Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

J. Parallel Distrib. Comput. 64 (2004) 266–276

Journal of Parallel and Distributed Computing

http://www.elsevier.com/locate/jpdc

A parallel rendezvous algorithm for interpolation between multiple grids

Steven J. Plimpton,* Bruce Hendrickson, and James R. Stewart

Sandia National Laboratories, Albuquerque, NM 87185-1111, USA

Received 29 August 2002; revised 3 October 2003

Abstract

A number of computational procedures employ multiple grids on which solutions are computed. For example, in multi-physics simulations a primary grid may be used to compute mechanical deformation of an object while a secondary grid is used for thermal conduction calculations. When modeling coupled thermo-mechanical effects, solution data must be interpolated back and forth between the grids each timestep. On a parallel machine, this grid transfer operation can be challenging if the two grids are decomposed across processors differently for reasons of computational efficiency. If the grids move or adapt separately, the complexity of the operation is compounded. In this paper, we describe two grid transfer algorithms suitable for massively parallel simulations which use multiple grids. They use a rendezvous technique wherein a third decomposition is used to search for elements in one grid that contain nodal points of the other. This has the advantage of enabling the grid transfer operation to be load-balanced separately from the remainder of the computations. The algorithms are designed for use within the multi-physics code SIERRA, an object-oriented framework developed at Sandia. Performance and scalability results are given for the grid transfer operation running on up to 1024 processors of two large parallel machines, the Intel Tflops (ASCI Red) and DEC-Alpha CPlant cluster.

Keywords: Multiple grid interpolation; Rendezvous algorithm; Partitioning

1. Introduction

Multiple meshes arise in several kinds of continuum calculations when grids are used to discretize partial differential equations. Examples include the following:

- Iterative linear solvers such as multigrid accelerate convergence by alternating between coarse and fine mesh approximations within a multilevel solution strategy.
- Adaptive mesh codes maintain a tree-like data structure of parent and children elements to expedite refinement or coarsening.
- Codes that solve for multiple physical quantities may use separate grids to solve the appropriate equations for each variable. Consider a machine part that consists of a steel plate with two drilled holes. One hole is a heat sink or source; the other is a location where mechanical stress is applied. As shown in Fig. 1 for a 2d representation, the part is gridded with a high-resolution mesh around the smaller hole to solve the heat equation for thermal conduction while a second mesh with fine resolution around the larger hole is used for the stress/strain mechanics solution. A coupled thermo-mechanical solution is computed by interpolating data back and forth between the two grids. As the simulation progresses, one or both of the meshes may be further adapted to adequately resolve the variables of interest.
- Some operations are more efficient on structured grids, such as radiation transport sweeps or fast Fourier transforms (FFTs). However, complex geometries are better fitted with unstructured grids. In some simulations, both kinds of grids are used during the course of a computation.
- Two different objects may be meshed independently and joined at an interface within a coupled simulation. For 3d objects this creates a 2d interface containing nodal points from both meshes. Solution data from both objects are transferred from one mesh to the other within the interface region during each timestep of the simulation.

*Corresponding author. Fax: +505-845-7442.
E-mail address: sjplimp@sandia.gov (S.J. Plimpton).

0743-7315/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2003.11.006

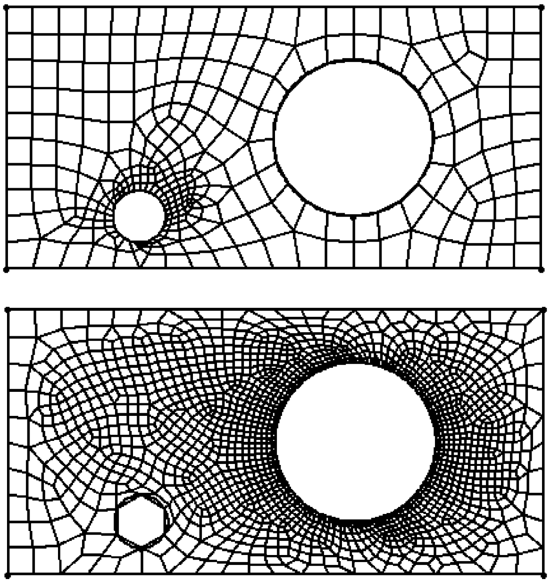

Fig. 1. Two meshes for a 2d plate, each with fine resolution around a different hole.

In all of these settings, solutions on one mesh must be accurately interpolated and transferred to the second mesh for the calculation to proceed. We term this a ‘‘grid transfer’’ operation. In the multigrid and adaptive mesh examples, this is a well-studied problem on parallel machines, where the hierarchical nature of the coarse and fine grids can be exploited for this purpose [3,21,22]. However, in the latter three cases where multiple independent grids are used, the grid transfer operation is complex in a different way. It involves a search for which element of the first mesh contains each nodal point of the second mesh, and a subsequent interpolation. If the two meshes move relative to each other, or if one or both of them adapt independently, then the search operation must be repeatedly invoked.

Now consider implementation of the multi-physics example of Fig. 1 on a distributed-memory parallel machine. A typical timestep consists of computing a solution on the first mesh, interpolating the result to the second mesh, computing a solution on the second mesh, and interpolating it back to the first mesh. We expect the two grid-based computations (finite element or difference or volume) to dominate the run time. Thus for optimal performance, each mesh computation should be independently load-balanced across all processors. Grid partitioning is a well-studied problem; the optimal solution for a parallel machine is one that balances the number of elements on each processor while minimizing the interprocessor communication needed between connected elements. Packages such as Chaco [9] and METIS [13] typically use graph-based algorithms for computing good partitions.

If the two meshes are partitioned independently, there is no assurance that the portions of the two meshes owned by an individual processor will overlap spatially. This is obvious if the two grids themselves only partially overlap, e.g. at the interface between two materials. But it is also true if two meshes of differing resolutions are used to grid the same physical object, as in Fig. 1. It can also be true if the two meshes are initially identical, but adapt over time due to different physical criteria. For example, mechanical deformation could cause grid refinement in one region of the first mesh, while local hot spots could cause the second thermal mesh to refine elsewhere. Insisting that a processor own the portion of both meshes that overlays the same geometric region would obviate the need for a parallel grid transfer operation, but would result in load imbalance or extra communication in one or both of the computational phases. As we discuss in more detail below, multi-constraint partitioning techniques [17,18] can, in principle, be used to reduce these imbalance and communication costs, but have their own shortcomings. By contrast, the algorithms we propose here allow each grid to be decomposed independently and optimally, while incurring the (hopefully small) overhead of the grid transfer operation itself.

In the structured/unstructured grid case, non-overlapping partitions may also be required for good performance. For example, multi-dimensional FFTs are typically performed in parallel by having processors own entire dimensions of a structured mesh (columns or planes of a 3d mesh), so that library routines for fast 1d FFTs can be invoked. If an irregular grid is overlayed with a 3d regular FFT grid, the volume spanned by a processor's FFT columns will not (in general) coincide with the volume of the graph-based partition it owns in the irregular grid.

These issues are the source of algorithmic complexity for general grid transfer operations on a distributed-memory parallel machine. For each of the nodal points it owns in one mesh, how does a processor determine which element in the other mesh contains that node, and what processor owns that element? To what other processor(s) should a processor send its interpolated quantities? Since the resulting pattern of data transfer is irregular and dynamic (if the grids are moving or adapting), how can the communication of grid geometry and interpolated solutions be performed optimally?

Other researchers have addressed some of these issues in different settings. For example, parallel fluid flow calculations often employ overset grids [1,2,23], which are a collection of distinct body-fitted grids that overlap in arbitrary ways. The individual grids are typically topologically regular, only move by bulk translation or rotation relative to each other, and usually have only small regions of overlap. All of these restrictions simplify the search operation relative to the more general problem we are addressing here.

In [16] the manipulation of distributed data on multiple grids is addressed. The PILGRIM library is presented, developed for unstructured grid applications and motivated by data sets arising in the earth sciences. PILGRIM assumes that a grid's decomposition across multiple processors is globally known, so that each processor knows which processor owns every grid cell. This limits the sizes of grids that can be arbitrarily decomposed and is an assumption we do not make. PILGRIM also represents data dependencies between two grids (e.g. for interpolation purposes) as a sparse matrix and enables the interpolation to be efficiently performed in parallel once a matrix is known. However, it includes no algorithms for creating the data dependence matrix efficiently. By contrast, our grid transfer algorithms are designed to quickly compute these data dependencies in parallel and perform the associated interpolation, though we never explicitly form such a matrix.

A different approach to the problem of parallelizing multi-physics simulations is the multi-constraint partitioning model of Schloegel et al. [17,18]. Their goal is to find a single partitioning of the union of grids such that each computation is load balanced. When successful, this approach would not require grid transfer operations. However, for many of the applications of interest to us, this technique is not readily applicable. For instance, if the two meshes are independent, and related only geometrically, then they cannot be easily combined into a single graph partitioning problem. We are also interested in combining mesh and particle methods, and particle interactions are most naturally described geometrically, not with a mesh or graph. Also, multi-constraint partitioning problems tend to be hard to solve, and so the resulting single partition will be of lower quality than those achieved by standard partitions of two independent problems; the parallel multi-physics models themselves will thus run slower.

The new algorithms we propose address the general problem of transferring data between two (possibly) unrelated grids with arbitrary overlap. An early version of one of the algorithms was presented at the SC98 conference [15]. In the next section, we outline the steps of our parallel algorithms. In Section 3, we discuss their implementation and give performance and scalability results on two parallel machines, the Intel Tflops (ASCI Red) and a DEC-Alpha CPlant cluster. As we shall see, the algorithms are fast enough to be used in a dynamic simulation where one or both grids may move or adapt each timestep.

2. Algorithms

We begin with a precise definition of the grid transfer operation. Assume we have a blue mesh consisting of elements with one or more scalar or vector values defined at its nodal points. We also have a red mesh of elements and nodal points whose spatial extent overlaps that of the blue mesh in some arbitrary way. The grid transfer task is to interpolate from nodal values of the blue mesh onto nodes of the red mesh. This requires first finding the unique blue element that contains each red nodal point, then using the solution data on the nodal (corner) points of the blue element to interpolate new solution values onto the red nodal point. If a red nodal point lies on the face (or edge) between two (or more) blue elements, it can be considered inside either element for interpolation purposes. If a red nodal point is outside the entire blue mesh, then it may be ignored or an extrapolation procedure may be used. In this paper we focus on grid points for which interpolation can be performed.

The geometry of this problem is illustrated for a simple 2d example in Fig. 2. Nodal point A of the red mesh is inside the blue element with corner points 1–4. The solution at nodal points 1–4 can be used to interpolate an accurate solution at any location in that blue element's interior, such as point A. Red nodal point B is outside the blue mesh's domain so no interpolation is possible.

As described in the introduction, on a parallel machine, we allow for the blue and red grids to be partitioned across processors in arbitrary and different ways. We call these partitions the blue and red decompositions, respectively. For general decompositions on a distributed-memory machine, a processor has no global knowledge of what elements or nodes are owned by other processors. Yet to perform a particular interpolation, a processor must have information about both a red node and the blue element that contains it. Our solution to this problem is to have elements and

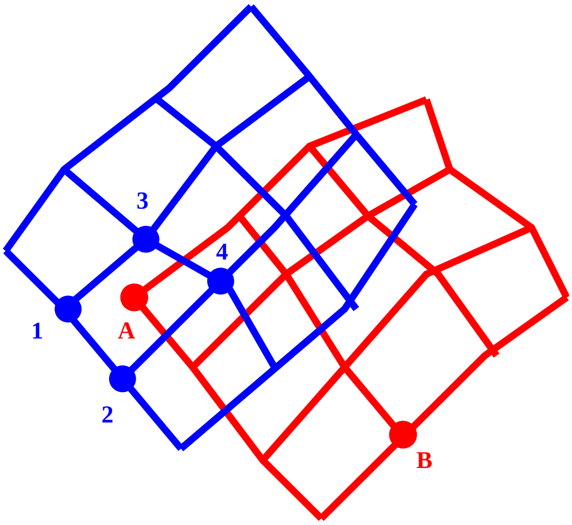

Fig. 2. Arbitrary overlap of two meshes. Node A of the red mesh is inside the blue mesh cell with corner points 1–4.

nodes "rendezvous" in a load-balanced way at specific intermediary processors [11]. We construct a new rendezvous decomposition so that processors in that decomposition own both blue elements and red nodes in the same geometric region of space. To populate this third decomposition with data, processors determine which rendezvous processor needs to be sent the data for each blue element and red node. If each processor can do this independently for the blue elements and red nodes it owns, then no global knowledge of the blue or red decompositions is required.

What properties do we require of the new rendezvous decomposition? First, it should be geometrically based, since the search for blue elements that contain red nodes will take place in this decomposition. Second, it should be fast to compute in parallel, since it may be recomputed every timestep as the blue and red grids move or adapt. The simple geometric partitioning algorithm known as recursive coordinate bisectioning (RCB), first proposed by Berger and Bokhari [4], meets both these criteria. Its name comes from the use of cutting planes normal to the *x*-, *y*-, or *z*-axis. The algorithm takes as input the geometric locations of a set of objects (elements or nodes in this case), determines in which coordinate direction the set of objects is most elongated, and then divides the objects in half by positioning a cutting plane normal to that direction. The processors are likewise split into two groups and each exchanges data as needed with a partner processor in the other group. The original set of points is now split into two halves, each on a subset of processors, which can be further divided by applying the same procedure recursively. With minor modifications, RCB can be used to divide a set of points into an arbitrary number of sets, each containing an equal number of points. Fig. 3 illustrates a typical RCB partitioning for a 2d set of points.

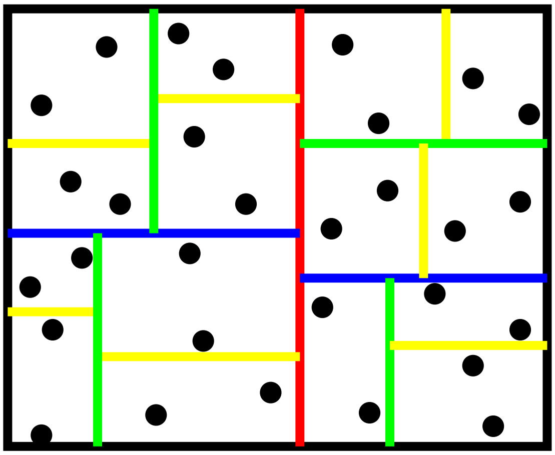

Fig. 3. Recursive coordinate bisectioning of 30 points across 15 processors. The top-level cut is shown in red, the second-level in blue, the 3rd in green, and the lowest-level in yellow.

Although the partitions produced by RCB are not as high quality as those generated by graph-based load-balancing methods, RCB has been used successfully as a parallel dynamic load balancer for a number of applications including adaptive mesh calculations [12], contact detection in crash simulations [14], and molecular dynamics [19]. For the grid transfer problem, RCB has several attractive properties [8]. As required above, it is geometrically based and is fast to generate in parallel; the key step is finding a median of a distributed set of values. The resulting decomposition can be compactly described as a set of $(P-1)$ cutting planes, where P is the number of processors. This means that unlike the blue and red decompositions, where we assumed a processor could not store information about every other processor's partition, for the rendezvous decomposition each processor can easily store the set of cuts that describe the entire decomposition. By stepping through the cutting planes in log(P) stages, any processor can quickly find which rendezvous processor owns the sub-domain containing an arbitrary point.

With this background, we now describe our first Algorithm A for performing a parallel grid transfer operation. As will be discussed below, for load-balancing reasons it is most suitable for meshes whose grid cells are roughly uniform in size. The stages of Algorithm A are listed in Fig. 4.

In step (1) we determine two bounding boxes that contain the blue and red mesh respectively, and intersect them. The resulting intersection box bounds the region of overlap between the two meshes. This inexpensive operation is unnecessary when the two meshes occupy the same region of space. However, in some problems the two meshes may overlap only slightly—e.g. two 3d meshes which overlap at a 2d interface. This step then serves to reduce the number of blue elements and red nodes that need be considered in the remainder of the grid transfer operation.

In step (2) we perform an RCB operation on the set of blue mesh elements which are inside the intersection bounding box defined in the previous step. This creates a new rendezvous decomposition, in which each processor owns a compact region of space containing equal numbers of blue elements. The advantages of using this third decomposition for searching rather than the blue decomposition itself will become clear in the subsequent steps.

In step (3), geometry information about each blue cell is sent from the blue decomposition to the new owner of that cell in the rendezvous decomposition. This is typically a list of corner node coordinates which will be needed for determining whether red nodal points lie within a particular blue element. This information could have been carried along with the blue element as it migrated to a new processor during the RCB operation,

```
(1) Compute a box that bounds the blue and red mesh intersection.
(2) Create a rendezvous decomposition by performing RCB on the blue mesh.
(3) Send element geometry from blue decomposition to rendezvous decomposition.
(4) Clone blue elements which overlap into nearby RCB sub-domains.
(5) Find which rendezvous sub-domain each red node is inside of.
(6) Send node geometry from red decomposition to rendezvous decomposition.
(7) Find which blue element each red node is inside.
(8) Send element/node pairings from rendezvous decomposition to blue decomposition.
(9) Interpolate solutions from blue nodes to red nodes.
(10) Send red node solutions from blue decomposition to red decomposition.
```

Fig. 4. Algorithm A: Steps of a parallel grid transfer operation, exchanging data between three decompositions: blue, red, and rendezvous. This algorithm is best for meshes whose grid cells are roughly uniform in size.

but it is more efficient to represent the element as a single point during RCB. In step (4), blue elements which extend into neighboring processor's RCB sub-domains are identified and cloned. That is, all the information for those elements is shared with the appropriate neighbor processors. Each processor in the rendezvous decomposition now has information for all blue elements whose volumes have any overlap with its RCB sub-domain.

In step (5), each processor computes which rendezvous processor's RCB sub-domain contains each of its red nodal points that lie inside the intersection bounding box of step (1). As in step (4), this computation can be done on-processor (without communication) by using the stored information for the $P-1$ cuts of the RCB rendezvous decomposition. Information about these red points is sent to the appropriate processors in the rendezvous decomposition in step (6).

At this stage of the algorithm, a list of blue elements and red nodal points contained within a rendezvous processor's RCB sub-domain has been communicated to that processor. Step (7) determination of which element contains each point is now a local search. Since it is identical in form to a serial grid transfer operation, existing serial search code with a variety of optimizations can be used. For example, the elements can be first sorted into a data structure which makes the search for each nodal point quicker. Once the search is completed, in step (8) the list of red nodes inside a particular blue element is sent back to the processor which owns the element in the blue decomposition. That processor can then perform the interpolation from blue corner nodes to red nodal points in step (9). Finally, in step (10), the red nodal solution data is sent directly to its final destination in the red decomposition by the blue decomposition processors which performed the interpolation.

In summary, the search and interpolation steps (7) and (9), are on-processor computations for which existing sequential code in the application can be used. The operations of steps (1)–(6) serve to migrate blue elements and red nodes that are close to each other to the same processor so that the search can be computed efficiently. We note that the chief benefit of using RCB to create the rendezvous decomposition is it enables fast and precise identification of which blue elements and red nodal points occupy the same geometric region. If, as an alternative, red points were sent directly to the blue decomposition for searching, there would be no simple way to identify which unique processor should receive them. Simple bounding boxes around a processor's owned blue region could, in principle, be constructed for this task, but the boxes would include (potentially quite large) regions of mesh not owned by the processor, and thus redundant communication and computation would ensue.

We now address the issue of load balance in the grid transfer operation. The computational steps (7) and (9) of Algorithm A will be load balanced if the number of red nodal points inside each blue element is roughly constant across the overall geometry. This will insure that each processor searches on the same number of nodes and elements as other processors (step (7)), and does the same number of interpolations as other processors (step (9)). Geometrically, a variety of grid pairs meet this criterion: two grids with the same uniform spatial resolution, two grids with unequal but uniform resolution (e.g. one coarse and one fine mesh), or two grids with varying spatial resolution but which both coarsen or refine together in the same geometric regions. For all of these scenarios, Algorithm A is a good choice.

However, consider a pair of grids as in Fig. 1. A coarse grid cell near the large hole (upper mesh) will contain a very large number of nodal points in the other mesh. This will lead to imbalance in steps (7) and (9) of Algorithm A. Some processor's RCB sub-domain will likely contain many coarse blue elements. It will then be required to search for a disproportionately large number of red nodal points. Similarly in the blue decomposition, any processor that owns mostly coarse blue elements will perform more than its share of interpolations. We can alleviate this imbalance (at the cost of some additional communication) by balancing the rendezvous decomposition on blue cells and red nodes combined, and by performing the interpolation in the rendezvous decomposition rather than in the blue decomposition. These changes motivate Algorithm B of Fig. 5.

The key differences in Algorithm B versus A are as follows. In step (2), the RCB decomposition is created using both blue elements and red nodal points together. The appropriate elements and nodes can then be sent directly to the appropriate RCB processor in steps (3) and (5). The search step (6) is often better balanced than before since each processor will own the same total number of elements plus nodes within the RCB decomposition. Some processors may own many elements and few nodes, and some vice versa. The binning search algorithm we describe in the next section scales roughly linearly (for typical grids) in both the number of elements and the number of nodes, so this preserves overall balance.

In step (8), the interpolation from blue elements to red nodes is now performed within the RCB decomposition. This requires additional physics quantities on the blue nodal points to be sent from the blue decomposition to the RCB processors in step (7). The final red node results are then sent from the RCB processors to the red decomposition in step (9). Again, this change can help the interpolation step (8) be better balanced than in Algorithm A. If the two grids contain roughly the same number of elements (and nodes), then the distribution of red nodes within the RCB decomposition is at most a factor of two imbalanced in Algorithm B (a processor owns all red nodes and no blue cells). By contrast, in Algorithm A, a processor in the blue decomposition could end up doing interpolations for an arbitrarily large number of red nodes.

In summary, Algorithm B is a better choice than A for grid pairs where the mesh resolution varies within and between the grids, e.g. when the two grids are adapted due to different criteria. The additional cost incurred in B to communicate and store blue grid physics quantities means Algorithm A is a better choice otherwise. A few additional comments concerning variants and details of both algorithms are now in order:

- Nothing in the formulation of the two algorithms is specific to the kind of grids being used. The blue and red meshes can be 2d or 3d, composed of tetrahedra or hexahedra, or a mixture of different element types. We only presume that for the search step it is possible to write a function that can use an element's geometry to quickly determine if a point is inside it or not.
- If the two meshes do not change their nodal positions or topology (e.g. due to refinement) during the next timestep, then the search portion of the two algorithms need not be repeated, if the element/node pairings are stored between timesteps. In Algorithm A, steps (1)–(8) can then be skipped and only the interpolation and communication of steps (9) and (10) need be performed. In Algorithm B, steps (1)–(6) can be skipped, and only steps (7)–(9) need be performed. In the limit of two static meshes, the full algorithm is executed only once as a pre-processing step; just steps (9) and (10) in A or steps (7)–(9) in B are executed every timestep for the remainder of the simulation.
- As mentioned above, RCB is a commonly used algorithm for dynamic load balancing [8]. If the blue mesh were originally decomposed via RCB, then steps (1)–(3) and (8) of Algorithm A could be skipped, since the blue decomposition becomes the rendezvous decomposition. No such savings is possible in Algorithm B since its RCB decomposition is performed on blue elements and red nodes together.
- One particular communication operation occurs repeatedly throughout the two algorithms. Often each processor knows what data it wishes to send to whom, but not which processors it will be receiving data from nor how much data it will receive. We call this an "inverse communication" operation and have written optimized library routines to perform it. Initially a communication "pattern" is constructed, and the communication operation can then be invoked with arbitrary data (e.g. element geometry or solution vectors). To construct the pattern, each processor first determines how many messages it will receive. To do this, each processor constructs a vector of length P with a value of 1 for processor indices that the processor wants to send messages to, and zeros otherwise. Summing these vectors over processors (via an MPI_Reduce_scatter operation) enables each processor to learn the number of messages it will

```
(1) Compute a box that bounds the blue and red mesh intersection.
(2) Create a rendezvous decomposition by performing RCB on the blue elements and red nodes.
(3) Send element geometry from blue decomposition to rendezvous decomposition.
(4) Clone blue elements which overlap into nearby RCB sub-domains.
(5) Send node geometry from red decomposition to rendezvous decomposition.
(6) Find which blue element each red node is inside.
(7) Send element physics from blue decomposition to rendezvous decomposition.
(8) Interpolate solutions from blue nodes to red nodes within rendezvous decomposition.
(9) Send red node solutions from rendezvous decomposition to red decomposition.
```

Fig. 5. Algorithm B: Steps of a second parallel grid transfer operation. This algorithm is best for meshes with widely varying grid cell sizes.

receive. Each processor then sends a brief message to the other processors it has data for, containing the amount of information it plans to send. In this way, each processor learns how much data it will receive from whom, and the subsequent irregular communication operations can be performed efficiently.

- Developing an accurate performance model of the grid transfer algorithms is complicated by the irregular nature of the communication operations, but some asymptotic results are helpful. Let R be the total number of red nodes and B the number of blue cells. The search and interpolation steps are the dominant on-processor computations. Both should scale optimally as $O((B+R)/P)$ and $O(R/P)$, respectively, on P processors, assuming the searches can be performed in linear time (see the next section) and the number of interpolations is roughly load balanced across processors.

The cost of RCB construction in step (2) of the grid transfer algorithms depends on the median finding algorithm. We use a binary search procedure which, if the points are not badly distributed, has a computational cost of $O(M * \log(M)/P)$, where $M = B$ in Algorithm A and $M = B + R$ in Algorithm B. Applying this procedure recursively to successively smaller data sets on successively fewer processors leads to an overall computational cost for RCB that is still $O(M * \log(M)/P)$, with a communication cost that is $O(M * \log(P)/P)$. The cost in step (5) of Algorithm A to determine the rendezvous processor for each red node is $O(R * \log(P)/P)$. In the other communication steps, the total volume of data communicated by all processors is proportional to $R + B$. However, depending on the problem size the communication cost may be dominated by latency instead of bandwidth. The number of messages, and hence the latency cost, can depend in a complicated way on the attributes of the initial red and blue decompositions and their overlap.

3. Results

We have implemented both grid transfer algorithms of the previous section using a collection of load balancing and communication functions that can be assembled in different ways [6,10]. Specifically, the toolkit includes functions for creating RCB decompositions and finding points within them, and for setting up and performing irregular communication in various formats. The toolkit is written in standard C with MPI library calls, but was designed using object-oriented principles. This abstraction serves to isolate internal data structures from the application, such as the storage that encodes irregular communication patterns. The toolkit allowed us to quickly experiment with alternative formulations of the algorithms, such as the A and B versions, as well as other variants. As discussed in [6,10], the toolkit has proven useful in other settings besides grid transfer.

The algorithms of this paper were designed to support a multi-physics finite element code called SIERRA [20,7], developed at Sandia. SIERRA is a C++ computational mechanics framework that allows the application developer to easily couple various physics and material models with different solution techniques. The SIERRA framework provides support for parallel communications, data and mesh management, as well as grid transfer operations.

We present two sets of benchmark timings here. The first are for Algorithm A within a driver that uses idealized grids to let us quickly test the algorithm's performance on a wide range of grid sizes. The second set of results are for Algorithm B using true finite element grids and the associated search and interpolation functions they require. The latter results are similar to what we expect for the eventual performance of grid transfer operations within SIERRA using these algorithms.

Results are given for two different parallel machines at Sandia, one that is a traditional massively parallel machine, the other a less tightly coupled cluster with faster processors but a slower communication network. The first is the Intel Tflops machine built with 333 MHz Pentium processors and a proprietary communications network that sustains processor-to-processor bandwidths of 310 Mbytes/s (in the limit of long messages) with message latencies of 15 ms. The second machine is a cluster known as CPlant [5] built with 500 MHz EV6 DEC Alpha processors interconnected via Myrinet. The Myrinet network delivers roughly 100 Mbytes/s of message bandwidth with latencies of about 60 ms. All of these communication numbers represent what is achieved from the MPI-level within a user application, i.e. by timing MPI_Send and MPI_Receive calls.

To test Algorithm A, two independent 3d regular hexahedral meshes were created with different grid spacings. The first mesh was rotated relative to the other so as to overlap it in an arbitrary fashion. The grid transfer algorithm does not exploit (or even know) that in this case individual grid cells are rectilinear (with one exception noted below) or that the global grids are topologically regular. Each processor is simply given a list of individual cells and associated nodal points to compute on. The rotation of the red mesh causes about 18% of its nodal points to fall outside the global boundaries of the blue mesh. In Algorithm A, step (1) effectively discards these red points. Since the discarded points are not evenly distributed across processors, this causes some load imbalance.

The global sizes of the blue and red meshes were equal in each case and ranged from $N = 8000$ to $N =$

8, 192, 000 grid cells each. Initially the blue mesh was broken into large 3d "bricks", with one brick assigned to each processor, though in a different ordering than the RCB rendezvous decomposition will assign them. The red mesh was partitioned in a columnar fashion with a columnar section assigned to each processor in a quasi 2d decomposition. These choices of blue and red decompositions insure that the typical data exchanges between decompositions in the grid transfer operation (red ↔ rendezvous, blue ↔ red) are irregular in nature and require each processor to send its data to many other processors. The blue ↔ rendezvous exchanges also entail irregular point-to-point communication, but with each processor sending all its data to one other processor, since both decompositions are brick-like.

In each of the following plots, pairs of curves are shown. The first curve (circular data points) is the CPU time in seconds to perform the full grid transfer operation—steps (1)–(10) of Algorithm A of Fig. 4—on both a scalar and a 3-vector quantity interpolated from the blue to red mesh. We use a binning method for the local on-processor search of step (7), which overlays a processor's RCB sub-domain with a 3d grid of bins, assigns blue elements to multiple bins (if they are not wholly contained in a single bin) and red nodal points to unique bins, then checks within a bin to find which element contains a particular red point. For typical grids this procedure scales roughly linearly in the number of elements and points. For this benchmark, the check for a blue element containing a red point is done in a simple way that exploits the fact that each blue hexahedral element is a 3d box aligned with the coordinate axes. The second curve in each plot (square data points) is the CPU time for grid transfer during timesteps when the grids do not move. Steps (1)–(8) of Algorithm A are skipped; only steps (9) and (10) are performed using search results stored from the earlier full invocation of the algorithm.

Fig. 6 shows run times for a fixed-size problem with 64,000 blue elements (and 64,000 red nodal points) run on 1–256 processors. The dotted lines indicate perfect speed-up. On the Tflops machine the algorithm evidences reasonable speed-up until about 128 processors (500 elements/processor) when communication costs begin to dominate. On the CPlant machine the roll-off in performance happens much sooner; communication costs are already a large fraction of the run-time on 8 processors. This is due to CPlant's less favorable compute/communication ratio versus Tflops; a single CPlant processor runs this benchmark 2.5× faster than a Tflops processor (the one-processor timings in Fig. 6), but the CPlant communication network is slower.

Fig. 7 shows timings for a scaled-size problem run on 1–1024 processors of each parallel machine. The blue and red meshes were successively doubled in different dimensions as processors were added so that there were always 8000 blue elements (and 8000 red nodes) per processor. Thus on one processor there were 8000 elements in each global mesh; on 1024 processors each of the two meshes contained over 8 million elements. On this plot perfect scalability for the grid transfer algorithm would thus be a horizontal line. For the full algorithm (circles) the run time rises from about 0.2−0.75 s on Tflops and 0.1−1.4 s on CPlant. Since the on-processor computations (search and interpolation) scale nearly perfectly, the growth in total time is due primarily to communication and to a lesser extent on the logarithmic dependence of the RCB operation on grid size and processor count. On the largest problems, the communication steps move an immense volume of

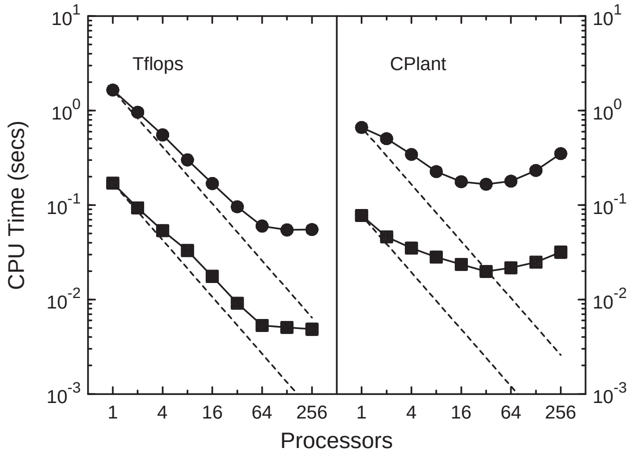

Fig. 6. CPU time on two parallel machines for performing a grid transfer operation using Algorithm A between two fixed-size 64,000 element grids on varying numbers of processors. Circles represent the full algorithm timing; squares are for only interpolation and subsequent solution communication.

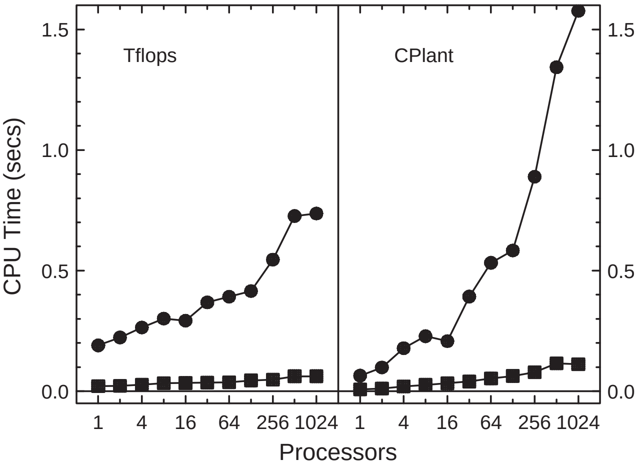

Fig. 7. CPU time on two parallel machines for performing a grid transfer operation using Algorithm A between pairs of scaled-sized grids. Each data point is for grids with 8000 elements per processor. The circles and squares have the same meaning as in Fig. 6.

data between large numbers of processors in an irregular pattern; the communication bandwidth of even the Tflops network eventually saturates and the algorithm runtime increases.

The lower curves (squares) in Fig. 7 are the time to perform only the interpolation and solution communication on timesteps when the meshes do not move relative to each other. Again there is a modest rise in runtime on the largest problems, from 0.021 to 0.062 s/timestep on Tflops and 0.0076–0.11 s/timestep on CPlant, due to communication saturation. However, the most important conclusion from these scaled-size timings is that they indicate a grid transfer operation can be performed quite rapidly for large meshes on both tightly coupled massively parallel machines (like Tflops) and on large commodity clusters (like CPlant).

Algorithm B of Fig. 5 was benchmarked on pairs of finite element grids created for a regular parallelepiped with two cylindrical holes, similar to a 3d version of Fig. 1, except that the two holes pierce different faces of the 3d block so as to be at right angles to each other. The meshes were constructed with 8-node trilinear hexahedral elements. As in the figure, each of the two meshes was locally refined around a different hole. Three different pairs of meshes were used in the benchmarking—"small", "medium", and "large"—with about 18,000, 142,000 and 1.38 million elements in each mesh of the pair. As a pre-processing step, each mesh was decomposed independently using multilevel graph-based partitioning options in the CHACO package [9], which assigns each processor a compact sub-domain from each mesh.

For these meshes, the search step (6) of Algorithm B uses the same binning approach described above (with a bin size set for the larger elements, so that some bins contain a large number of small elements). The low-level test for a red node inside a blue element inverts the finite element shape functions for the blue element. Since these functions contain higher-order monomial cross-terms, this requires inversion of a small nonlinear system which usually converges in a few Newton iterations. Likewise, the interpolations of step (8) were performed using trilinear finite element basis functions.

Fig. 8 shows grid transfer CPU times for the small, medium, and large mesh pairs (blue, green, red data, respectively) on various numbers of Tflops and CPlant processors. The largest grid could not be run on fewer than 16 processors due to memory requirements. These are CPU times to perform two grid transfers, as would be required each timestep in SIERRA in a coupled thermo-mechanical or thermo-fluid simulation. In the first transfer, a 3-vector (e.g. velocity) is transferred from the blue mesh to the red; in the second a scalar (e.g. temperature) is transferred from the red mesh to the blue. As before, two sets of timing data are shown for each case—the "full" time (circles) for steps (1)–(9) of Algorithm B, and a "partial" time (squares) for only steps (7)–(9).

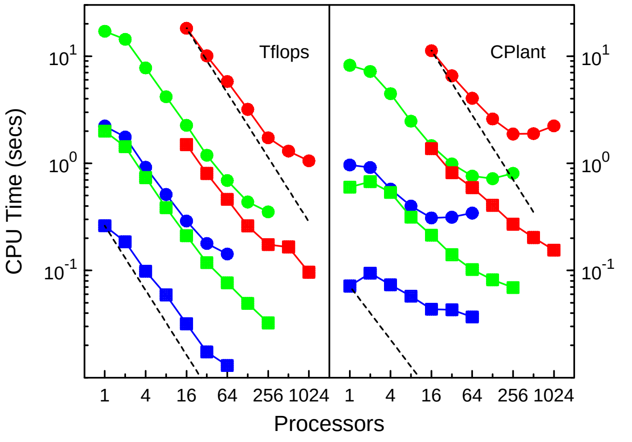

Fig. 8. CPU time on two parallel machines for performing two-grid transfer operations using Algorithm B back and forth between two fixed-sized finite element grids. The blue curves are for meshes with roughly 18,000 elements; the green for meshes with 142,000 elements, the red for 1.38 million elements. Circles represent the full algorithm timing; squares are for only physics communication, interpolation, and subsequent solution communication.

These results indicate the grid transfer operation is somewhat more expensive and less scalable than in the previous plots. The chief reason for the additional computational cost (about 2.5 times greater CPU time per element per transfer on a single processor) is the finite element shape and basis function manipulations now required. Perfect scalability for these fixed-size benchmarks is indicated by the dotted lines shown for various curves in Fig. 8. The fall-off from perfect scaling is similar to that in Figure 7 for Algorithm A, though more pronounced. However once again, the bottom line for the Algorithm B performance is that the grid transfer operation can be done very quickly even for large grids. Preliminary SIERRA timings indicate the full transfer operation should be an order of magnitude or more less expensive than the physics solutions computed on the same meshes.

It is also worth noting that this benchmark is a considerably more stringent test of the grid transfer algorithm with respect to load balance than Fig. 6 case. The size discrepancy (3d volume) between coarse and fine grid cells in these paired grids is as large as a factor of 500–1000, meaning that some blue cells contain hundreds of red nodal points while others have none. Algorithm B cannot completely compensate for this imbalance (the factor of 2× discussed in Section 2) and its extra communication overhead also affects scalability.

Finally, we have compared the performance of Algorithm A versus B on this two-grid problem. As expected from the discussion in Section 2, Algorithm A runs this benchmark less scalably than Algorithm B.

The 1-processor timings are roughly the same for both algorithms. But on 32 Tflops processors Algorithm A performs 2 full transfers about 5.5× slower than Algorithm B on the medium-sized grids (6.61 versus 1.19 s). Similarly on 64 CPlant processors, A runs 3.1× slower than B for the same problem (2.34 versus 0.76 s). These effects are magnified on more processors and larger grid sizes.

In conclusion, these two sets of benchmark results indicate that the parallel interpolation and transfer of solution data between multiple grids can be performed effectively on machines with 1000 s of processors using our proposed algorithms. More importantly, the overall time for the grid transfer operation is typically small compared to the total computation time a production-level application requires to solve PDEs on the same grids, particularly on timesteps where the grids have not changed. Our rendezvous-based approach balances the work of grid transfer independent of the decomposition(s) used by the application for its multiple grids. Thus, it allows the application the freedom to optimize its partitioning and solution strategies as needed for each stage of a coupled multi-physics computation. Finally, by writing routines that perform the low-level communication and load-balancing operations needed for grid transfer in a portable and object-oriented style, we have created a toolkit that has proven useful for quickly experimenting with several variants of our grid transfer algorithms as well as for other applications that require irregular communication.

Acknowledgments

This work was performed at Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract number DE-AC-94AL85000. It was sponsored by DOE's Accelerated Strategic Computing Initiative (ASCI) program. We thank Bob Cochran and Scott Mitchell at Sandia for useful discussions and their help with generating the finite element grids used in our benchmarking.

References

[1] D. Barnette, A user's guide for BREAKUP: a computer code for parallelizing the overset grid approach, Technical Report SAND98-0701, Sandia National Laboratories, Albuquerque, NM, 1998.

[2] E. Barszcz, S. Weeratunga, R. Meakin, Dynamic overset grid communication on distributed memory parallel processors, AIAA-93-3311-CP, 1993.

[3] P. Bastian, K. Birken, K. Johannsen, S. Lang, V. Reichenberger, C. Wieners, G. Wittum, C. Wrobel, A parallel software-platform for solving problems of partial differential equations using unstructured grids and adaptive multigrid methods, in: W. Jager, E. Krause (Eds.), High Performance Computing in Science and Engineering, Springer, Berlin, 1999, pp. 326–339.

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

[5] R.B. Brightwell, L.A. Fisk, D.S. Greenberg, T.B. Hudson, M.J. Levenhagen, A.B. Maccabe, R.E. Riesen, Massively parallel computing using commodity components, Parallel Comput. 26 (2–3) (2000) 243–266.

[6] K. Brown, S. Attaway, S. Plimpton, B. Hendrickson, Parallel strategies for crash and impact simulations, Comput. Methods Appl. Mech. Eng. 184 (2000) 375–390.

[7] H.C. Edwards, J.R. Stewart, SIERRA, a software environment for developing complex multiphysics applications in: K.J. Bathe, (Ed.), Computational Fluid and Solid Mechanics, Proceedings of First MIT Conference, Cambridge, MA, Elsevier, Oxford UK, 2001, pp. 1147–1150.

[8] B. Hendrickson, K. Devine, Dynamic load balancing in computational mechanics, Comput. Methods Appl. Mech. Eng. 184 (2000) 485–500.

[9] B. Hendrickson, R. Leland, The Chaco user's guide: Version 2.0, Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1994, see also http://www.cs.sandia.gov/˜bahendr/chaco.html.

[10] B. Hendrickson, S. Plimpton, Tinkertoy parallel programming: complicated applications from simple tools, in: Proceedings of 10th SIAM Parallel Processing for Scientific Computing Conference, SIAM, Philadelphia, PA, 2001.

[11] W.D. Hillis, The use of rendezvous algorithms in a synchronous SIMD environment is discussed in W.D. Hillis, The Connection Machine, MIT Press, Cambridge, MA, 1985; An example of their use in an asynchronous context is described in J. Eckstein, Distributed versus centralized storage and control for parallel branch and bound: Mixed integer programming on the CM-5, Computational Optimization and Applications, 7 (2) (1997) 199–220.

[12] M.T. Jones, P.E. Plassmann, Computational results for parallel unstructured mesh computations, Comput. Systems Eng. 5 (1994) 297–309.

[13] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, Technical Report CORR 95-035, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1995, see also http://www.users.cs.umn.edu/˜karypis/metis/index.html.

[14] S. Plimpton, S. Attaway, B. Hendrickson, J. Swegle, C. Vaughan, D. Gardner, Parallel transient dynamics simulations: algorithms for contact detection and smoothed particle hydrodynamics, J. Parallel Distrib. Comput. 50 (1998) 104–22.

[15] S. Plimpton, B. Hendrickson, J. Stewart, A parallel algorithm for interpolation between multiple grids, in: Proceedings of SC98, ACM, IEEE, New York, 1998.

[16] W. Sawyer, L. Takacs, A. daSilva, P. Lyster, Parallel grid manipulations in Earth Science Calculations, Lecture Notes in Computer Science, Vol. 1573, Springer, Berlin, 1999, pp. 666–679.

[17] K. Schloegel, G. Karypis, V. Kumar, A New Algorithm for multi-objective Graph Partitioning, Lecture Notes in Computer Science, Vol. 1685, Springer, Berlin, 1999, pp. 322–331.

[18] K. Schloegel, G. Karypis, V. Kumar, Parallel static and dynamic multi-constraint graph partitioning, Concurrency Comput. Practice Experience 14 (3) (2002) 219–240.

[19] S.G. Srinivasan, I. Ashok, H. Jonsson, G. Kalonji, J. Zahorjan, Dynamic-domain-decomposition parallel molecular-dynamics, Comput. Phys. Comm. 103 (1997) 44–58.

[20] J.R. Stewart, H.C. Edwards, The SIERRA framework for developing advanced parallel mechanics applications, in: Proceedings of First Sandia Workshop on Large-Scale PDE-Constrained Optimization, Lecture Notes in Computational Science and Engineering, Springer, Berlin, 2002.

[21] R.S. Tuminaro, C.H. Tong, Parallel smoothed aggregation multigrid: aggregration strategies on massively parallel mechanics, in: Proceedings of SC2000, ACM, IEEE, New York, 2000.

[22] R.S. Tuminaro, C.H. Tong, J.N. Shadid, K.D. Devine, D.M. Day, On a multilevel preconditioning module for unstructured mesh Krylov solvers: two-level Schwarz, Comm. Numer. Method Eng. 18 (2002) 383–389.

[23] S. Weeratunga, E. Barszcz, K. Chawla, Moving body overset grid applications on distributed memory mind computers, AIAA-95-1751-CP, 1995.