Automatic Data Distribution for Composite Grid Applications

LORIE M. LIEBROCK[1] AND KEN KENNEDY[2]

[1]*Liebrock-Hicks Research, 2315 County Road, Calumet, MI 49913*
[2]*Department of Computer Science, Rice University, Houston, TX 77251*

ABSTRACT

Problem topology is the key to efficient parallelization support for partially regular applications. Specifically, problem topology provides the information necessary for automatic data distribution and regular application optimization of a large class of partially regular applications. Problem topology is the connectivity of the problem. This research focuses on composite grid applications and strives to take advantage of their partial regularity in the parallelization and compilation process. Composite grid problems arise in important application areas, e.g., reactor and aerodynamic simulation. Related physical phenomena are inherently parallel and their simulations are computationally intensive. We present algorithms that automatically determine data distributions for composite grid problems. Our algorithm's alignment and distribution specifications may be used as input to a High Performance Fortran program to apply the mapping for execution of the simulation code. These algorithms eliminate the need for user-specified data distribution for this large class of complex topology problems. We test the algorithms using a number of topological descriptions from aerodynamic and water-cooled nuclear reactor simulations. Speedup-bound predictions with and without communication, based on the automatically generated distributions, indicate that significant speedups are possible using these algorithms.

1 INTRODUCTION

In the simulation of physical phenomena, connectivity of the physical system is a part of the essence of the problem. Researchers in simulation of specific physical phenomena use their understanding of the topology of the problem in most, if not all, stages of their research. This is reflected in the algorithms they develop and the programs they write. The topology of problems has been traditionally useful in parallelization of regular problems in that it is often taken advantage of explicitly whenever a physical simulation application is parallelized for a distributed memory machine [1]. Problem topology is the connectivity of the problem, to which we add consideration of dimensionality and size. In regular problems, each problem element is directly dependent on only a fixed set of neighbors, e.g., the north, south, east, and west neighbors in a 2-dimensional problem. In languages such as Fortan D [2] and High Performance Fortran (HPF) [3], problems that are regular allow many communication optimizations including generation of blocked and parallel communications. Unfortunately, many applications must sacrifice regularity to some extent for computational efficiency. One such example is found in com-

Received December 1994
Revised February 1996

Scientific Programming, Vol. 6, pp. 95–113 (1997)
CCC 1058-9244/97/010095-19

plex topology problems, such as the simulation of airflow over an aircraft, in which many regular meshes are connected to allow use of efficient algorithms without requiring the simulation of inactive space. Applications of this type are called multiblock, composite grid, or irregularly coupled regular mesh (ICRM) problems. Shaw and Weatherill [4] provide an introduction to the problem of automatic grid generation and illustrate difficulties associated with multiblock problems:

> The fundamental problem that is encountered in the construction of structured, body-conforming meshes for general aerodynamic configurations is that each component of the configuration has its own natural type of grid topology and that these topologies are usually incompatible with each other. In other words, in attempting to discretize the flow domain around an arbitrary set of 2-dimensional shapes, or some complex 3-dimensional shape, a direct conflict arises between the maintenance of a globally structured grid and the preservation of a grid which naturally aligns itself with the local geometric features of a configuration.
>
> This inconsistency has motivated the development of a general category of mesh construction techniques known as multiblock or composite grid generation. Here, the single set of curvilinear coordinates, inherent to a globally structured grid, is replaced by an arbitrary number of coordinate sets that interface node to node with each other at notional boundaries within the physical domain. Returning to the mapping concept, the approach can be viewed as the decomposition of the flow domain into subregions, which are referred to as blocks, each of which is transformed into its own unit cube in computational space. Global structure within a grid is sacrificed, but the concept proves the flexibility of connections required to construct a grid whose topological structure, local to each component of a complex configuration, is compatible with the particular geometric characteristics of the component. All points that are connected by a common coordinate system can be directly referenced with respect to each other, but bear no direct structured relationship to any of the grid points that lie in coordinate-sets in other blocks.
>
> Thus, the multiblock technique necessitates information to be defined describing how the blocks connect together . . . (pp. 358–359)

From this description it can be seen that composite grid applications exhibit partial regularity. In the past, problems that were not completely regular were treated as irregular for purposes of parallelization. Irregular treatment has typically led to less efficient execution on parallel machines than regular treatment does. In particular, optimizations such as communication blocking and parallelization of communication cannot be applied at compile-time. Problem topology is the key to efficient parallelization support for partially regular applications. Specifically, problem topology provides the information necessary for automatic data distribution and regular application optimization of a large class of partially regular applications.

The two primary goals of this research are to reduce the amount of work that users with a partially regular problem must do to parallelize their application and to take advantage of optimizations developed for regular applications on these partially regular applications. Therefore, an approach is developed that achieves more of the computational efficiency of a regular approach. To do this we exploit not only the regularity inside of each mesh, but also the topological connections between meshes to automatically determine distribution of data structures. A strategy to exploit the partial regularity found in composite grid applications is introduced in this article. In composite grid research, the problem topology includes the connectivity between the meshes. In this case, problem topology is used to automate distribution of the coupled meshes, which, when combined with the automatic HPF program transformation procedure, allows regular application optimization. In order for the application programmer to obtain the same results, he/she would have to determine the distribution for each mesh and perform the entire transformation procedure, including generation and modification of the appropriate clones of all of the computation and coupling update routines. Using the approach presented here, the user is required to do less work and the compiler can perform optimizations developed for regular applications with regular distributions.

As a result of this discussion we have a set of computationally intensive simulation problems, which are difficult to parallelize efficiently, coming from inherently parallel physical phenomena. None of the current parallelization approaches for these problems, e.g., PARTI [5], save the programmer from having to decipher the grid assembly and determine data distributions. Since the description of the grid assembly is normally part of the input, user-supplied data distribution would imply user intervention for each new grid assembly input. Other automatic distribution systems consider only a single mesh or general graphs, thereby loosing regularity for compilation [6–12]. The algorithms presented here explicitly address composite grid applications and preserve regularity to support compile-time optimization of these applications.

An attempt at parallelization of a reactor simulation has been done at Argonne National Laboratories. The preliminary experiments involve parallelization of a code for analysis of thermal-hydraulic and neutronic

events in nuclear reactors and nuclear power plants [13]. This code is a scaled-down, hand-parallelized version of a system's analysis code for parallel experimentation. The best speedups Tentner et al. [13] achieved were two on four processors of a Cray Y-MP/464 and three on eight processors of an Alliant FX/80.

In the next section, sample applications are introduced. Section 3 introduces the programming requirements and transformation procedure which enhance the usefulness of the algorithms presented in this article. The automatic distribution algorithms for composite grid applications are outlined in Section 4. The results of using these algorithms on the sample composite grid applications are presented in Section 5. We conclude in Section 6.

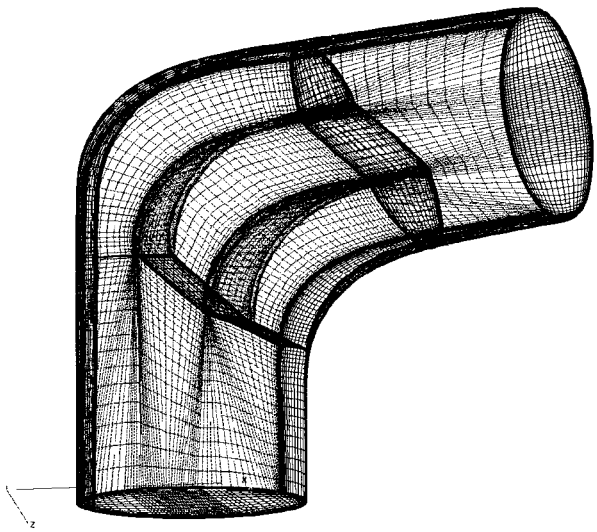

FIGURE 1 Automatically generated grids for flow through an elbow.

2 COMPOSITE GRID APPLICATIONS

Detailed simulation of complex physical phenomena is computationally intensive. An increasing number of these problems involve more than one grid, each of which corresponds to a different physical entity. Two important application areas for composite grid approaches are aerodynamic and nuclear reactor simulation. Problems from these two application areas provide an indication of the spectrum of composite grid application topologies.

Early in this research, problems were solicited from scientists working on composite grid problems. The selection criteria were that the topology and program statistics had to come from real applications and that the specifications had to be obtained in a timely manner. The problem descriptions come from three different application areas.

2.1 Fluid Flow and Aerodynamic Simulations

In these applications, the meshes are large and the computation being done is the same for every element of every mesh. A simple example of a large mesh ICRM problem is the simulation of the material flow through an elbow with two vanes inside. Hugh Thornburg at Mississippi State University generated coupled meshes for this simulation, which are shown in Figure 1. There are five 3-dimensional meshes ranging in size from 44,772 to 70,520 elements with a total of 275,356 elements and six couplings in this problem.

Perhaps one of the best known application areas for composite grids is aerodynamics. In aerodynamic simulations, different grids are used to resolve flow in the space surrounding the fuselage, wings, foreplane, pylons, etc. [14–17]. In recent years, it has become possible to automatically generate the grids used in some ICRM problems [4, 17–22].

The two aerodynamic test problems, which are used for distribution algorithm testing, were generated semi-automatically. The grids for the full F15e simulation are shown in Figure 2. The grid descriptions for both of the aerodynamic examples were provided by Paul Craft working with Dr. Brahat Soni at Mississippi State University. The Fuselage-Inlet-Nozzle problem has 10 meshes ranging in size from 540 to 179,820 elements with a total of 713,766 elements and 25 couplings. In the full F15e simulation, there are 32 meshes ranging in size from 540 to 230,688 elements with a total of 1,269,845 elements and 106 couplings. When grids are generated automatically, it is particularly important that distribution is done automatically. The application programmer must not be required to decipher the automatically generated grid assembly and parallelize the simulation code for a specific configuration by hand.

2.2 Water-Cooled Nuclear Reactor Simulations

In these simulations, not only does the size of the meshes vary greatly, but the computation performed may vary with the mesh. For example, there are 13 different types of components associated with TRAC reactor simulations: ACCUM, BREAK, FILL, PIPE, PLENUM, PRIZER, PUMP, ROD (or SLAB), STGEN, TEE, TURB, VALVE, and VESSEL [23]. Table 1

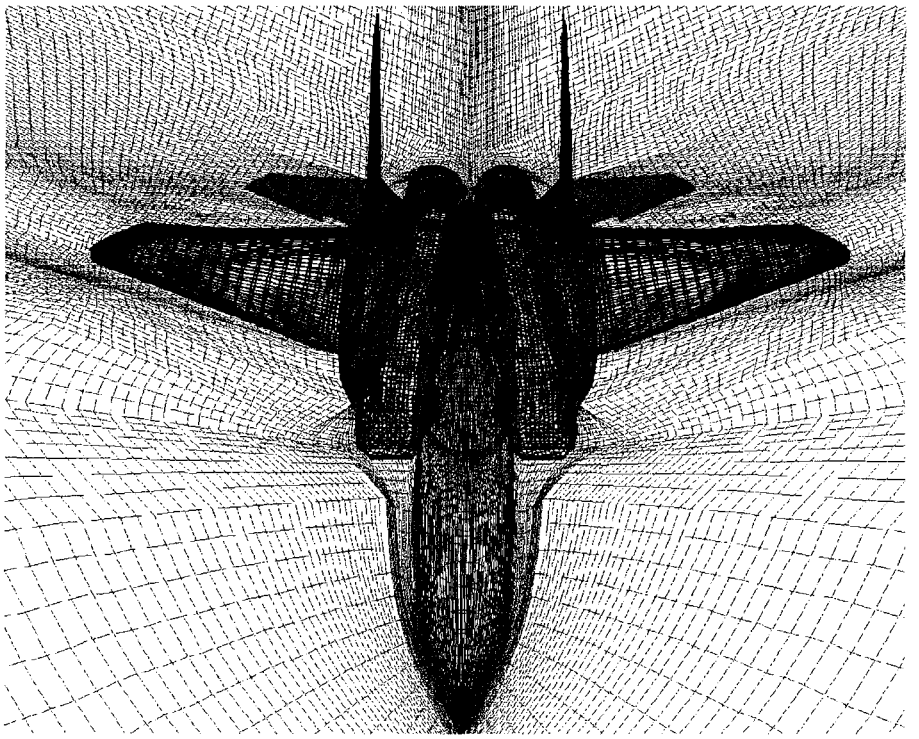

FIGURE 2 F15e volume grid configuration for full aircraft.

shows the approximate number* of additions/multiplications and divisions for each element of 1-dimensional, 2-dimensional (ROD), and 3-dimensional (VESSEL) components [S. J. Jolly-Woodruff and J. F. Lime, Private Communication]. For nuclear reactor applications, mesh and coupling descriptions can be obtained by translating the grid specifications found in the input file for simulators such as TRAC [23]. These values are an average over all the elements of each component type for a specific problem. The actual values also vary by component type.

Table 1. Operation Counts for all Component Types

Component Type	No. Additions/ Multiplications	No. Divisions
1-D	11,798	704
2-D	64,427	888
3-D	117,057	1073

In nuclear reactor simulations, different grids are used for each of the reactor vessels, pipes, pumps, etc. [23]. To provide an indication of the complexity of the topology (dimensionality, size, and connectivity) of these problems, Figure 3 shows just the overview diagram from the Westinghouse AP600 advanced reactor design.† The AP600 design has two loops, with one hot leg, one steam generator, two reactor pumps, and two cold legs in each loop. Figure 3 shows the overall plan of the reactor cooling system, automatic depressurization system, and the passive safety injection system. Each of the systems shown in Figure 3 is composed of numerous grids and heat structures. For more detail on this and other reactor simulations, see [25]. This AP600 reactor configuration has a total of 173 hydrodynamic components (with 1060 3-D cells and 865 1-D cells) and 47 heat structures. This large number of grids and the associated couplings illustrate the complexity of the AP600 topology, which in turn

* These operation counts were obtained from information collected by Susan Jolly-Woodruff using the CRAY hardware performance monitor on a TRAC run with data sets generated by Jim Lime at Los Alamos National Laboratories.

† This figure and the topology specifications for this problem were provided by Jim F. Lime who developed this TRAC model for the AP600 at Los Alamos National Laboratories with support from the Nuclear Regulatory commission's Office of Nuclear Regulatory Research. Preliminary large-break loss-of-coolant accident results were obtained using TRAC-PF1/MOD2 [24].

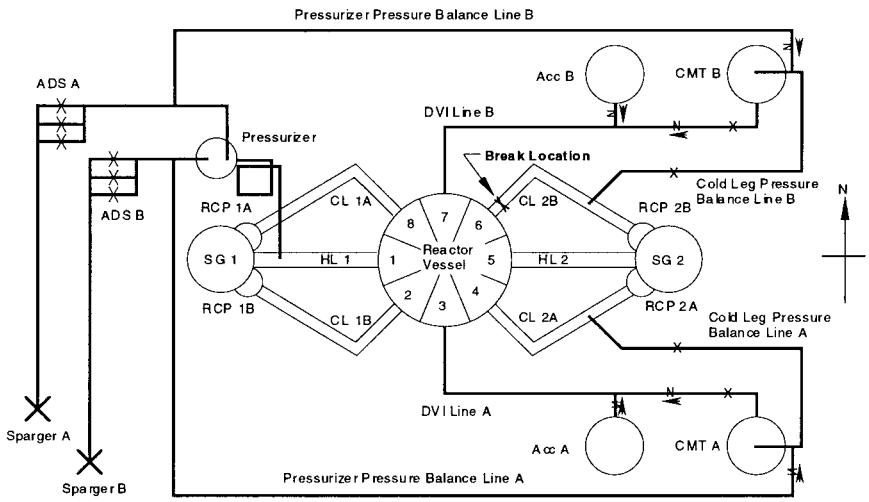

FIGURE 3 Plan view of Westinghouse AP600 model.

illustrates the need for automatic distribution. The AP600 is used as a test problem.

Many composite grid problems require the use of the fastest computers available, even for simplified simulations. For the solution of the grand-challenge simulations in these problems, it is clear that parallelization will be necessary. For example, to verify the design criterion of the AP600 that it be capable of unsupervised operation for 3 days, simulations of many scenarios need to be run. Since simulation of the AP600 reactor currently takes approximately 40 times as long as real time using two communicating workstations (one running RELAP5* and the other running Contain†), each scenario takes about 120 days to run. Fortunately, the physical phenomena being simulated are inherently parallel. Unfortunately, their simulation is not necessarily easy to parallelize. Even with inherently parallel algorithms for composite grid problems, the generation of a parallel program with all of the clones of compute and coupling update routines to allow input of alignment and distribution specification is at best tedious and error prone. Further, support of clones in programs makes code development more tedious and error prone. To make rewriting of applications codes, such as TRAC or RELAP5, for parallel processors practical, the resulting program must be portable, easy to develop, and easy to support [J. Kelly, Private Communication]. For this reason, we will next introduce programming requirements and automatic transformation procedure for HPF programs to support the generation of clones and the use of alignment and distribution specifications. Future code development then occurs on a single version of each compute and coupling update routine and is followed by rerunning the transformation process on the modified subroutine(s).

3 PROGRAMMING REQUIREMENTS AND PROGRAM TRANSFORMATION

A programming style and organization is briefly described that facilitates run-time input of data-mapping specifications, extraction of run-time constant input, analysis of computation and communication for automatic distribution, and automatic program transformation. A procedure is outlined that transforms a user program, written in the specified style, into an HPF program which reads and implements the full alignment and distribution specifications that our algorithms determine. The combination of the use of the programming style, the transformation procedure, and the automatic distribution algorithms, which is the primary focus of this article, eliminates the need for user-specified data distribution and allows the use of optimizations for regular applications on these partially regular ones.

3.1 HPF Programming Requirements for Composite Grid Applications

The composite grid programs that are input to the transformation system are explicitly parallel HPF pro-

* A nuclear reactor simulator developed at Idaho National Engineering Laboratories.

† A simulator for integrated analysis of containment phenomena.

```
      DO N = 1, MaxSteps
C     -- timestep may be computed here
!HPF$ INDEPENDENT
         DO I = 1, NGrids
            UpdtGrid(Grids[i], dt)
         ENDDO
!HPF$ INDEPENDENT
         DO I = 1, NLinks
            UpdtLink(Grids[Links[i].Grid1],
                     Grids[Links[i].Grid2], dt)
         ENDDO
      ENDDO
```

FIGURE 4 Composite grid programming use of INDEPENDENT directive.

grams, but they do not include data-mapping specifications, i.e., TEMPLATE, ALIGN, and DISTRIBUTE directives. Associated with each type of grid there is one or more compute routines (Subroutine UpdtGrid in the example of Fig. 4) that is called to update all of the structures on every timestep. In addition, there is a routine for internal boundary data exchange (Subroutine UpdtLink in Fig. 4) for each pair of coupled grid types. In many composite grid applications, the program will use the HPF INDEPENDENT directive to express the parallelism across the composite grids. For example, consider the simplest type of composite grid application in which all of the grids use the same computational procedure and only one type of coupling update. Then the "NGrids" loop of the code segment in Figure 4 specifies that the computation of the updates for all of the grids is independent and can be done in parallel for each timestep. Note that this subroutine-level parallelism can not be expressed with a FORALL statement. The "NLinks" loop makes a similar specification for the update of the couplings. In general, the coupling update may require multiple loops so that all of the updates for a single loop are independent and do not violate the Bernstein's condition restriction for the INDEPENDENT directive [3, p. 193]. In general, the composite grid program can include any valid HPF constructs except ALIGN, TEMPLATE, and DISTRIBUTE.

Appropriate use of programming style and language features can make development and support more efficient in terms of programmer effort [G. F. Niederauer, Private Communication], while not inhibiting dependence analysis or program transformation. Programmer effort includes the amount of time that a new researcher needs to study the code before being able to make contributions to the application development [J. Kelly, Private Communication]. Further, programs written using the following style should also enable efficient compilation, resulting in High Performance execution.

Two of the most important language features for this discussion are dynamic memory allocation and user-defined data types. User-defined data types allow all of the arrays and scalars for a given grid to be grouped into one data structure with the storage for the arrays allocated at run-time according to the input. Further, an allocatable array of such structures can be used to represent all of the grids of a given type. The basic ideas for data structure creation are that all arrays and scalars for each component type are to be grouped into a user-defined data structure, arrays for each component are to be dynamically allocated, an array of user-defined data structures is to be created for each type of component, and arrays of user-defined data structures are to be allocated dynamically.

HPF requires a strict program form in order to support run-time input of the data to processor mapping. Of particular importance is the appropriate nesting of routines for input and data-mapping specification. The organization required for run-time input of data-mapping specifications is outlined in Figure 5.

Given a program written in this style, the analysis necessary for execution of the automatic distribution algorithm is similar to the interprocedural analysis performed by Rice University's Fortran D compiler [26]. Further, this style permits the use of the transformation procedure described next.

3.2 Program Transformation

Since the HPF language definition does not require interprocedural analysis, which is needed to perform the following transformations, precompiler support must be added for HPF. Interprocedural analysis is needed in order to allow the user to write programs in the form just described, rather than requiring the user to support a program with distribution specifications and subroutine clones. With the use of a precompiler, which can be completely machine independent, the user can compile with any full HPF compiler for the target machine.

An outline of the steps needed for transformation is shown in Figure 6. These steps provide an indication of the complexity of the HPF program that results from the transformation procedure. Of particular importance is that there are potentially many clones of subroutines and it is vital that the user not have to support the clones directly. Each subroutine has an associated clone for each possible data distribution: subroutine clones are a requirement of HPF for efficient compilation to permit optimization without interprocedural analysis. Although not implemented, a more detailed discussion of this transformation procedure, with sample code segments, is presented elsewhere [25, 27].

```
      module composite
C     -- user defined data types go here
C     -- global declarations go here
      contains
      subroutine main
C     -- non-runtime-constant input goes here
C     -- initialization goes here
C     -- all computation goes here
C     -- output goes here
      contains
C     all subroutines except runtime constant input and storage allocation go here
      end subroutine main
C     runtime constant input routines go here
C     storage allocation routines go here
      end module composite

      program icrm
      use module composite
C     -- calls to runtime constant input routines go here
C     -- calls to allocation routines go here
      call main()
      end program icrm
```

FIGURE 5 Required organization for composite grid HPF programs with run-time input of data-mapping specifications.

4 COMPOSITE GRID DISTRIBUTION ALGORITHMS

Minimally, for the algorithms we are about to describe, we need, in addition to the information in coupling specifications, measures of: the amount of computation per element in each mesh, the amount of communication in each dimension of each mesh, and the amount of communication between each coupled pair of elements in each coupling between meshes.

The coupling specifications are used to determine which meshes must communicate. The amount of computation for each element of each mesh is used in load balancing. The amount of communication in each dimension of each mesh is used for communication minimization. The amount of communication for each coupling is used to prioritize the order in which coupled meshes get mapped and to determine the placement of coupled meshes. In nuclear reactor simulations, these measures may vary by component type, but the nature of the computation carried out for each mesh of the same type is similar. The computation can also vary by cell according to the material in the cell and the phase of the material.

```
Add distribution specifications to user defined types (UDTs).
Insert code to read distribution specifications.
Modify storage allocation to implement data distribution.
Add a ''processors'' statement to the main subroutine.
For each subroutine with distributed data:
   For each set of distribution types for the various parameters:
      Clone the subroutine.
      Replace all UDT parameters with their elements and update uses.
      Insert distribution specifications for all distributed data.
      Add a ''processors'' statement.
For each call to a subroutine with distributed parameters:
   Replace all UDT parameters in the call with their elements.
   Add a control structure to select the clone of the callee with the
      proper distribution specifications.
   Insert parameters with distribution information.
```

FIGURE 6 Transformation steps for composite grid HPF programs.

We are currently targeting a torus-based communications topology. This seems reasonable as many available machines either are tori or can have them efficiently embedded in the machine topology.

For efficient parallelization of composite grid applications, we would like to distribute each mesh according to its dimensionality [28]. Once the number of processors to be used on the target machine is fixed, there are only a finite number of possible m-dimensional processor configurations that can be used, where $1 \leq m \leq$ maximum processor dimensionality. This set is limited to only those configurations with at most n dimensions (n is the number of dimensions in the mesh with the maximum number of dimensions) and is called the standard processor configurations.

All meshes are classified according to their size relative to the number of processors being used. Assume for simplicity that the same computation is performed for each element of every mesh. Roughly speaking, it makes sense to distribute a mesh over all processors if such a distribution results in parallel execution that makes efficient use of all processors. Meshes are large if they have enough elements so that it makes sense to distribute them over all of the processors. Eliminate the large meshes from further mesh classification consideration. Meshes are small when they have so few elements that they have less than the amount of work that should be assigned to one processor. Eliminate the small meshes and those left are medium size meshes.

Algorithm development began with large mesh distribution to support applications such as aerodynamic simulations. Next, small meshes, such as those that represent the many pipes, pumps, et cetera, in reactor simulations, were considered. The resulting two algorithms were used for data distribution to obtain the initial results presented in Section 5 and are labeled "without packing." The initial results lead to development of an algorithm to pack medium-sized meshes for distribution. Load imbalance consideration with the use of this packing approach led to the development of a modified small mesh mapping algorithm. The results of the combination of these algorithms with large mesh mapping are presented in Section 5 and are labeled "with packing."

These algorithms could be used at run-time for dynamic load balancing if computation/communication statistics were collected in the program and the new alignment/distribution specifications were used for data redistribution. This should be useful for applications where the amount of computation per cell changes dramatically over time.

Figure 7 presents an outline of the overall algorithm. In the interest of space, only a brief introduction to the algorithms (with and without packing) is presented here. A more detailed discussion with examples is available elsewhere [27].

4.1 Data Distribution without Packing

Data distribution without packing consists of two parts: (1) large mesh distribution and (2) topology-based small mesh distribution. If there are no large meshes, the small mesh with the most computation is mapped to an arbitrary processor before topology-based small mesh distribution begins.

Large Mesh Distribution

The large mesh distribution procedure for selecting a data distribution uses a computational model presented elsewhere [29]. The model provides expected run-time based on application and machine parameters as well as on the selected data distribution.

In large mesh distribution, all meshes are distributed over all of the processors in the same dimensionality [30] and the topology is used to reduce communication. When considering only large meshes, the following steps are performed.

Table Generation. The best mapping and predicted run-time for each template (representing a mesh) on each standard processor configuration is found independently, ignoring coupling communication cost. A table of entries is built, with one entry per standard processor configuration. Each entry consists of an assignment of template dimensions to processor dimensions and the run-time predicted by the model. Only one template dimension may be assigned to any processor dimension and all unassigned dimensions are sequentialized. When two mappings produce the same predicted time, one is selected arbitrarily.

Although an exhaustive search can be used to find the best mapping of a template onto each standard processor configuration, it is not necessary. In some problems, the number of elements in each dimension of a template is the same, but the number of bytes of communication is higher in one dimension than in the others. This makes it more favorable to use fewer processors in the high communication dimension. This leads to a heuristic for reducing the number of mappings that are considered.

1. **Heuristic 1:** *Let the number of elements in each dimension of a template be the same, but let one dimension have more communication than the other dimensions. If the standard processor configuration has fewer processors in one dimension*

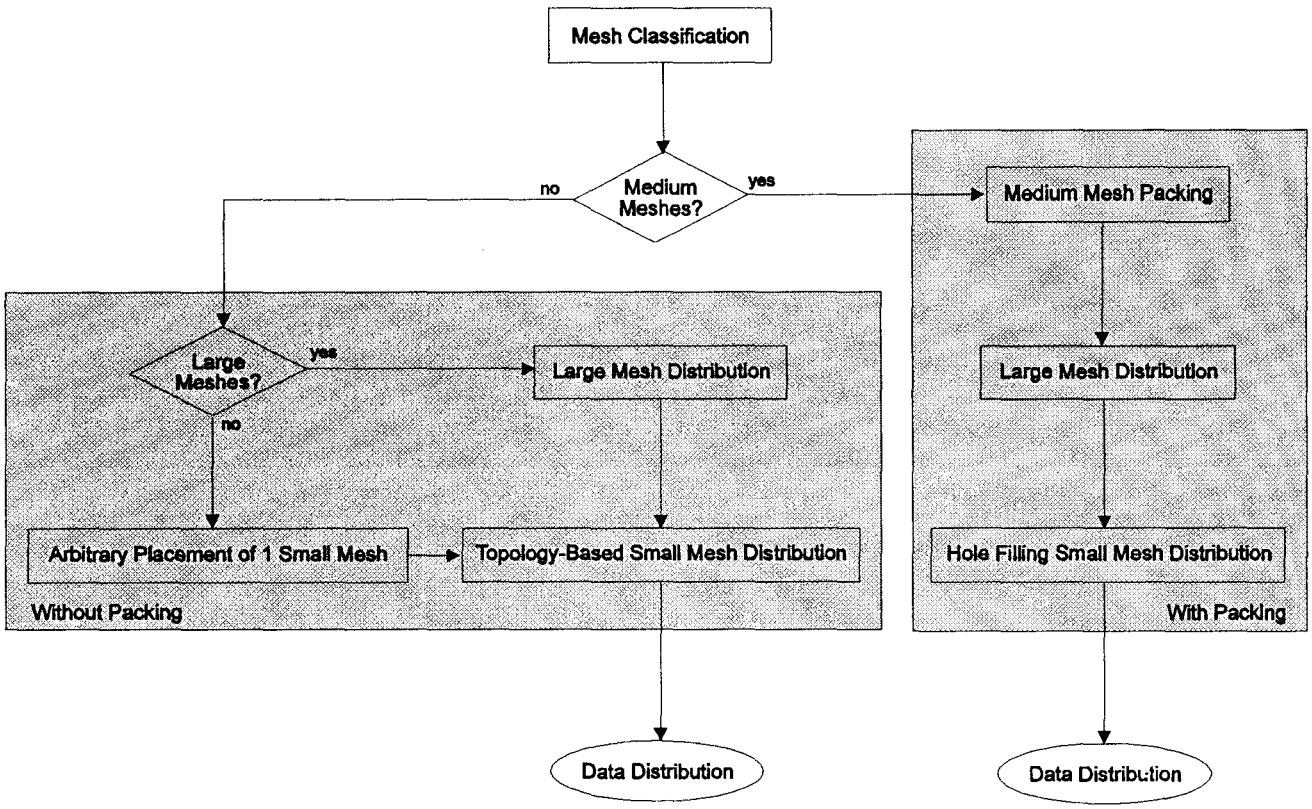

FIGURE 7 Automatic data distribution algorithm flow diagram.

than in the other dimensions, then map the template dimension having greater communication to a processor dimension with fewest processors.

In other problems, the number of bytes of communication is the same in each dimension. This makes the number of elements to be communicated the only communication factor that influences the choice of mapping. Surface-to-volume effects imply that each communication "edge" should be the same length [28]. This leads to the second heuristic for reducing the number of evaluations of the model.

2. **Heuristic 2:** *Let every dimension of a template have the same amount of communication. If the standard processor configuration has fewer processors in one dimension than in the other dimensions, then map the template dimension having the fewest elements to the processor dimension with the fewest processors.*

Finally, consider the case when two dimensions of a mesh are the same size and have the same amount of communication. This implies that the dimensions are equivalent as far as distribution of the mesh is concerned (independent of the rest of the problem). This consideration leads to a final heuristic.

3. **Heuristic 3:** *Let two dimensions of a mesh have the same size and communication. For each pair of mappings with these dimensions interchanged, only one of the mappings needs to be evaluated. Pairs of equivalent dimensions are recorded for possible use during coupling communication reduction.*

In practice, the heuristics are applied only when the number of elements in each dimension of the template is divisible by the number of processors in each dimension. With this qualification, the heuristics usually produce the same results as an exhaustive search. The only exception to this is caused by the use of the last heuristic, where the heuristic sometimes provides better performance because it supports dimension interchange, which the exhaustive search does not.

This phase of the algorithm takes time on the order of the number of templates times the number of standard processor configurations. Note that this is because there is a fixed upper bound on the number of possible

dimension alignments due to Fortran's limitation to seven dimensional arrays.

Global Minimization. The standard processor configuration that produces the minimum total run-time for all templates is found, ignoring coupling communication cost. To find the best standard processor configuration, the predicted run-times for each standard processor configuration are summed over all of the templates and the one with the minimum total predicted run-time is selected.

This phase of the algorithm takes time on the order of the number of standard processor configurations times the number of templates. This step puts the greatest accuracy requirements on the model because predicted run-times are added.

Coupling Communication Reduction. Coupling communication reduction uses the knowledge of topology to reduce communication costs associated with coupling. To do this, the templates are aligned with respect to each other in the distribution via transposing, shifting, or folding around the torus. This is done by aligning the centers of the coupled subranges of dimensions and setting directions (increasing or decreasing) to be the same for each processor dimension. Further, if interchanging the mapping for two equivalent dimensions will reduce coupling cost, the interchange is performed.

For each dimension of every template, a starting processor index and a direction are selected. The processor indices run between one and the number of processors in the processor dimension to which the template's dimension is mapped. The processor index specifies which processor contains the first block of elements for the given dimension of the template. The direction, +1 or −1, specifies whether the elements in the template's dimension will be mapped in increasing or decreasing order.

First, select a template with the single highest coupling communication cost and call it T_1. Set the processor index for T_1 to 1 and the direction to increasing in every parallel dimension. T_1 is now completely mapped to the best standard processor configuration. Make T_1 the only element in the set MAPPED. Create a maximum heap of all couplings involving T_1 with order determined by the coupling costs.

The following steps are repeated until all templates are in the set MAPPED.

1. Delete the maximum element from the coupling heap.
2. If both templates in the coupling are in MAPPED, then return to step 1. Otherwise, call the unmapped template in the coupling T_u and call the mapped template T_m.
3. If interchanging equivalent dimensions of T_u will allow a greater decrease in coupling cost, then do it.
4. Consider each pair of dimensions, where dimension $T_m.i$ is the dimension of T_m coupled to dimension k of T_u and the dimensions are both mapped to the same processor dimension. Related to coupling specification for dimension i of T_m are $T_m.start$, $T_m.end$, and $T_m.stride$. Related to dimension k of T_u are $T_u.start$, $T_u.end$, and $T_u.stride$. Since T_m is already mapped, the $T_m.direction$ and $T_m.start\_proc$ for each dimension are already defined. To map dimension k of T_u, its direction and starting processor must be found.
   A. The direction for dimension k of T_u is the same as that of dimension i of T_m if the coupling stride for each template is either increasing or decreasing. If one template's stride is increasing and the other template's stride is decreasing, then the direction for T_u is the opposite of the direction for T_m.
   B. Determine the offset of the processor that owns the middle entry of the coupling in dimension i of T_m, call it $T_m.Poffset$.
   C. Determine the offset of the processor that owns the middle entry of the coupling in dimension k of T_u, call it $T_u.Poffset$ (computed as if T_u was mapped beginning in processor 1).
   D. Determine the processor that should own the first element of dimension k of T_u. It is $T_m.start\_proc[T_m.i] + T_m.Poffset - T_u.Poffset$.
5. For each pair of coupled dimensions that are not mapped to the same processor dimension, the direction and starting processor are found for T_u as follows, where k is the dimension of T_u being mapped.
   A. The direction for T_u in dimension k is positive; this is arbitrary.
   B. Find the dimension of T_m that is mapped to the same processor dimension as dimension k of T_u, label it $T_m.i$.
   C. Let j be the index in the coupling specification that involves dimension $T_m.i$ of T_m.
   D. Determine the offset of the processor that owns the middle element of the j^{th} coupling entry for T_m, call it $T_m.Poffset$.
   E. Determine the offset of the processor that owns the middle entry of the coupling in dimension k of T_u, call it $T_u.Poffset$ (com-

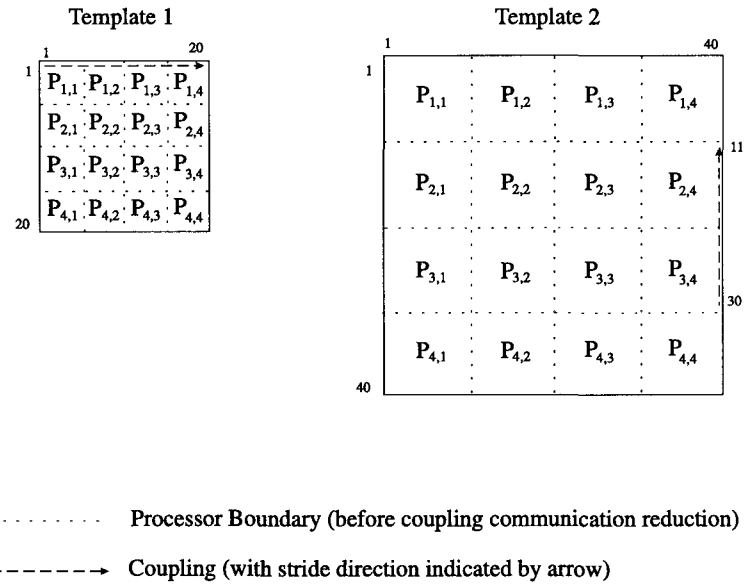

FIGURE 8 Template-to-processor mapping before coupling communication reduction.

puted as if T_u was mapped beginning in processor 1).

F. Determine the processor that should own the first element of dimension k of T_u. It is $T_m.start\_proc[T_m.i] + T_m.Poffset - T_u.Poffset$.

6. For each coupling involving T_u, which has not already been put into the heap, insert it.
7. Add T_u to the set MAPPED.

The time for this stage of the algorithm is bounded by the number of distributed dimensions multiplied by $c \log c$, where c is the number of couplings in the system.

To illustrate the coupling communication reduction stage, we present a simple example. Consider a two-grid problem: Grid 1 is 20 elements by 20 elements and Grid 2 is 40 elements by 40 elements. There is one coupling between these grids with columns 1 to 20 of row 1 in Grid 1 coupled to rows 30 down to 11 of column 40 in Grid 2. Assume the computation is the same on both grids and that the communication is the same in each dimension on both grids. Associate Template 1 with Grid 1 and Template 2 with Grid 2. After global minimization, we arrive at the template to processor mappings shown in Figure 8. Coupling communication reduction for this simple example proceeds as follows.

1. To begin coupling communication reduction, we set the processor index to 1 and the direction to increasing for Template 1 for both dimensions: it is fully mapped.
2. Step 1 selects the only coupling in the problem.
3. Step 2 makes T_m be Template 1 and T_u be Template 2.
4. Step 3 swaps the mapping of template dimensions to processor dimensions for T_u.
5. In Step 4, we see that dimension 1 of T_m is coupled to dimension 2 of T_u and that these dimensions are both mapped to processor dimension 1. From the above description of the problem, the following variable assignments may be derived:

$$T_m.start = 1,\ T_m.end = 1,\ T_m.stride = 1$$

$$T_u.start = 40,\ T_u.end = 40,\ T_u.stride = 1$$

Since there is only one element (40) to the second dimension of the coupling for T_u, the direction for dimension 2 is increasing. The middle element of the coupling for T_m is located on processor 1, so there is no offset ($T_m.Poffset = 0$). The middle element of the coupling for T_u would be located on processor 4, so the offset is 3 ($T_u.Poffset = 3$). Therefore, the mapping for dimension 2 of T_u is shifted to be $T_u.start\_proc[2] = 2$ (recall the architecture is a torus so this will wrap around and -2 is congruent to 2). This shift says that in dimension 2 of T_u the first block of elements gets assigned to processor 2.

6. In Step 4, we see that dimension 2 of T_m is coupled to dimension 1 of T_u and that these dimensions are both mapped to processor dimension 2. From the above description of the problem, the following variable assignments may be derived:

$$T_m.start = 1,\ T_m.end = 20,\ T_m.stride = 1$$

$$T_u.start = 11,\ T_u.end = 30,\ T_u.stride = -1$$

Since the direction (the sign of the stride) for dimension 1 of T_u is different than the direction for dimension 2 of T_m, the direction for dimension 1 is decreasing. This means that mapping starts with the high order end of the template as the first block of elements. The middle element of the coupling for T_m has offset 2 ($T_m.Poffset = 2$). The middle element of the coupling for T_u would have offset 2 ($T_u.Poffset = 2$). Therefore, the mapping for dimension 1 of T_u is not shifted: $T_u.start\_proc[1] = 1$. In this dimension the first processor gets the first block of elements, which is indexed off the high end of the template (40 down to 31).

In this example, there are no coupled dimensions that cause steps 5 and 6 to be used. Step 5 performs realignment similar to that shown in this example, when coupled dimensions are mapped to different processor dimensions. The coupling communication reduction phase for this example results in the mapping depicted in Figure 9. The computation and communication internal to each mesh have not changed, but the communication associated with the coupling and mapping in Figure 8 has been nearly eliminated and made nearest neighbor.

First Mesh Mapping

When there are only small meshes, two preprocessing steps are performed: (1) Standard processor configuration selection: The highest dimensionality, nearly square standard processor configuration is selected for the problem and (2) First mesh mapping: The mesh with the most computation is mapped to an arbitrary processor.

Topology-Based Small Mesh Distribution

This algorithm maps meshes that communicate to the same processor or at least as near as possible. If we simply mapped to the processor with the greatest coupling cost, we would end up with very few processors with small meshes allocated to them. The number of processors used in that case would be bounded by the number of couplings between large and small meshes: one in the case of a single premapped small mesh. Therefore, a load-balance measure is used to determine when a processor has too much computation already allocated to accommodate the work associated with a particular unmapped mesh. When this happens, we try to allocate it to a neighbor processor. If all of the neighbors are too full, the closest processor with no work allocated is found. If this also fails, we save the mesh as a "fill" element to be used after all other meshes are mapped. These "fill" elements are then used in a final load-balancing step that assigns them to underutilized processors.

Small mesh distribution begins by determining the amount of computation, for the small meshes only, that should be assigned to each processor for perfect small mesh load balance. We statically distribute small meshes by trying to obtain load balance within $\pm\ \varepsilon$ of perfect balance. If the large meshes are not perfectly load balanced, then we could add their computation to this consideration to improve the overall load balance. We are not currently starting with large mesh load balance, but will in future work.

Two types of heaps are used in this mapping algorithm. The mesh heap contains all of the unmapped meshes with scheduled communication. Communication is scheduled between a mapped mesh, M, and each coupled unmapped mesh, N, by adding the coupling communication cost to N's communication for the processor to which M is mapped. This introduces the second type of heap. A processor heap is associated with each mesh in the mesh heap. Each entry in a processor heap gives the current coupling cost of the associated mesh to a specific processor. This two-level structure specifies, at each step of the algorithm, the mesh with the maximum communication scheduled to a single processor (from the mesh heap) and the associated processor (from the mesh's processor heap). The essence of the algorithm is to repeatedly attempt to map the unconsidered mesh with the maximum communication involving a single processor to that processor or a nearest neighbor. If the mesh can be so mapped, all structures are updated to reflect the mapping. Otherwise, the mesh is saved for the cleanup stage. In the final cleanup stage of the algorithm, two heaps are also used, one being a maximum heap of meshes sorted according to the amount of computation in the mesh and the other being a minimum heap of processors sorted according to the amount of computation assigned to the processor. The remaining meshes are mapped by repeatedly placing the largest computation mesh on the least loaded processor.

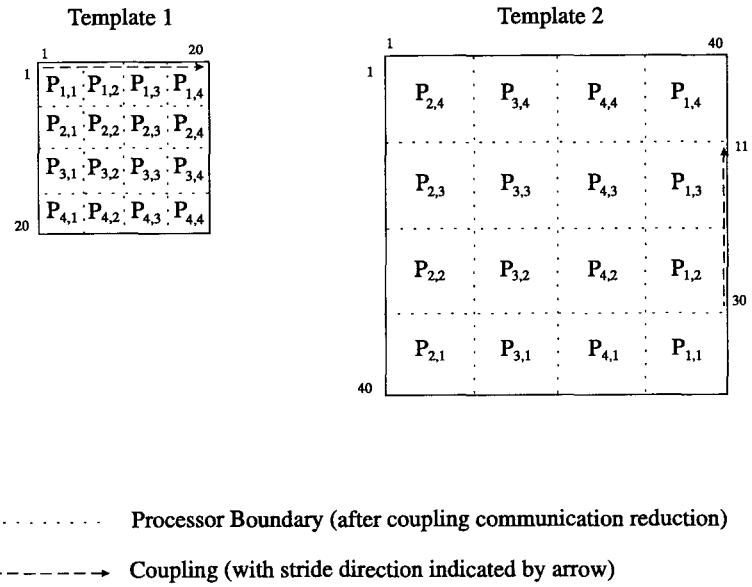

FIGURE 9 Template-to-processor mapping after coupling communication reduction.

From studying the types of coupling communication that occur in the test problems, we learned that there are often many couplings with the same communication cost. This led to exploration of further ordering options in the mesh heap. In the end, a combination of maximum coupling cost to a processor and the computation cost associated with the mesh was used to select the next mesh to map. The coupling cost is still the primary selection criterion with the higher computation cost only used to make the selection when communication cost ties occur.

The run-time of this algorithm is

$$O(M\ log(M)\ C\ log(C)\ P\ log(P))$$

where M is the number of small meshes, C is the maximum number of couplings involving a single mesh, and P is the number of processors.

4.2 Data Distribution with Packing

Data distribution with packing consists of three parts: (1) medium mesh packing, (2) large mesh distribution, and (3) hole-filling small mesh distribution. Notice that in this situation, the packing of medium meshes ensures that there will be at least one large mesh to distribute.

Medium Mesh Packing

All of the medium meshes are packed into some number of large bins for distribution. This allows distribution of all meshes with nearest-neighbor communication for all communications that involve only one mesh. This packing process is outlined in Figure 10. Only meshes having three or fewer dimensions are currently packed. All of the test problems involve meshes with at most three dimensions, so this does not restrict the algorithm for the test problems. Further, the physical phenomena simulations that are being targeted here use at most 3-dimensional meshes. This work can be extended to higher dimensions with the effect that the optimization problem becomes more difficult and expensive. Medium-sized mesh packing consists of the following steps.

Find Bin Bound. An upper bound on the number of bins is found by adding up the number of elements in the meshes and dividing by the number of processors. This assumes that the same computation is taking place on each mesh. If this is not the case, the meshes could be separated into groups according to computational load and then packed.

Perform Packing. Now for each number of bins, up to the maximum, the optimal packing of the meshes into the bins is sought, with each bin being at least big enough to distribute over all processors. The best solution is the set of bins and associated packing that produces the minimum total volume over all of the bins. To do this, a model was developed for this nonlinear optimization problem. The optimization model says to minimize the sum of the volumes of the bins subject to the following constraints:

1. All bin volumes must be at least as large as a precomputed lower bound.
2. All meshes must be completely contained in one and only one bin.
3. No two meshes, assigned to the same bin, may overlap.
4. Each dimension of a mesh assigned to a bin must be assigned to exactly one dimension of the bin.

Since this nonlinear optimization problem is not computationally tractable, the packing problem is solved iteratively via a local linear approximation using the CPLEX callable library [31]. This use of a local linear approximation in optimization implies the addition of a modeling constraint to limit the change that may occur in a single optimization step. This limit is governed by a trust region computation [32].

Build Large. From the best solution, a large mesh is created to represent each composite bin. The size of the large mesh is the size of the composite bin. The computation per element is the maximum of the computation costs for the medium meshes that were packed into the composite.

Translate Couplings. Each new large mesh has all of the couplings translated from the medium-sized meshes it contains into their corresponding positions in the large composite mesh. This translation involves dimension and offset alignment. The direction of alignment is currently increasing, but it could be modified to reduce coupling communication cost between two meshes packed in the same composite bin.

The one modification that was made to this basic optimization procedure is that meshes are prepacked when possible. A prepacking makes a single n-dimensional larger mesh out of two n-dimensional smaller meshes when the sizes in $n - 1$ of the dimensions are the same. In this case, there are no "holes" introduced in the new larger mesh. This was done because the optimization-based packing algorithm is expensive. This modification comes into play in the reactor simulation problems in particular. In the simulation of the AP600, with only 64 processors, this prepacking reduces the number of meshes to be packed using CPLEX from 32 to 11. Details of the linear optimization model may be found elsewhere [27].

Large Mesh Distribution

The large composite meshes are distributed exactly as the meshes were with the original large mesh distribution algorithm. During this distribution, each large mesh is assumed to have the same computation being performed for each element, i.e., the new composite meshes are treated as regular. Note that any large meshes originally in the problem are distributed at the same time as the large meshes that were built during the medium mesh packing process.

Hole Filling Small Mesh Distribution

This algorithm targets improving load balance via small mesh distribution in the presence of packed meshes. Small meshes are mapped by repeatedly putting the unmapped small mesh, with the most computation, onto the least computationally loaded processor. To do this, the following steps are performed.

1. Determine the amount of computation that has been assigned to each processor.
2. Sort processors in increasing computation order using a minimum heap and sort small meshes in decreasing computation order using a maximum heap.
3. While there are unmapped small meshes, re-

```
compute the maximum number of usable bins
For each number of bins up to the number of usable bins
   find a feasible solution as a starting point
    while not converged
       prepare the linear programming problem for CPLEX
       call CPLEX routines to solve the local linear problem
       check to see if the solution has converged
       if not converged
          update the variables for next iteration
       else if the solution is better than any previous solution
          save it
create a large mesh for each bin in the best solution
fill in each large mesh's information from the medium meshes packed in it
translate the couplings from each medium mesh to a large mesh with offsets
```

FIGURE 10 Medium mesh packing algorithm.

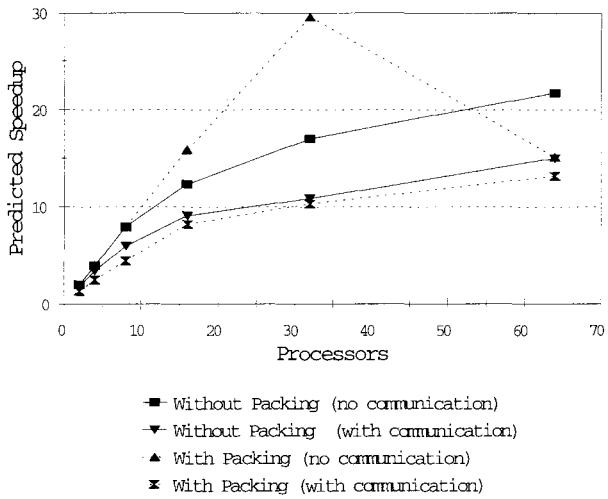

FIGURE 11 Performance predictions for the H. B. Robinson model.

peatedly map the mesh with the most computation onto the processor with the least assigned computation.

The runtime of this algorithm is

$$O(M \log(M)\ P \log(P)).$$

In future work, we will use the ideas developed for the topology-based small mesh distribution algorithm in mapping small meshes according to coupling requirements while trying to improve load balance. This combination will be more expensive, but will also produce better performance.

5 PERFORMANCE PREDICTIONS

Since the applications we are focusing on in this article are not written in HPF, we present predicted speedups based on load balance and communication in place of timing results. The measure of load balance used is the total number of floating-point additions, multiplications, and divisions. The speedup bound, $Dist_{sb}$, is the speedup, with the data distribution supplied, that would be achieved if there were no communication, i.e.,

$$Dist_{sb} = \frac{Computational\ time\ for\ n\ processors}{Computational\ time\ for\ one\ processor}.$$

A second speedup measure is presented that includes a communication factor. This is a loose approximation in the sense that communication network contention is not taken into account and average communication message size and processor distances are used. Computation and communication costs are based on measured values for the Intel i860 (see [29]).

For each of these predicted speedups, we present the value obtained without use of the packing algorithm and the value obtained with use of the packing algorithm. With these measures of speedup we present the results of the distribution algorithms on five problems. Results of other tests are available elsewhere [27].

5.1 H. B. Robinson Reactor Model

The H. B. Robinson model is a small model that has been used to illustrate concepts in the Los Alamos TRAC manuals. Its nodalization came from the TRAC User's Guide [23]. This model has 100 components (144 3-dimensional cells and 433 1-dimensional cells) and 21 heat structures.

The experiment results are summarized in Figure 11. In this and the following figures, results are summarized for the topology-based distribution algorithm with and without the use of packing. $Dist_{sb}$ is presented on the lines labeled "no communication." The lines labeled "with communication" include an approximation for the cost of communication.

On eight processors the speedup bound with communication is 6 without packing and 4.5 with packing as compared with the best speedup of 3 achieved with the hand-parallelized code at Argonne. The speedup bound for the H. B. Robinson model is low on 64 pro-

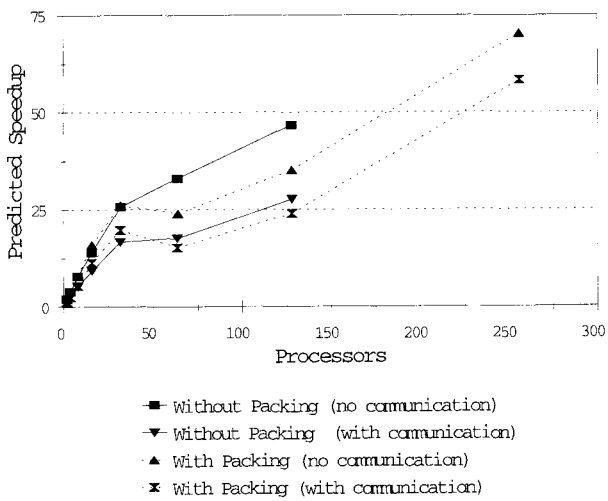

FIGURE 12 Performance predictions for the Westinghouse AP600 reactor model.

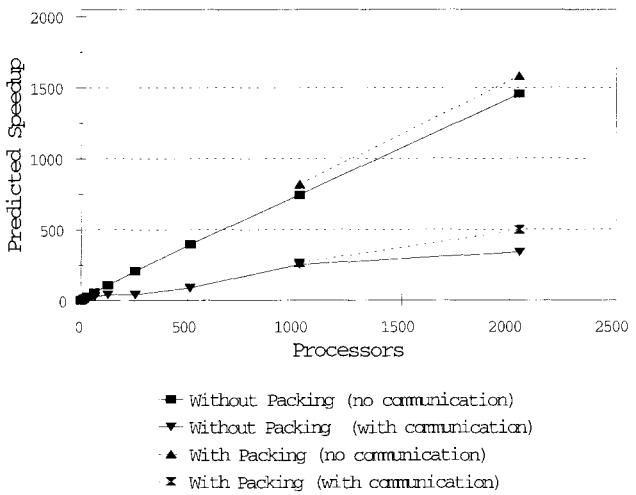

FIGURE 13 Performance predictions for flow through an elbow with vanes.

cessors because packing of the 1-dimensional meshes favors a 1-dimensional distribution, but a 3-dimensional distribution is favored for reactor core consideration. This suggests an extension to the algorithm that will be mentioned briefly in the next section.

5.2 Westinghouse AP600 Reactor Model

The Westinghouse AP600 reactor model nodalization was developed by Jim Lime at Los Alamos National Laboratories with support from the Nuclear Regulatory Commission [24]. This model has 173 hydro components (1,060 3-dimensional cells and 865 1-dimensional cells) and 47 heat structures.

The results for the Westinghouse AP600 reactor model are summarized in Figure 12. The results for the AP600 simulation are somewhat disappointing, but in hindsight understandable. The packing algorithm does a 2-dimensional packing if there are no meshes to pack of higher dimensionality. For this reason, when 64 or 128 processors are used, there is a direct conflict between the processor layout that is preferred for the reactor core and the one that is preferred for the large packed mesh. Although the problem is present with the unpacked algorithm (as indicated by the limited speedups), it is amplified by packing. To lessen this effect, the packing algorithm should be extended to give substantial preference to 3-dimensional packings. Actually, substantial preference should be given to n-dimensional packings where n is the dimensionality of the highest dimensionality large mesh. If there are no large meshes, then the easiest packing should be done.

Even though disappointing, these results compare favorably to the Argonne results. On eight processors the speedup bound with communication is 5.3 without packing and 5.4 with packing, as compared with the best speedup of 3 achieved with the hand-parallelized code at Argonne.

5.3 Simulation of Material Flow in an Elbow with Vanes

The Elbow problem, shown in Figure 1, has five 3-dimensional meshes with a total of 275,356 3-dimensional cells. Note that in the remaining results of experiments, the "without packing" results use only the large mesh algorithm as all components are large. The results for flow through an elbow with two vanes are summarized in Figure 13. Note that while the processor utilization drops off as more processors are added, the speedup bound, $Dist_{sb}$, does continue to increase. There is a large increase in the total communication, from 128 processors to 256 processors due to the increase in the maximum amount of communication between a pair of processors. This increase comes about because the interface between adjacent processors in the third processor dimension is significantly larger for 256 processors than for 128 processors. This interface size has increased due to the change in mapping template dimensions to processor dimensions. The reverse situation occurs in the same problem when going from 512 processors to 1,024 processors.

Packing can improve the speedup bounds. However, in fluid/aerodynamic simulations, the improvements are not as impressive as were seen for the reactor

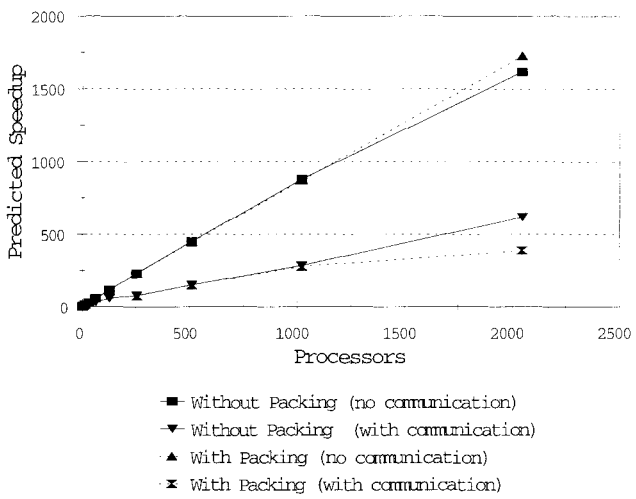

FIGURE 14 Performance predictions for Fuselage-Inlet-Nozzle simulation.

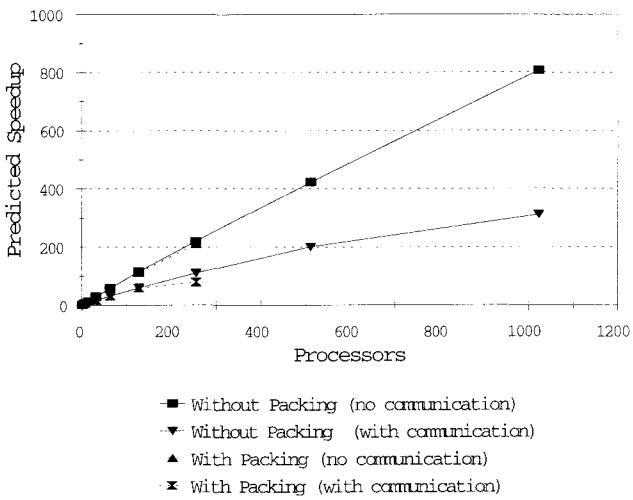

FIGURE 15 Performance predictions for full F15e Simulation.

simulations because there are no small meshes to put in the holes that are left after packing.

5.4 Simulation of Aerodynamics over Fuselage-Inlet-Nozzle of an F15e Aircraft

The Fuselage-Inlet-Nozzle aerodynamic simulation has 10 3-dimensional meshes with a total of 713,766 3-dimensional cells. The results of the large mesh algorithm experiments for this problem are summarized in Figure 14.

In the Fuselage-Inlet-Nozzle problem, there are too few processors for the packing algorithm to come into practical use. In fact, the load balance gets worse with the use of the packing algorithm until 2,048 processors are used. This is due to the fact that five meshes have been packed. Those meshes could not be evenly distributed individually. When they are packed into two larger ones, the larger ones are less evenly distributed. For the larger mesh, the number of elements per processor on busy processors is greater than the sum of the elements assigned from the smaller meshes.

5.5 Simulation of Aerodynamics over a Complete F15e Aircraft

The full F15e aerodynamic simulation, shown in Figure 2, has 32 3-dimensional meshes with a total of 1,269,845 3-dimensional cells. The results of the large mesh algorithm experiments for this problem are summarized in Figure 15. The total communication and the speedup bounds are monotonically increasing.

The packing algorithm can be expensive. This applies in particular to higher dimensional meshes with large sizes. This is the only situation where the size of a mesh changes the execution time of one of the automatic distribution algorithms. This long and size-dependent run-time of the automatic distribution algorithm is due directly to the difficulty of the problem being solved and the use of CPLEX to obtain a solution.

The run-time for the packing algorithm for 256 processors was to 9.44 hours. Therefore, the execution time for the packing algorithm was prohibitive on this problem for more processors. Also notice that more processors need to be used before packing becomes effective on this problem. This indicates that more algorithmic development is needed. If the prepacking heuristic can be extended to significantly reduce the number of meshes that are input to the local linear optimization procedure in cases such as this one where the meshes are essentially all different odd sizes, then it would be reasonable to apply this approach to these general aerodynamic simulation data sets.

5.6 Applicability of Packing Approach

Distribution algorithms for all combinations of different sizes of meshes have now been presented. The performance predictions for the packing algorithm make a discussion of the applicability of packing desirable. There are three factors that determine how well the packing approach will work for medium-sized meshes. The first factor is whether the meshes being packed are the same dimensionality as any large meshes in the problem. If the medium-sized meshes have the same dimensionality as the large meshes, then the packed mesh(es) will also. This implies that there will not be a conflict between the dimensionalities of the preferred distributions for the meshes. In future work, we will use the large mesh dimensionality to guide the packing algorithm.

The second factor in determining the effectiveness of the packing approach is how well the meshes pack. One measure of how well meshes pack is the number and size of the holes left in the bin once it is packed. Packing performs particularly well when subsets of the meshes to be packed are the same size in $n - 1$ (or n) dimensions. In this case, the prepacking algorithm packs the subsets of meshes with no wasted space. Thus, prepacking reduces the number of meshes to be packed and this in turn makes packing take less time.

The final factor in determining the effectiveness of packing is whether there are small meshes in the problem. In problems with small meshes, the small meshes are used to fill up the holes left during distribution of medium meshes. This factor can even help to improve poor load balance that is due to the other two factors.

6 CONCLUSIONS

The algorithms outlined here automatically find a distribution of data in ICRM problems given topology specifications. These experiments show that it is not necessary for users to determine distribution specifications by hand for these complex topology problems.

We are currently working on extension of these algorithms for automatic distribution. In particular, we are working on the improvement of the algorithms in two areas. The first area deals with medium mesh packing. The packing algorithm needs to be extended to give preference to the generation of n-dimensional packed meshes, where n is the dimensionality of the large meshes in the problem. The second area deals with small mesh distribution. The topology-based small mesh distribution algorithm needs to be combined with the use of the load-balance measures available after large mesh distribution. This merging of algorithms will promote the improved load balance that is achieved in the small mesh hole filling algorithm while allowing a reduction in communication based on the topology of the problem. This should increase the number of processors that we can make practical use of for these problems. This approach will also extend the set of applications for which we can automatically determine distributions.

ACKNOWLEDGMENTS

This research was supported by: Center for Research in Parallel Computation, a National Science Foundation Science and Technology Center, through Cooperative Agreement No. CCR-9120008, NSF/NASA Agreement No. ASC-9213821, and ONR Agreement No. N00014-93-1-0158. The processes described herein are covered by the pending Patent No. 08/423,483.

REFERENCES

[1] D. Heermann and A. Burkitt, *Parallel Algorithms in Computational Science*. New York: Springer-Verlag, 1991.

[2] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu, "Fortran D language specification," Department of Computer Science, Rice University, Houston, TX, Tech. Rep. TR90-141, Dec. 1990.

[3] C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele, Jr., and M. E. Zosel, *The High Performance Fortran Handbook*. Cambridge, MA: MIT Press, 1994.

[4] J. Shaw and N. Weatherill, "Automatic topology generation for multiblock grids," *Appl. Math. Computation*, vol. 52, pp. 355–388, 1992.

[5] C. Chase, K. Crowley, J. Saltz, and A. Reeves, "Compiler and runtime support for irregularly coupled regular meshes," *Proceedings of the 1992 International Conference on Supercomputing*. New York: ACM, pp. 438–446, July 1992.

[6] N. Mansour and G. C. Fox, "Parallel physical optimization algorithms for allocating data to multicomputer nodes," *J. Supercomput.*, vol. 8, pp. 53–80, 1994.

[7] A. Zaafrani and M. Ito, "Partitioning the global space for distributed memory systems," in *Proc. of Supercomputing '93*, pp. 327–336, 1993.

[8] C.-W. Ou and S. Ranka, "Parallel incremental graph partitioning," Northeast Parallel Architecture Center, Syracuse University, New York, Tech. Rep. SCCS 652, Aug. 1994.

[9] C.-W. Ou and S. Ranka, "Parallel remapping algorithms for adaptive problems," Northeast Parallel Architecture Center, Syracuse University, New York, Tech. Rep. SCCS 653, 1994.

[10] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *Siam J. Matrix Anal. Appl.*, vol. 11, pp. 430–452, July 1990.

[11] L. Dagnum, "Automatic partitioning of unstructured grids into connected components," *Proc. of Supercomputing '93*, pp. 94–101, 1993.

[12] C. Farhat and M. Lesoinne, "Mesh partitioning algorithms for the parallel solution of partial differential equations," *Appl. Numerical Math.*, vol. 12, pp. 443–457, May 1993.

[13] A. Tentner, R. Blomquist, T. Canfield, P. Garner, E. Gelbard, K. Gross, M. Minkoff, and R. Valentin, "Advances in parallel computing for reactor analysis and safety," *Commun. ACM*, vol. 37, pp. 55–64, 1994.

[14] Y. Yamamoto, "Numerical simulation of hypersonic viscous flow for the design of H-II orbiting plane (HOPE)," *Comput. Methods Appl. Mech. Eng.*, vol. 89, pp. 59–72, 1990, August.

[15] L.-E. Eriksson, "Flow solution on a dual-block grid around an airplane," *Comput. Methods Appl. Mech. Eng.*, vol. 64, pp. 79–83, 1987.

[16] J. L. Steger and J. A. Benek, "On the use of composite grid schemes in computational aerodynamics," *Comput. Methods Appl. Mech. Eng.*, vol. 64, pp. 301–320, 1987.

[17] R. E. Smith and L.-E. Eriksson, "Algebraic grid generation," *Comput. Methods Appl. Mech. Eng.*, vol. 64, pp. 285–300, 1987.

[18] J. Steinbrenner, S. Karmen, and J. Chawner, "Generation of multiple block grids for arbitrary 3-d geometries, in *3-Dimensional Grid Generation for Complex Configurations*, H. Yoshihara, Ed. AGARDograph, 1988, p. 309.

[19] J. Shaw, J. Georgala, and N. Weatherill, "The construction of component adaptive grids for aerodynamic geometries," in *Numerical Grid Generation in Compu-*

tational Fluid Mechanics '88, S. Sengupta, J. Hauser, P. Eiseman, and J. Thompson, Eds. Pineridge Press, 1988, pp. 383–394.

[20] S. Allwright, "Techniques in multiblock domain decomposition and surface grid generation," in *Numerical Grid Generation in Computational Fluid Mechanics '88*, S. Sengupta, J. Hauser, P. Eiseman, and J. Thompson, Eds. Pineridge Press, 1988, pp. 559–568.

[21] J. F. Thompson, "A general 3-dimensional elliptic grid generation system on a composite block structure," *Comp. Methods Appl. Mech. Eng.*, vol. 64, pp. 377–411, 1987.

[22] M. S. Shephard and M. K. Georges, "Reliability of automatic 3d mesh generation," *Comp. Methods Appl. Mech. Eng.*, vol. 101, pp. 443–462, 1992.

[23] B. Boyack, H. Stumpf, and J. Lime, *TRAC User's Guide*, Los Alamos, NM: Los Alamos National Laboratory, 1985, p. 87545.

[24] J. Lime and B. Boyack, "TRAC large-break loss-of-coolant accident analysis for the AP600 design," presented at the Int. Topical Meeting on Advanced Reactors Safety, Pittsburgh, PA, 1994.

[25] L. Liebrock and K. Kennedy, "Automatic data distribution of small meshes in coupled grid applications," in *Concurrency: Practice and Experience, 1996*, 1996.

[26] M. W. Hall, S. Hiranandani, K. Kennedy, and C. Tseng, "Interprocedural compilation of Fortran D for MIMD distributed-memory machine," in *Proc. of Supercomputing '92*, pp. 522–534, 1992.

[27] L. Liebrock, "Using problem topology in parallelization," Center for Research in Parallel Computation, Rice University, Houston, TX, Tech. Rep. 94-477-S, Sept. 1994.

[28] L. Liebrock and K. Kennedy, "Parallelization of linearized application in Fortran D," in H. J. Siegel, Ed., *International Parallel Processing Symposium '94*, Washington, DC: IEEE, 1994, pp. 51–60.

[29] L. Liebrock and K. Kennedy, "Modeling parallel computation," Center for Research in Parallel Computation, Rice University, Houston, TX, Tech. Rep. 94-499, May 1994.

[30] L. Liebrock and K. Kennedy, "Automatic data distribution of large meshes in coupled grid applications," Rice University, Center for Research in Parallel Computation, Houston, TX, Tech. Rep. 94-395, Apr. 1994.

[31] CPLEX Optimization, Inc., *Using the CPLEX Callable Library, Version 3.0*, Incline Village, NV: CPLEX, 1994.

[32] J. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, NJ: Prentice-Hall, 1983.