallel code to re-use the bulk of the intricate serial PRONTO-3D code that performs those tasks in the global setting. Thus the parallel parts of the new parallel version of PRONTO-3D simply create and maintain the new decompositions; the transient dynamics computations are still performed by the original serial code with only minor modifications.

In the following sections we provide more detail as to how these new decompositions are used to parallelize the SPH and contact-detection computations. But first, in the next section, we provide a few details about two fundamental operations that are central to all of the higher level algorithms.

3 Fundamental Operations

Our parallel algorithms for SPH and contact detection involve 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. 5.

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 local copy of a P-length vector with zeroes and stores a 1 in each location corresponding to a processor it needs to send data to. The MPI operation 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.

Another low–level operation used to create both our SPH- and contact-decompositions is recursive coordinate bisectioning (RCB). Initally each processor owns a few "points" which may be scattered anywhere in the domain; in PRONTO-3D the "points" can be SPH particles or corner nodes of finite elements. The RCB operation repeatedly divides the set of points along coordinate axes into 2 subsets until each processor owns a small regular parallelepiped containing N/P points.

The RCB algorithm was first proposed as a static technique for partitioning unstructured meshes [5]. Although for static partitioning it has been eclipsed by better approaches, RCB has a number of attractive properties as a dynamic partitioning scheme [13]. The sub-domains 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 positions of the points being balanced induce only small changes in the partitions. Most partitioning algorithms do not exhibit this behavior.

Table 1 illustrates the speed and scalability of the RCB operation. The time to decompose a 3-d collection of N points on P processors is shown. The times given are an average of 10 calls to the RCB routine; between invocations of RCB the points are randomly moved a distance up to 10% of the diameter of the global domain to mimic the motion of finite elements or SPH particles in a real PRONTO-3D simulation. For fixed P, as the problem size N is increased, the timings show linear scaling once the problem size is larger than about 100 particles per processor. For fixed N, as more processors are used the scaling is not as optimal, but the amount of work performed by the RCB operation is also logarithmic in the number of processors. For example, running RCB on 32 processors means each processor participates in 5 cuts to create its sub-domain; for P=512, each processor participates in 9 cuts. The bottom-line for using RCB in PRONTO-3D as a decomposition tool is that its per-timestep cost must be small compared to the computational work (e.g. SPH particle interaction or FE contact detection) that will be performed within the new decomposition. As we shall see in Section §6, this is indeed the case.

4 Parallel Smoothed Particle Hydrodynamics

Our parallel algorithm for the SPH portion of the global timestep (step (2) of Fig. 3) is outlined in Fig. 6. Since the SPH particles have moved during the previous timestep, we first rebalance the assignment of SPH particles to processors by performing an RCB operation in step (1). This results in a new SPH-decomposition where each processor owns an equal number of SPH particles contained in a geometrically compact sub-domain. This will load-balance the SPH calculations which follow and enable the neighbors of each particle to be efficiently identified.

Once the RCB operation is complete, in step (2) we send full information about only those SPH particles that migrated to new processors during the RCB operation to the new owning processors. This is a much larger set of information (about 50 quantities per particle) than was stored with the RCB "points" in step (1). Step (2) uses the unstructured communication operations detailed in the previous section and is cheap because only a small fraction of the particles move to new processors in any one timestep.

The geometry of the SPH-decomposition can be represented as a set of $P-1$ cuts, where P is the number of processors. One of these cuts is stored by each processor as the RCB operations are carried out. In step (3) we communicate this cut information so that every processor has a copy of the entire set of cuts. This is done optimally via an MPI allgather operation.

Before the SPH calculations can be performed, each processor must know about nearby particles that overlap its sub-domain and are thus potential neighbors of its own particles. This information is acquired in steps (4) and (5). First in step (4), each processor checks which of its SPH particles extend beyond its sub-domain. For those that do, a list of processors whose sub-domain is overlapped is created. This is done using the RCB vector of cuts created in step (3). The information in this vector enables a processor to know the bounds of the sub-domains owned by every other processor. In step (5) needed data for overlapping SPH particles is communicated to the appropriate processors.

Now we are ready to begin computing SPH interactions. As indicated earlier, this consists of two kinds of computations. First, in step (6), pairwise interactions are computed between all pairs of particles which overlap. Each processor determines the interacting neighbors for each of its owned particles. Then the interactions are computed for each pair. This is done in exactly the same way as in the serial PRONTO-3D code (including the sorts and searches for neighbors) with one important exception. It is possible that a pair of interacting particles will be owned by 2 or more processors after extra copies of particles are communicated in step (5). Yet we want to insure that exactly one processor computes the interaction between that pair of particles. We enforce this by defining a unique point, which we call the center-of-interaction (COI) between two spheres A and B. Let x be the point within A which is closest to the center of B, and let y be the point within B which is closest to the center of A. The COI of A and B is (x+y)/2 . A processor computes a pairwise interaction if and only if the COI for that pair is within its sub-domain.

This is illustrated in Fig. 7 for two kinds of cases. In the upper example two particles are owned by processors responsible for neighboring sub-domains. After step (5), both processors will own copies of both particles. However, only processor 0 will compute the pairwise interaction of the particles since the COI for the pair is within processor 0's sub-domain. The lower example illustrates a case where processors 0 and 2 own the particles but the COI is on processor 1. Thus only processor 1 will compute the pairwise interaction for those particles. Note that this methodology adds only a tiny check (for the COI criterion) to the serial SPH routines and allows us to take full advantage of Newton's 3rd law so that the minimal number of pairwise interactions are computed.

When step (6) is complete, as a result of computing the pairwise interactions based on the COI criterion, the processor owning a particular particle may not have accumulated the entire set of pairwise contributions to that particle, since some of the contributions may have been computed by other processors. In step (7) these contributions are communicated to the processors who actually own the particles within the SPH-decomposition. Then in step (8) each processor can loop over its particles to compute their equations-of-state and the total force acting on each particle.

In actuality, steps (5)–(8) are iterated on several times in PRONTO-3D [4], to compute necessary SPH quantities in the proper sequence. From a parallel standpoint however, the calculations are as outlined in Fig. 6: communication of neighbor information, followed by pairwise computations, followed by reverse communication, followed by particle updates.

In summary, steps (2) and (4) require unstructured communication of the kind outlined in Fig. 5. Steps (1), (2), (5), and (7) are also primarily devoted to communication. Steps (6) and (8) are solely on-processor computation. They invoke code which is only slightly modified from the serial version of PRONTO-3D. This is because the SPH-decomposition produced in step (1) reformulated the global SPH problem in a self-similar way: compute interactions between a group of particles contained in a box.

5 Parallel Contact Detection Algorithm

Our parallel algorithm for the contact detection portion of the global timestep (step (4) of Fig. 3) is sketched in Fig. 8. It uses some of the same low–level operations as our SPH algorithm, but in different ways. Recall that for load-balancing purposes we desire a decomposition of all objects that will potentially come in contact (contact nodes, contact surfaces, and SPH particles) that is different than the FE- and SPH-decompositions. In step (1), the current position of each contact node is communicated by the processor who owns and updates it in the FE-decomposition to the processor who owned that node in the contact-decomposition of the previous timestep. (On the first timestep this step is simply skipped.) Likewise current SPH particle positions are communicated from the current SPH-decomposition to the processors who owned the SPH particles in the contact-decomposition of the previous timestep. Both of these operations involve unstructured communication as detailed in §3. The purpose of step (1) is to give the contact-decomposition a starting point that is close to the correctly balanced answer, since the finite elements and SPH particles do not move far in any one timestep. In step (2) we perform an RCB operation as described in §3 on the combined set of contact nodes and SPH particles, based on their current positions.

Once the RCB operation is complete, in step (3) we send full information about the contact nodes and SPH particles to their new contact-decomposition owners. This is a much larger set of information than was sent in step (1) and we delay its sending until after the RCB operation for two reasons. First, it does not have to be carried along as "points" migrate during the RCB stages. More importantly, in step (3) we also send contact surface information to processors in the new contact-decomposition by assigning each contact surface to the contact-decomposition processor who owns one (or more) of the 4 contact nodes that form the surface. By delaying the sending of contact surfaces until step (3), the surfaces do not have to be included in the RCB operation.

Once step (3) is finished, a new contact-decomposition has been created. Each processor owns a compact geometric sub-domain containing equal numbers of contact nodes and SPH particles.

This is the key to load-balancing the contact detection that will follow. Additionally each processor in the new decomposition owns some number of contact surfaces.

Steps (4)–(6) are similar to steps (3)–(5) in the SPH algorithm. The only difference is that now contact surfaces are the objects that overlap into neighboring sub-domains and must be communicated to nearby processors since they may come in contact with the contact nodes and SPH particles owned by those processors. The extent of this overlap is computed by constructing a bounding box around each contact surface that encompasses the surface's zone of influence.

In step (7) each processor can now find all the node/surface and particle/surface contacts that occur within its sub-domain. Again, a nice feature of our algorithm is that this detection problem is conceptually identical to the global detection problem we originally formulated, namely to find all the contacts between a group of surfaces, nodes, and particles bounded by a box. In fact, in our contact algorithm each processor calls the original unmodified serial PRONTO-3D contact detection routine to accomplish step (7). This enables the parallel code to take advantage of the special sorting and searching features the serial routine uses to efficiently find contact pairs [7]. It also means we did not have to recode the complex geometry equations that compute intersections between moving 3-D surfaces and points! Finally, in step (8), information about discovered contacts is communicated back to the processors who own the contact surfaces and contact nodes in the FE-decomposition and who own the SPH particles in the SPH-decomposition. Those processors can then perform the appropriate force calculations and element/particle push-back in steps (5) and (6) of Fig. 3.

In summary, steps (1), (3), (6) and (8) all involve unstructured communication of the form outlined in Fig. 5. Steps (2) and (4) also consist primarily of communication. Steps (5) and (7) are solely on-processor computation.

6 Results

In this section we present timings for PRONTO-3D running on the Intel Paragon and Teraflop machines at Sandia. The latter has 4500 computational nodes, each of which has 128 Mbytes of memory and two commodity 200 Mhz Pentium-Pro processors [1]. The nodes are configured in a 2-D mesh with a proprietary communication network that supports 350 Mbyte bandwidth between nodes (in the limit of long messages) with 20-30 microsecond latencies.

Fig. 9 shows the results of a parallel PRONTO-3D simulation of a steel shipping container being crushed due to an impact with a flat inclined wall. The front of the figure is a symmetry plane; actually only one half of the container is simulated. As the container crumples, numerous contacts occur between layers of elements on the folding surface. We have used this problem to test and benchmark the implementations of our parallel FE analysis and contact detection algorithms in PRONTO-3D.

The first set of timing results we present is for a fixed-size problem containing 7152 finite elements. Both the container and wall were meshed 3 elements thick, so roughly 2/3 of the elements are on a surface. Since each surface element contributes both a surface and node, there were about 9500 contact surfaces and nodes in the problem. The average CPU time per timestep for simulating this problem on various numbers of Intel Paragon processors from 4 to 128 is shown in Fig. 10. Whether in serial or parallel, PRONTO-3D spends virtually all of its time for this calculation in two portions of the timestep – FE computation and contact detection. For this problem, both portions of the code speed-up adequately on small numbers of processors, but begin to fall off when there are only a few dozen elements per processor.

Fig. 11 shows performance on a scaled version of the crush simulation where the container and wall 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 refining the mesh in a given dimension. Thus the leftmost data points are for a 3750 element simulation running on 2 processors; the rightmost points are for a 6.57 million element simulation on 3504 processors of the Intel Teraflop machine. The topmost curve in Fig. 11 is the total CPU time per timestep averaged over a 100 microsecond (physical 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 lower curve is the portion of time spent in the FE computation. Contact detection is the time between the lower and middle curves. Additional overhead, including the contact push-back, is the time between the top two curves.

In contrast to the previous graph, we now see excellent scalability to thousands of processors. Linear speed-up would be horizontal lines on this plot; the FE computation scales nearly perfectly. Contact detection consumes a roughly constant fraction of the time for all runs. The variations in these times are primarily due to the changing surface-to-volume ratios of mesh elements as refinement is done in different dimensions. The total CPU time begins to rise in a non-scalable way on the largest $P = 2048$ and $P = 3504$ runs because the normally small push-back computation becomes somewhat unbalanced on very large numbers of processors.

A few comments about raw performance on the Teraflop machine are in order. The horizontal axis in Fig 11 actually represents "nodes" of the machine, not processors. As mentioned above, each computational node on the Teraflop has 2 Pentium processors, but the 2nd "co"-processor cannot be used directly without special non-portable coding. The FE kernels in PRONTO-3D, represented by the lower curve in Fig 11, were modified and tuned by Ted Barragy of Intel to run in co-processor mode on the Teraflop at about 120 Mflops on a single node. The majority of the remaining time for contact search is primarily integer-based searching and sorting. Even with this non-floating-point overhead however, the aggregate flop-rate on the 3504-node, 6.57 million element run still reached 76 Gflops. A larger version of the container crush problem with 13.8 million elements ran at a sustained rate of 120 Gflops on 3600 computational nodes of the machine. These numbers formed the basis of a performance entry that was a finalist in the 1997 Gordon Bell competition at last years Supercomputing '97 conference [2]. More importantly, for the analysts who use PRONTO-3D, a 2-processor Teraflop node is now nearly as fast as a single processor of a Cray Jedi machine running a version of PRONTO-3D optimized for that platform. Thus the Teraflop machine can perform simulations thousands of times faster than they could be performed previously on traditional vector processors.

The final set of timing results are for a parallel PRONTO-3D simulation of a metal penetrator impacting a target at approximately 700 ft/sec. These are simulations performed by Kurt Metzinger at Sandia. The complex geometry of the penetrator is modeled with 155,000 finite elements; the target is modeled with 415,000 SPH particles since it undergoes large deformations during the impact. Fig. 12) shows a cutaway view after the penetrator has nearly passed through the target. Timing data for this fixed-size problem running on the Intel Teraflop is shown in Table 2 on varying numbers of processors. Average per-timestep timings are shown for the FE computation, the SPH computation, and the contact detection which in this case includes both mesh-mesh and mesh-particle contacts.

Due to memory limitations the smallest number of processors (again, actually nodes) this problem could be run on was 64. The last column in the table indicates the speed-up relative to the 64-processor run as a baseline. The timing data shows that PRONTO-3D is maintaining scalable performance for hybrid FE/SPH problems where all 3 decompositions are active every timestep.

7 Conclusions

We have described the parallelization of the multi-physics code PRONTO-3D, which combines transient structural dynamics with smoothed particle hydrodynamics to simulate complex impacts and explosions in coupled structure/fluid models. Our parallel strategy uses different decompositions for each of the three computationally dominant stages, allowing each to be parallelized as efficiently as possible. This strategy has proven effective in enabling PRONTO-3D to be run scalably on hundreds of processors of the Intel Paragon.

We use a static decomposition for finite element analysis and a dynamic, geometric decomposition known as recursive coordinate bisectioning for SPH calculations and contact detection. RCB has the attractive property of responding incrementally to small changes in the problem geometry, which limits the amount of data transferred in updating the decomposition. Our results indicate that the load balance we achieve in this way more than compensates for the cost of the communication required to transfer data between decompositions.

One potential concern about using multiple decompositions is that mesh and particle data may be duplicated, consuming large amounts of memory. We reduce this problem by using a dynamic decomposition of the SPH particles, so that each particle is only stored once (although the processor which owns it may change over time). However, we do suffer from data duplication in the contact detection portion of the code both for contact surfaces and nodes and for SPH particles. This has not been a major bottleneck for us because the duplication is only for surface elements of the volumetric meshes and because we are typically computationally bound, not memory bound, in the problems that we run with PRONTO-3D.

8 Acknowledgements

This work was performed at Sandia National Laboratories which is operated for the Department of Energy under contract No. DE–AL04–94AL8500. The work was partially supported under the Joint DOD/DOE Munitions Technology Development Program, and sponsored by the Office of Munitions of the Secretary of Defense.

References

[1] WWW address: http://mephisto.ca.sandia.gov/TFLOP/sc96/index.html.

[2] S. Attaway, T. Barragy, K. Brown, D. Gardner, B. Hendrickson, S. Plimpton, and C. Vaughan, *Transient solid dynamics simulations on the sandia/intel teraflop computer*, in Proc. Supercomputing '97, ACM/IEEE, November 1997.

[3] S. Attaway, B. Hendrickson, S. Plimpton, D. Gardner, C. Vaughan, M. Heinstein, and J. Peery, *Parallel contact detection algorithm for transient solid dynamics simulations*

using PRONTO3D, in Proc. ASME Intl. Mech. Eng. Congress & Exposition, ASME, November 1996.

[4] S. W. Attaway, M. W. Heinstein, and J. W. Swegle, *Coupling of smooth particle hydrodynamics with the finite element method*, Nuclear Eng. Design, 150 (1994), pp. 199–205.

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

[6] R. A. Gingold and J. J. Monaghan, *Kernel estimates as a basis for general particle methods in hydrodynamics*, J. Comp. Phys., 46 (1982), pp. 429–453.

[7] 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.

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

[9] B. Hendrickson, S. Plimpton, S. Attaway, C. Vaughan, and D. Gardner, *A new parallel algorithm for contact detection in finite element methods*, in Proc. High Perf. Comput. '96, Soc. Comp. Simulation, April 1996.

[10] 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.

[11] T. J. R. Hughes, *The finite element method - linear static and dynamic finite element analysis*, Prentice Hall, Englewood Cliffs, NJ, 1987.

[12] G. R. Johnson, E. H. Petersen, and R. A. Stryk, *Incorporation of an SPH option in the EPIC code for a wide range of high velocity impact computations*, Intl. J. Impact Engineering, 14 (1993), pp. 385–394.

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

[14] 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.

[15] 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.

[16] L. B. Lucy, *A numerical approach to the testing of the fission hypothesis*, Astro. J., 82 (1977), pp. 1013–1024.

[17] J. G. Malone and N. L. Johnson, *A parallel finite element contact/impact algorithm for non-linear explicit transient analysis: Part II – parallel implementation*, Intl. J. Num. Methods Eng., 37 (1994), pp. 591–603.

[18] J. J. Monaghan, *Why particle methods work*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 422–433.

[19] S. Plimpton, S. Attaway, B. Hendrickson, J. Swegle, C. Vaughan, and D. Gardner, *Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics*, in Proc. Supercomputing '96, ACM/IEEE, November 1996.

[20] J. W. Swegle and S. W. Attaway, *On the feasibility of using smoothed particle hydrodynamics for underwater explosion calculations*, Comp. Mech., 17 (1995), pp. 151–168.

[21] 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.

[22] T. Theuns and M. E. Rathsack, *Calculating short range forces on a massively parallel computer: SPH on the connection machine*, Comp. Phys. Comm., 76 (1993), pp. 141–158.

[23] M. S. Warren and J. K. Salmon, *A portable parallel particle program*, Comp. Phys. Comm., 87 (1995), pp. 266–290.

Figure 1. *Combination finite element and smoothed particle hydrodynamics simulation.*

Figure 2. *Simulation of a steel rod hitting a brick wall.*

(1)	Compute FE forces on each mesh element
(2)	Compute SPH forces on each particle
(3)	Move mesh elements and particles
(4)	Detect mesh/mesh and particle/mesh contacts
(5)	Generate push-back forces
(6)	Final update of mesh and particle positions

Figure 3. *One timestep of a PRONTO-3D calculation.*

Figure 4. *Various types of mesh/mesh and particle/mesh contacts.*

(1)	Form vector of 0/1 denoting who I send to
(2)	Reduce-scatter vector over all P processors
(3)	$nrecvs = \text{vector}(q)$
(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 all my data

Figure 5. *Parallel algorithm for unstructured communication for processor q.*

Table 1. *CPU seconds to perform an RCB decomposition on varying numbers of points N using different numbers of processors P on the Intel Paragon.*

```
(1)  Perform RCB on SPH particles to rebalance
(2)  Send full data for migrated SPH particles to new procs
(3)  Share RCB cut info with all processors
(4)  For all my particles
         If particle extends beyond my sub-domain
             Determine which processors it overlaps
(5)  Send overlapping particle info to nearby processors
(6)  Compute pairwise interactions with COI in my sub-domain
(7)  Communicate results back to processors owning particles
(8)  Update particle equations-of-state, forces, etc.
```

Figure 6. *A parallel algorithm for smoothed particle hydrodynamics.*

Figure 7. *Upper: A pair of SPH particles owned by two processors whose center-of-interaction (COI) shown by an "x" is owned by the left processor. Lower: A pair of SPH particles owned by two processors whose COI is owned by neither.*

```
(1)  Send minimal contact node and SPH data to old RCB decomposition
(2)  Perform RCB to rebalance
(3)  Send full contact and SPH data to new RCB decomposition
(4)  Share RCB cut info with all processors
(5)  For all my surfaces
         If surface extends beyond my sub-domain
             Determine which processors it overlaps
(6)  Send overlapping surfaces to nearby processors
(7)  Find contacts within my sub-domain
(8)  Send contact results to FE and SPH owners
```

Figure 8. *A parallel algorithm for contact detection.*

Figure 9. *Simulation of a crushed shipping container from initial impact to final state after 3.2 milliseconds.*

Figure 10. *Average CPU time per timestep to crush a container with 7152 finite elements on the Intel Paragon. The dotted line denotes perfect speed-up.*

Figure 11. *Average CPU time per timestep on the Intel Teraflop machine to crush a container meshed at varying resolutions. The mesh size is 1875 finite elements per processor at every data point. The lower curve is finite element computation time; the middle curve includes both FE and contact detection time; the upper curve is total CPU time including contact push-back.*

Figure 12. *Mixed FE/SPH simulation of a penetrator striking a target.*

Table 2. *Breakdown of CPU seconds per timestep to perform a penetrator simulation on different numbers of processors P of the Intel Teraflop machine.*