DIME: Portable Software for Irregular Meshes for Parallel or Sequential Computers

Roy Williams
California Institute of Technology

Introduction

A large fraction of the problems that we wish to solve with a computer are continuum simulations of physical systems, where the interesting variable is not a finite collections of numbers but a function on a domain. For the purposes of the computation such a continuous spatial domain is given a structure, or mesh, to which field values may be attached and neighboring values compared to calculate derivatives of the field.

If the domain of interest has a simple shape such as a cylinder or cuboid, there may be a natural choice of mesh whose structure is very regular like that of a crystal, but when we come to more complex geometries such as the space surrounding an aircraft or inside turbomachinery, there are no regular structures that can adequately represent the domain. The only way to mesh such complex domains is with a unstructured mesh, such as that shown at the top of Figure 1. At the bottom is a plot of a solution to Laplace's equation on the domain.

Notice that the mesh is especially fine at the sharp corners of the boundary where the solution changes rapidly: a desirable feature for a mesh is the ability for it to adapt, so that when the solution begins to emerge, the mesh may be made finer where necessary.

Naturally we would like to run our time-consuming physical simulation with the most cost-effective general purpose computer, which we believe to be the MIMD architecture. In view of the difficulty of programming an irregular structure such as one of these meshes, and the special difficulty of doing so with an MIMD machine, I decided to write not just a program for a specialized application, but a *programming environment* for unstructured triangular meshes.

The resulting software (DIME: Distributed Irregular Mesh Environment[1]) is responsible for the mesh structure, and a separate application code runs a particular type of simulation on the mesh. DIME keeps track of the mesh structure, allowing mesh creation, reading and writing meshes to disk, and graphics; also adaptive refinement and certain topological changes to the mesh; it hides the parallelism from the application code, and splits the mesh among the processors in an efficient way.

The application code is responsible for attaching data to the elements and nodes of the mesh, manipulating and computing with these data and the data from its mesh-neighborhood.

DIME is designed to be portable, not only between different MIMD parallel machines, but it also runs on any Unix machine, treating this as a parallel machine with just one processor. This ability to run on a sequential machine is due to DIME's use of the Cubix server, part of the Express programming environment[2].

Applications and Extensions

The most efficient speed for aircraft flight is just below the speed of sound: the transonic regime. Simulations of flight at these speeds are a natural candidate for a DIME application. In addition to the complex geometries of airfoils and turbines for which these simulations are required, the flow tends to

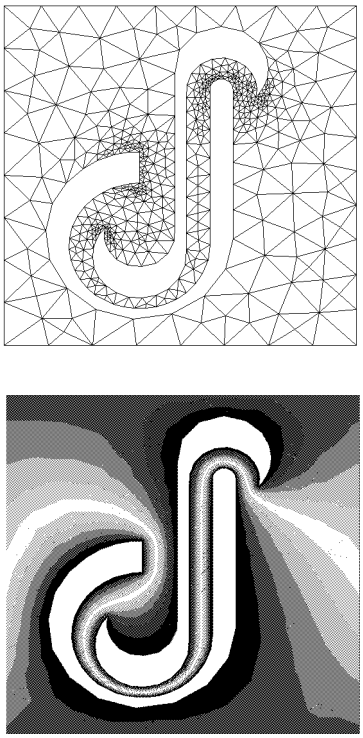

Figure 1: Mesh and solution of Laplace equation

develop singular regions or shocks in places that cannot be predicted in advance; the adaptive refinement capability of a DIME mesh allows the mesh to be fine and detail resolved near shocks while keeping the regions of smooth flow coarsely meshed for economy[3].

The current version of DIME is only able to mesh two-dimensional manifolds. The manifold may, however, be embedded in a higher-dimensional space. In collaboration with the Biology Division at Caltech we have simulated the electrosensory system of the weakly electric fish *Apteronotus leptorhynchus*. The simulation involves creating a mesh covering the skin of the fish, and using the Boundary Element Method to calculate field strengths in the three-dimensional space surrounding the fish[4].

In the same vein of embedding the mesh in higher dimensions, we have simulated a bosonic string of high-energy physics, embedding the mesh in up to 26 spatial dimensions. The problem here is to integrate over not only all positions of the mesh nodes, but also over all *triangulations* of the mesh[5].

The information available to a DIME application is certain data stored in the elements and nodes of the mesh. When doing Finite Element calculations, one would like a somewhat higher level of abstraction, which is to refer to functions defined on a domain, with certain smoothness constraints and boundary conditions. We have made a further software layer on top of DIME to facilitate this: DIMEFEM. With this we may add, multiply, differentiate and integrate functions defined in terms of the Lagrangian Finite Element family, and define linear, bilinear and nonlinear operators acting on these functions. When a bilinear operator is defined, a variational principle may be solved by conjugate-gradient methods. The preconditioner for the CG method may in itself involve solving a variational principle. The DIMEFEM package has been applied to a sophisticated incompressible flow algorithm[6].

The Components of DIME

Figure 2 shows the structure of a DIME application. The shaded parts represent the contribution from the user, being a definition of a domain which is to be meshed, a definition of the data to be maintained at each element, node and boundary edge of the mesh, and a set of functions that manipulate this data. The user may also supply or create a script file for running the code in batch mode.

The first input is the definition of a domain to be meshed. A file may be made using the elementary CAD program `curvetool`, which allows straight lines, arcs and cubic splines to be manipulated interactively to define a domain.

Before sending a domain to a DIME application, it must be predigested to some extent, with the help of a human. The user must produce a coarse mesh that defines the topology of the domain to the machine. This is done with the program `meshtool`, which allows the user to create nodes and connect them up to form a triangulation.

The user writes a program for the mesh, and this program is loaded into each processor of the parallel machine. When the DIME function `readmesh()` is called, (or "Readmesh" clicked on the menu), the mesh created by `meshtool` is read into a single processor, and then the function `balance_orb()` may be called (or "Balance" clicked on the menu) to split the mesh into domains, one domain for each processor.

The user may also call the function `writemesh()` (or click "Writemesh" in the menu), which causes the parallel mesh to be written to disk. If that mesh is subsequently read in, it is

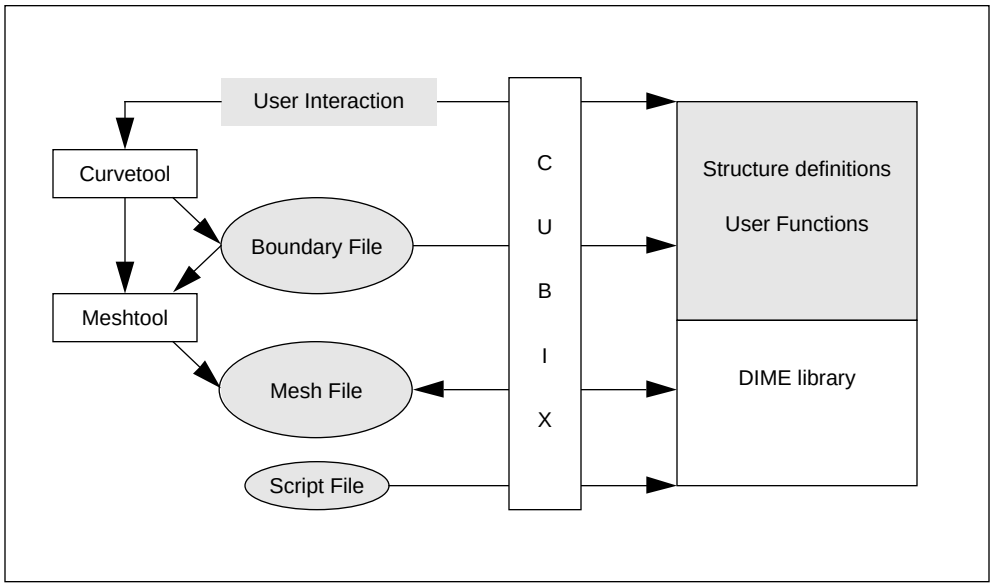

Figure 2: Major components of DIME

read in its domain-decomposed form, with different pieces assigned to different processors.

In the Figure, the Cubix server is not mandatory, but only needed if the DIME application is to run in parallel. The application ***also runs on a sequential machine***, which is considered to be a one-processor parallel machine.

Domain Definition

Figure 3 shows an example of a DIME boundary structure. The filled blobs are *Points*, with *Curves* connecting the points. Each Curve may consist of a set of curve segments, shown in the Figure separated by open circles. The curve segments may be straight lines, arcs of circles, or Bezier cubic sections. The program `curvetool` is for the interactive production of boundary files. When the domain is satisfactory, it should be meshed using `meshtool`.

The program `meshtool` is for taking a boundary definition and creating a triangulation of certain regions of it. `Meshtool` adds nodes to an existing triangulation using the Delaunay triangulation[7]. A new node may be added anywhere except at the position of an existing node. Figure 4 illustrates how the Delaunay triangulation (thick gray lines) is derived from the Voronoi tesselation (thin black lines.)

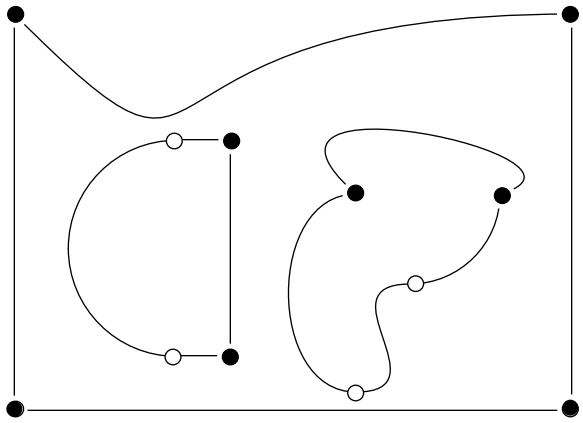

Figure 3: Boundary structure

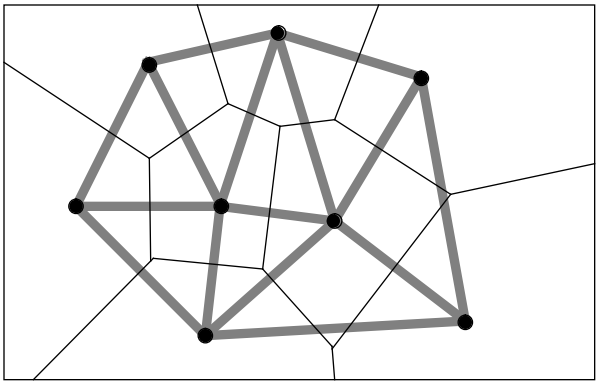

Figure 4: Voronoi tesselation and resulting Delaunay triangulation

Each node (shown by a blob in the Figure) has a “territory”, or Voronoi polygon, which is the part of the plane closer to the node than to any other node. The divisions between these territories are shown as thin lines in the Figure, and are the perpendicular bisectors of the lines between the nodes. This procedure tesselates the plane into a set of disjoint polygons and is called the Voronoi tesselation. Joining nodes whose Voronoi polygons have a common border creates a triangulation of the nodes known as the Delaunay triangulation. This triangulation has some desirable properties, such as the diagonal dominance of a Finite Element stiffness matrix derived from the mesh[8].

Mesh Structure

In Figure 5 is shown a triangular mesh covering a rectangle, and in Figure 6 the logical structure of that mesh split among four processors. The logical mesh shows the elements as shaded triangles and nodes as blobs. Each element is connected to exactly three nodes, and each node is connected to one or more elements. If a node is at a boundary, it has a boundary structure attached, together with a pointer to the next node clockwise around the boundary.

Each node, element and boundary structure has user data attached to it, which is automatically transferred to another processor if load-balancing causes the node or element to be moved to another processor. DIME knows only the size of the user data structures. Thus these structures may not

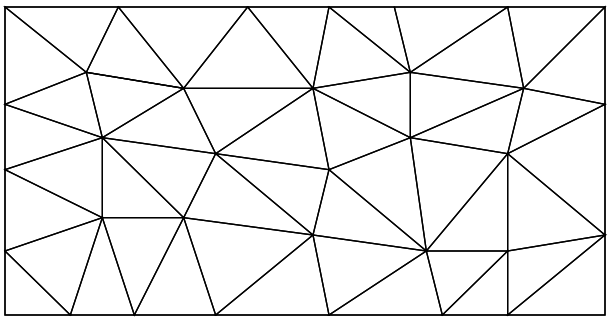

Figure 5: Mesh covering a rectangle

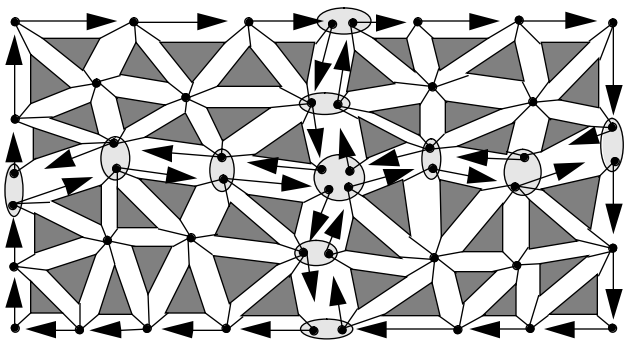

Figure 6: Logical structure of a mesh split among 4 processors

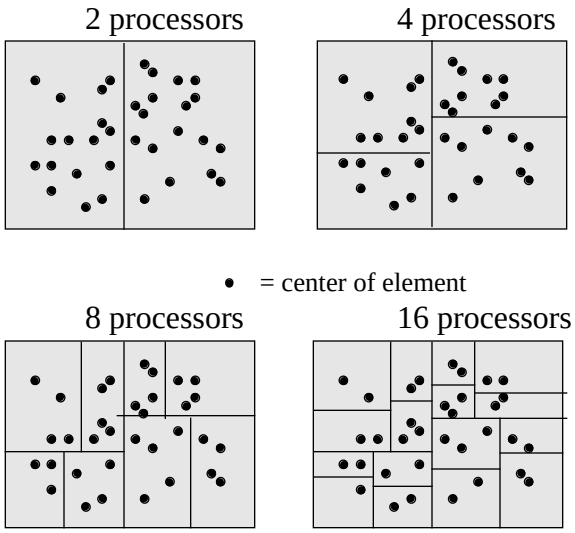

Figure 7: Orthogonal Recursive Bisection

contain pointers, since when those data are moved to another processor the pointers will be meaningless.

The shaded ovals in Figure 6 are *Physical Nodes*, each of which consists of one or more *Logical Nodes*. Each logical node has a set of *aliases*, which are the other logical nodes belonging to the same physical node. The physical node is a conceptual object, and is unaffected by parallelism; the logical node is a copy of the data in the physical node, so that each processor which owns a part of that physical node may access the data as if it had the whole node.

DIME is meant to make distributed processing of an unstructured mesh almost as easy as sequential programming. However, there is a remaining ‘‘kernel of parallelism’’ that the user must bear in mind. Suppose each node of the mesh gathers data from its local environment (i.e. the neighboring elements); if that node is split among several processors, it will only gather the data from those elements which lie in the same processor and consequently each node will only have part of the result. We need to combine the partial results from the logical nodes and return the combined result to each. This facility is provided by a macro in DIME called `NODE_COMBINE`, which is called each time the node data is changed according to its local environment.

Refinement

The Delaunay triangulation used by `meshtool` would be an ideal way to refine the working mesh, as well as making a coarse mesh for initial download. Unfortunately, adding a new node to an existing Delaunay triangulation may have global consequences; it is not possible to predict in advance how much of the current mesh should be replaced to accommodate the new node. Doing this in parallel requires an enormous amount of communication to make sure that the processors do not tread on each others toes[9].

DIME uses the algorithm of Rivara[10] for refinement of the mesh, which is well suited to loosely synchronous parallel operation, but results in a triangulation which is not a Delaunay triangulation, and thus lacks some desirable properties. The process of *topological relaxation* changes the connectivity of the mesh to make it a Delaunay triangulation.

It is usually desirable to avoid triangles in the mesh which have particularly acute angles, and topological relaxation will reduce this tendency. Another method to do this is by moving the nodes toward the average position of their neighboring nodes; a physical analogy would be to think of the edges of the mesh as damped springs and allowing the nodes to move under the action of the springs.

Load Balancing

When DIME operates in parallel, the mesh should be distributed amongst the processors of the machine so that each processor has about the same amount of mesh to deal with, and so that the communication time is as small as possible. There are several ways to do this, such as with a computational neural net, by simulated annealing, or even interactively.

DIME uses a strategy known as Orthogonal Recursive Bisection[11], which has the advantages of being robust, simple, and deterministic, though sometimes the resulting communication pattern may be less than optimal. The method is illustrated in Figure 7. Each blob represents the center of an element, and the vertical and horizontal lines represent processor divisions. First a median vertical line is found which splits the set of elements into two sets of approximately equal numbers, then (with 2-way parallelism) two horizontal medians which split the halves into four approximately equal quarters, then (with 4-way parallelism) four vertical medians, and so on.

References

1 R. D. Williams, DIME, available by anonymous ftp from delilah.ccsf.caltech.edu.

2 DIME runs in parallel with the Express programming environment, available from ParaSoft Corp., Pasadena, CA, (818) 792 9941.

3 R.D.Williams, *Supersonic Flow in Parallel with an Unstructured Mesh*, Concurrency, Practice & Experience, **1**, 51 (1989)

4 R.D.Williams, B. Rasnow and C. Assad, *Hypercube Simulation of Electric Fish Poetntials*, Proc. DMCC5 (Distributed Memory Computing Conference), Charleston, SC, 1990.

5 C. F. Baillie, D. A. Johnston and R.D.Williams, *Crumpling in Dynamically Triangulated Random*

Surfaces with Extrinsic Curvature, Nucl. Phys. **B335** (1990) 469

6 R.D.Williams, *Distributed Irregular Finite Elements*, in *Numerical Methods in Laminar and Turbulent Flow*, Pineridge Press, Swansea, UK, 1989, ed C. Taylor.

7 A. Bowyer, *Computing Dirichlet Tesselations*, Comp. J. **24** (1981) 162.

8 D. M. Young, *Iterative Solution of Large Linear Systems*, Academic Press, New York, 1971.

9 R. D. Williams and E. W. Felten, *Distributed Processing of an Irregular Tetrahedral Mesh*, California Institute of Technology, Concurrent Computation Report C3P 793.

10 M-C. Rivara, *Design and Data Structure of Fully Adaptive Multigrid, Finite-Element Software*, ACM Trans. in Math. Software, **10** (1984) 242.

11 R. D. Williams, *Performance of Dynamic Load-Balancing Algorithms for Unstructured Mesh Calculations*, Concurrency, Practice and Experience, (to be published).