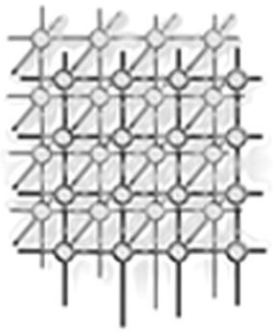
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2003; **15**:431–458 (DOI: 10.1002/cpe.662)



CartaBlanca—a pure-Java, component-based systems simulation tool for coupled nonlinear physics on unstructured grids—an update

W. B. VanderHeyden*,†, E. D. Dendy and N. T. Padial-Collins

Los Alamos National Laboratory, Theoretical Division and Los Alamos Computer Science Institute, Los Alamos, NM 87545, U.S.A.

SUMMARY

This paper describes a component-based nonlinear physical system simulation prototyping package written entirely in Java using object-oriented design. The package provides scientists and engineers with a 'developer-friendly' software environment for large-scale computational algorithm and physical model development. The software design centers on the Jacobian-free Newton–Krylov solution method surrounding a finite-volume treatment of conservation equations. This enables a clean component-like implementation. We first provide motivation for the development of the software and then discuss software structure. The discussion includes a description of the use of Java's built-in thread facility that enables parallel, shared-memory computations on a wide variety of unstructured grids with triangular, quadrilateral, tetrahedral and hexahedral elements. We also discuss the use of Java's inheritance mechanism in the construction of a hierarchy of physics systems objects and linear and nonlinear solver objects that simplify development and foster software re-use. We provide a brief review of the Jacobian-free Newton–Krylov nonlinear system solution method and discuss how it fits into our design. Following this, we show results from example calculations and then discuss plans including the extension of the software to distributed-memory computer systems. Copyright © 2003 John Wiley & Sons, Ltd.

KEY WORDS: Java; high performance computing; object oriented design; finite volume; nonlinear solver; development environment

1. INTRODUCTION

Specialized simulation software for nonlinear physical systems is one of the central research products from many of the programs here at Los Alamos National Laboratory (LANL) and at other similar

*Correspondence to: W. B. VanderHeyden, Los Alamos National Laboratory, Theoretical Division and Los Alamos Computer Science Institute, Los Alamos, NM 87545, U.S.A.
†E-mail: wbv@lanl.gov

Received 31 August 2001
Revised 21 January 2002

institutions. Such systems are often three-dimensional and are solved on unstructured computational grids. Thus, simulation software for these systems can be quite complex. Very often, simulation projects involve the modification of existing software to produce new capabilities. Furthermore, program goals change frequently as funding priorities evolve or as research developments lead to new branches of investigation. As such, application development is often the bottleneck on these projects. There is, therefore, substantial incentive for software and software environments that are 'developer-friendly'—easy for scientific and engineering software developers to modify and extend. Fortunately, two technology trends address this need in a coordinated fashion.

First, object-oriented and component-based software has made an enormous impact in the commercial software arena for telecommunications and e-business. Enhancement of software developer productivity has come from a variety of sources including new languages such as Java, which support object orientation and component software and a host of software developer productivity tools including graphical design applications, graphical debuggers, etc., that work with these languages. At the same time, significant advances have occurred in nonlinear systems solution methods. In particular, Jacobian-free Newton–Krylov (JFNK) [1] methods have been employed and extended to provide robust and flexible solution methods for a wide variety of coupled nonlinear physics simulation capabilities. In addition, the combination of JFNK and the finite-volume technique [2] fits well into an object-oriented or component-based software environment.

Thus, the goal of the CartaBlanca project is to produce a modern flexible software environment for prototyping physical models, discretization schemes and solution methods for nonlinear physics problems on unstructured grids. CartaBlanca employs an object-oriented, component-based design using the Java programming language. CartaBlanca uses unstructured grids composed of either triangular or quadrilateral mesh elements in two dimensions, or tetrahedral or hexahedral mesh elements in three dimensions. CartaBlanca employs the finite-volume method [2] to provide for flexibility with regard to both meshes and physical effects. Finally, CartaBlanca uses Java's built-in thread facility for shared-memory parallelization.

1.1. Status

We currently have a great deal of infrastructure for CartaBlanca in place. Our updated list of features since first reporting on CartaBlanca [3] includes:

- the acceptance of unstructured grids in two and three dimensions with triangular, tetrahedral, quadrilateral and hexahedral elements;
- the acceptance of Metis [4] generated mesh partition files; it computes in parallel on Metis subdomains;
- a Graphical User Interface (GUI) for problem specification;
- the use of abstract classes for state, physics and solvers objects; it imposes a uniform component-like interface for developers;
- pre-conditioned Conjugate gradient and GMRES Krylov linear solvers interact seamlessly with physics objects;
- a nonlinear quasi-Newton solver wraps around linear Krylov solvers to provide a JFNK solver;
- physics objects are available for high accuracy scalar advection, interface tracking, heat transfer and multiphase flow;

- automatic generation of Tecplot (from Amtec Engineering) graphics files for arbitrary physical systems;
- an embedded software-testing facility based on the JUnit framework [5];
- direct remote access to CVS revision control server is enabled for efficient team code development.

1.2. Java performance

A recent article by Schatzman and Donehower [6] provides a useful discussion of the potential pitfalls involved and tips on how to program in Java to try to achieve performance comparable to that obtainable using C, C++ and Fortran. Similar information is also available in [7]. More encouraging news regarding Java performance has been reported by Boisvert *et al.* [8]. These authors provide an up-to-date review, which shows that Java performance has improved significantly over the past several versions of the Java Virtual Machine (JVM). Remarkably, they show that, for some cases, Java is slightly faster than C when tested back-to-back on a suite of scientific computing benchmark applications called Scimark from the US National Institute of Standards and Technology. In addition, Reinholtz [9] writes that not only is it possible for Java to have better general performance than C++, but also that it is likely that this will actually occur. He argues that Java performance can exceed that of C++ because dynamic compilation gives the Java compiler access to run-time information not available to a C++ compiler. Furthermore, the rapidly growing market for embedded systems will drive such performance improvements to extend battery life. Considering the above information and the important benefits from Java's strong typing, clean design, object-orientation, it is our opinion that Java is now a serious alternative language for scientific and engineering computing applications.

1.3. Related efforts

CartaBlanca builds on some important existing software programs here at LANL. Here is a brief description of these programs. CFDLIB [10], a Fortran 77 program, is an outgrowth of the Caveat program [11], which provides a flexible finite-volume multiphase flow simulation capability. The Telluride program [12] provides a multi-material simulation capability with interface tracking on unstructured grids. Telluride makes extensive use of the Fortran 90 module concept. The CHAD program [13] is a flexible node-based finite-volume simulation program for flow simulation on unstructured grids. CHAD uses Fortran 90 and accommodates hybrid grids using an edge-based connectivity data structure [14]. Finally, Kokopelli is a C++ program for interfacial and polymer flow problems. The authors have had extensive experience with these programs. The design and implementation of CartaBlanca builds on the lessons learned from these experiences.

It is also worthwhile to note two relevant examples from outside our laboratory. First, Hauser *et al.* [15] have produced an object-oriented, pure-Java simulation code for aerospace applications. They employ a multi-block structured grid computation scheme and use Java's thread and RMI facilities for parallelization and to enable remote interaction between a GUI and the numerical application.

Another effort worth noting here is that of Hatakeyama *et al.* [16], who describe an object-oriented paradigm for flow simulation software in which the object-oriented concept is used at the computational node level to produce flexible abstractions. They demonstrate their concepts with a C++, structured grid

simulation of a wind tunnel with a test object and show that they can easily insert and extract arbitrary-shaped flow obstacles.

1.4. Outline

In Section 2 we provide an overview of the finite-volume method for discretization of conservation equations. In Section 3 we describe the major features of the JFNK solution method. We then proceed in Section 4 to give an overview of the CartaBlanca software packages. We follow this in Section 5 with some results from example calculations. Finally, in Section 6 we provide a discussion of conclusions and plans.

2. THE FINITE-VOLUME METHOD

CartaBlanca is based on the finite-volume method [2] for conservation equations. CartaBlanca adopts the node-based version of this scheme with edge-based connectivity [13,14]. We provide here a very simplified outline of the method. For an arbitrary control volume V with bounding surface A the generic conservation statement is of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V q\,\mathrm{d}V + \oint_A \mathbf{f}\cdot\mathbf{n}\,\mathrm{d}S + \int_V s\,\mathrm{d}V = 0 \tag{1}$$

where q is the density of some conserved quantity such as mass, momentum or energy, $\mathbf{f}$ is the local flux of this conserved quantity due to a variety of mechanisms, $\mathbf{n}$ is an outward normal vector defined on the surface of the control volume, and s is a generalized source density. The first and third integrals in Equation (1) are over the entire space of the control volume; the second integral is over the surface of the control volume. The derivative on the first integral quantity in Equation (1) is with respect to time. For numerical computations, Equation (1) is discretized in time and in space on a computational grid. On such a grid, conservation nodes are connected by edges as shown in Figure 1.

Each node is associated with a polyhedral control volume, V_i, as depicted in Figure 1. For each node, the averaged value of the conserved density is defined as

$$q_i \equiv \frac{1}{V_i}\int_{V_i} q\,\mathrm{d}V \tag{2}$$

The quantities q_i are, typically, the state variables for the numerical simulation. Similarly, the average source over each control volume is

$$s_i \equiv \frac{1}{V_i}\int_{V_i} s\,\mathrm{d}V \tag{3}$$

Let $\mathbf{f}_e$ be the average flux on the control volume face associated with edge e. Then, if we integrate Equation (1) over a time step, Δt, using, for example, a first-order difference approximation for the time derivative, we obtain the discretized form of the conservation equation

$$q_i^{n+1}V_i^{n+1} - q_i^n V_i^n + \Delta t\left\{\sum_{\text{edges}} \mathbf{f}_e\cdot\mathbf{n}_e A_e + s_i V_i\right\} = 0 \tag{4}$$

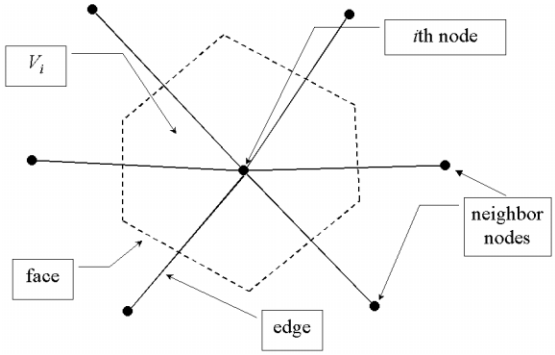


Figure 1. Control volume for the ith node.

where the superscripts n and $n+1$ denote the present and future time levels, respectively. Of course, the fluxes and source terms are generally functions of space, time and the state variables, q_i. Thus, the set of discretized conservation equations for all nodes and all types of conservation quantities forms a nonlinear algebraic system. The physics for a given application lies in the definition of the fluxes and sources in Equation (4). The aim of CartaBlanca is to provide scientists and engineers with a friendly environment using object-oriented Java for the implementation of component-like physics and solver objects for the solution of the corresponding coupled nonlinear conservation equations.

3. THE JFNK METHOD

We may write the set of conservation equations in the compact, abstract form

$$F_i(q^{n+1}) = 0 \tag{5}$$

where F_i denotes the left-hand side of Equation (4) and q^{n+1} denotes the entire set of state variables at the advanced time. The quantity F_i is called the residual function. The system represented by Equation (5) is, in general, nonlinear. We employ the JFNK method [1,17] in CartaBlanca to solve these systems. We provide here a brief outline in order to motivate our discussion of the software design. Newton's method for a nonlinear system begins with an initial guess of the solution, $q_j^{n+1(0)}$, where the superscript in parentheses denotes the iterate level. This is, typically, the solution from time level n. Newton's method then proceeds through a series of iterations involving the solution of a sequence of linear systems

$$J_{ij}(q^{n+1(k)})\delta_j^k = -F_i(q^{n+1(k)}) \tag{6}$$

along with the update

$$q^{n+1(k+1)} = q^{n+1(k)} + \delta^k \tag{7}$$

where there is an implied summation in Equation (6) on the repeated index, j. The goal, of course, is to proceed until we find the solution to Equation (5). The matrix quantity, J_{ij}, is the Jacobian matrix

defined as

$$J_{ij}(q) = \frac{\partial F_i(q)}{\partial q_j} \tag{8}$$

Explicit formation of the Jacobian matrix is typically a very expensive computation. Fortunately, the JFNK method takes advantage of the fact that Krylov linear solution methods require only the evaluation of matrix–vector products, Jv (where v is a Krylov vector), and not the matrix J by itself [1]. Furthermore, matrix–vector products can be approximated numerically using a directional difference formula,

$$Jv \approx \frac{F(q + \varepsilon v) - F(q)}{\varepsilon} \tag{9}$$

where ε is some small scalar perturbation parameter [1]. This approximation allows us to structure CartaBlanca in such a way that the physics developer can focus on providing residual functions inside physics objects or components. Using the abstraction embodied in Equation (5) we have genericized the rest of the infrastructure for solving and processing physics problems so that developers can work simultaneously on a variety of different problems using the same software.

4. SOFTWARE

CartaBlanca is composed, at present, of 12 separate packages. Each contains classes that perform distinct functions. We have tried to design these classes to serve as software components that can be interchanged in a 'plug and play' mode by developers. We have also tried to write the utility classes in such a way that physics and solver class developers need not concern themselves with the implementation of the parallel features of the software.

In the following, we describe each of these packages and the classes they contain. We also describe the interactions and associations between the classes in the different packages. We choose to start the discussion with the mesh and input packages. These are low-level packages; they are used by many but make sparing use of other packages. We then work our way up through the remaining packages of increasing complexity until we finally describe the main package, which contains the main methods. Before proceeding to the discussion of the CartaBlanca software packages, we start by commenting on our general design approach and on our software engineering methods.

4.1. Approach

Our approach to the design of CartaBlanca includes the following general guiding principles. First, we have endeavored to make use of object-orientation at the highest levels from a physical point of view. Thus, our objects are things that exist over entire sections of the computational domain or grid, rather than at individual nodes. This choice was made based on the idea that this would yield higher numerical performance by avoiding excessive overhead at the node level. This choice also provides a smoother transition into object-oriented programming for developers more familiar with procedural scientific legacy codes. Nevertheless, this approach has allowed us to make substantial use of Java and its object-oriented features.

Another principle we have employed is to make the top levels of the program as generic as possible so that the developer can plug physics into the appropriate program locations and then have the rest

of the program able to immediately interact. This was accomplished, in part, by the use of abstract classes, which provide general functionality and interfaces for things such as physics and solver objects. Thus, these objects are like components.

4.2. Software engineering

Our approach to team programming follows the lightweight processes advocated in the recent article by Fowler [18]. Iterative programming and component development has, for example, been very useful. The use of team coding has also proved helpful.

In order to foster the team software approach, we have incorporated the JUnit [5] testing facility into CartaBlanca. This has been useful in that any developer can perform tests easily on their local computing platform to make sure their modifications have not corrupted the software. This is in contrast to a situation in which software testing is performed using specialized software available only on a certain computing platform.

We have found it very helpful to use a common integrated development environment (IDE) for our software development. We are currently using JBuilder 4.0 Professional by Borland Technologies. JBuilder gives us an identical programming environment on our Windows NT and Solaris workstations. JBuilder is also available for LINUX operating systems. The JBuilder environment, conveniently, recognizes the JavaDoc @todo functionality. We use this feature as a simple issues tracking mechanism.

In addition to the JBuilder IDE, we use the GNU CVS revision control software for our software repository. We run CVS as a 'pserver' on one of our Solaris workstations. Thus, we can check pieces of software in and out over the network directly. We currently run simple implicit heat transfer, scalar advection and multiphase flow problems (discussed in Section 5) as test problems before committing software modifications to our CVS repository.

4.3. I/O package

CartaBlanca reads mesh files (see Section 4.6) and writes graphics files (see Section 4.11). In the initial stages of the project, an I/O package was imported from Chapman [19] for these operations [3]. This package contained classes with methods that enabled the developer to write C-language-syntax file print and read statements. We have since abandoned this package in favor of using Java's very effective Reader and Writer classes in the java.io package [20]. We make use of the non-standard ExponentialFormat class provided in [20] for writing doubles to text files. At present, this is the only class contained in the I/O package.

4.4. Input package

The input package contains the basic input facilities for problem specification. The user specifies parameters such as solver tolerances, physical properties and boundary conditions using a GUI written in Java's Swing library [21]. Problem data are written to a 'ProblemSpecifier' class object. The 'ProblemSpecifier' object contains all input information from the GUI. It can be queried as needed for information in the rest of the program. The 'ProblemSpecifier' object is serializable. This feature is

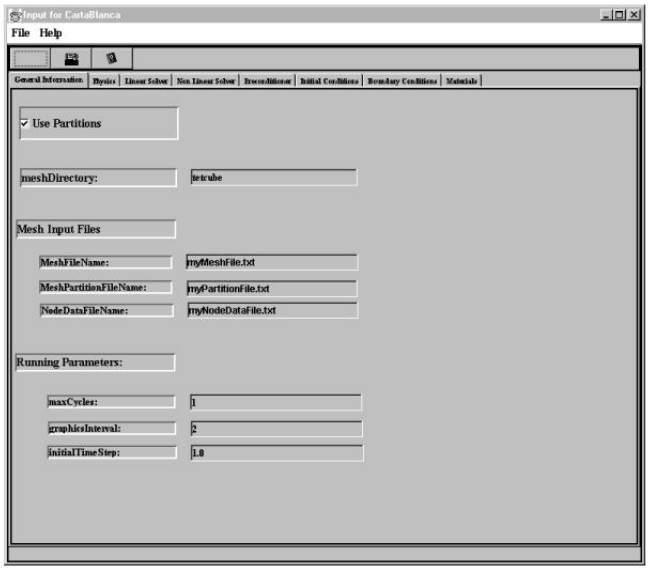


Figure 2. A snapshot of the GUI. Tabs enable the user to access the various categories of input.

used to save ProblemSpecifier settings to disk. This eliminates the need for any text-based input files, other than the mesh files (see Section 4.6).

The GUI is contained in three classes named 'TabbedInputClass', 'TabbedInputFrame' and 'TabbedInputFrame_AboutBox.' The GUI covers several categories of input separated into several tabbed input frames. The input categories are General Information, Physics, Linear Solver, Nonlinear Solver, Pre-conditioner, Initial Conditions, Boundary Conditions and Materials. A snapshot of the GUI interface is shown in Figure 2.

The user can click on the various tabs along the top of the GUI to access the different categories of input. As the user types in new information into the fields of the GUI, the information is written to the ProblemSpecifier object. When the user exits the GUI, the ProblemSpecifier is output to disk as a serialized object for future use and the rest of the program then begins to execute based on the information in the ProblemSpecifier object.

4.5. Communications package

The communications package contains classes of objects that provide functionality for inter-partition communication and for global mesh operations. The class CyclicBarrier provides a simple barrier that objects may invoke to synchronize calculations. The implementation was modeled on the barrier class provided in of Oaks and Wong [22, ch. 5]. The CyclicBarrier is used, for example, in discrete

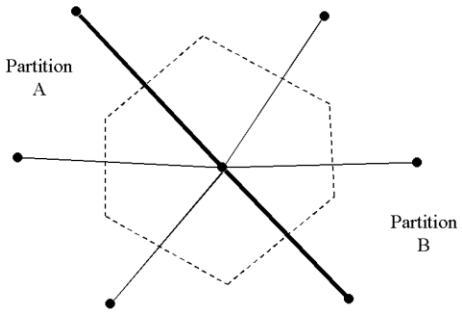


Figure 3. Partitioning in CartaBlanca. Meshes must be partitioned along node connections.

operations such as divergence field computations in which communication of flux quantities among mesh partitions are required.

The Reduction class in the communications package provides for the computation of global quantities across the entire mesh such as a global maximum or a global sum. Global sums are required, for example, for mesh-wide dot products of vectors in the various Krylov solvers. The Reduction class accomplishes this by using static class variables for sums and extrema.

4.6. Mesh package

The mesh package contains several classes that describe mesh elements, edges, interior boundary nodes, and partition and global meshes. These classes are built from mesh information read from mesh files. CartaBlanca requires three types of mesh information file, which follow the format used by the Metis mesh-partitioning program [4]. The three files contain the mesh connectivity, the node coordinates and the partitioning of the mesh elements; see the Metis manual [4] for a description of these files.

CartaBlanca requires mesh partitioning to be done in such a way that elements and not nodes are partitioned. Referring to Figure 3, the mesh partitioning for CartaBlanca must be done along node–edge connections. In Figure 3, the heavier edge connections denote the boundary between partitions A and B. To implement this mode of partitioning in CartaBlanca, nodes on the partition boundaries are duplicated. In the example in Figure 3, the three nodes along the partition boundary would be present in each partition as duplicates.

Figure 4 shows an example of an element-partitioned mesh for CartaBlanca. The mesh partitioning shown was performed using the Metis program and the Metis output was then fed to CartaBlanca for computations. The actual plot was generated using the Tecplot program which operates on graphics output files from CartaBlanca (see Section 4.11). A further example mesh is shown in Figure 5 for the case of a three-dimensional tetrahedral mesh.

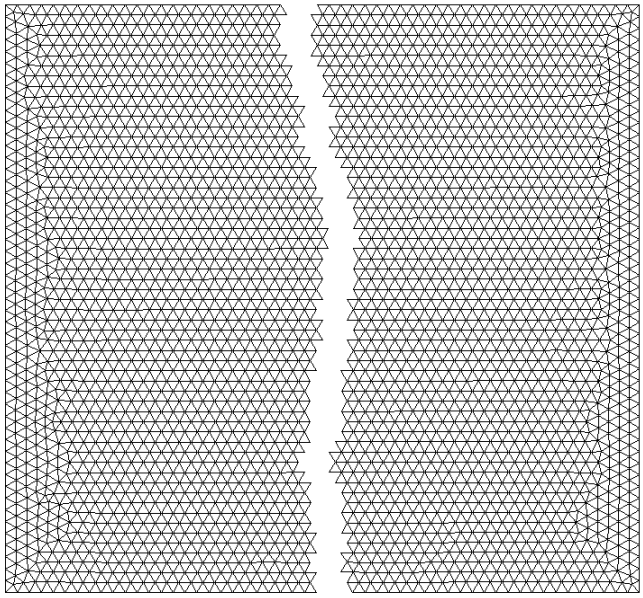


Figure 4. A two-dimensional 3000 node partitioned mesh. Partitions were generated using Metis.

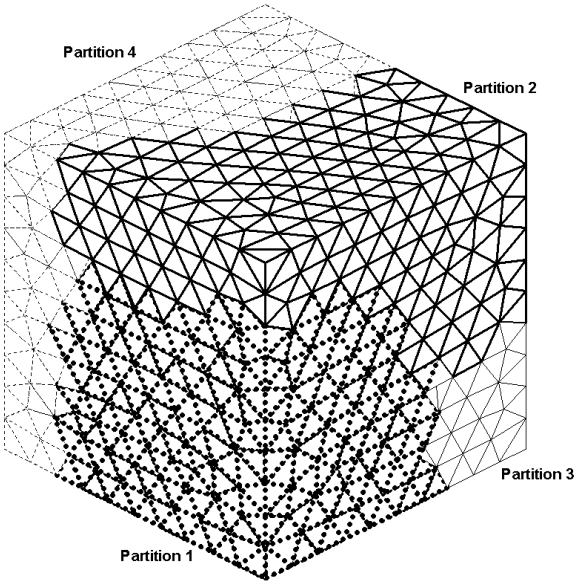


Figure 5. A three-dimensional tetrahedral element mesh. The shading denotes the four partitions that were computed by Metis.

In general terms, the mesh package classes perform the following functions.

- They read and store data from mesh input files. This includes element connectivity, node coordinates and element partitioning.
- They compute all required element and node geometrical information such as cell face areas and normal vectors for the global mesh.
- They compute all edge connectivity and geometric information from the element information for the global mesh.
- They link all nodes via edge elements.
- They set up partition meshes with links between global and partition mesh objects including nodes, elements and edges.
- They set up connectivity between duplicate nodes on different partitions.

4.7. Discrete operations package

The discrete operations package contains a class called Divergence which provides a variety of mesh-wide discrete operations including the computation of the divergence of a vector field, the gradient of a scalar at both mesh nodes and mesh faces as well as some more specialized operations. Some of the specialized operations include finding the maximum face-by-face inflow values for each node for advection calculations and finding the diagonal term of a mesh-wide matrix operator. All of these operations require communication and therefore use the duplicate node connectivity information from the mesh package classes and the barrier object from the communications package.

4.8. Physical properties package

Physical properties such as material densities and transport properties such as viscosities, mass diffusivities and thermal conductivities are required in simulations of physical systems. These quantities are often predicted using equations of state from system quantities such as temperature and pressure. The details of these predictions are kept separate from the solution of conservation equations by isolating the implementation in a separate package. This package contains, at present, classes for the prediction of material densities diffusivities, and inter-phase exchange parameters such as drag coefficients. The classes use information input by the user from the materials input pages of the GUI and provide methods to the physics package classes (see Section 4.9) for the materials properties predictions. Thus, the physical properties package classes also insulate the physics developers from changes in the details of the materials properties input specifications.

4.9. Physics package

The physics package contains classes that allow a developer to encode the conservation equations that he would like to solve. The developer first must set up an AbsState class corresponding to his physical system. AbsState is a container class (see Section 4.9.1). Once the AbsSstate class is set up, the user then can encode his conservation equations in an AbsProblemPhysics class. This is discussed in Section 4.9.2. When specifying both the AbsState class and the AbsProblemPhysics class, the user must extend abstract classes that provide the basic interface expected by the rest of CartaBlanca.

4.9.1. *AbsState class*

The AbsState class is an abstract class that must be extended by the developer to provide a data container for state variables for specific physics problems. The state variables are fundamentally stored in a two-dimensional array wherein the first dimension is the variable type and the second dimension is the node index. For example, if one is trying to solve a problem with state variables for pressure and three components of velocity, then the first dimension of this array would be four. The two-dimensional representation is convenient for developers since they tend to work with the governing equations one field or state variable type at a time. The two-dimensional view is also a convenient format for the graphics package since it also processes the data one field at a time.

Krylov solvers, however, work in terms of a one-dimensional state vector. Thus, the AbsState class also provides a one-dimensional view of the same state data. Currently, the one-dimensional view is provided as a copy of the two-dimensional data. The copy is performed using Java's System.arraycopy function for best performance.

4.9.2. *AbsProblemPhysics class*

For linear physical systems, developers can specify their physical system behavior by extending the AbsProblemPhysics class. AbsProblemPhysics is an abstract class that lays out what CartaBlanca expects from physics objects. The most important feature of this class of objects is the methods to get the right- and left-hand sides of the governing equations for the state variables. The solvers in CartaBlanca interact with these physics object methods to obtain the right-hand side of the linear equation system and the matrix–vector multiply. Another important behavior of AbsProblemPhysics objects is the pre-conditioning method. The Krylov solvers also interact with physics objects by invoking their pre-conditioning method. This method takes a Krylov vector from the Krylov solver and updates it according to some iterative improvement scheme. Currently, diagonal, Jacobi and symmetric successive over-relaxation pre-conditioning are available. Plans for a multigrid-like scheme are in place to obtain improved solver performance.

AbsProblemPhysics classes also inherit some methods for the base class for converting time n states to time $n + 1$ states. These methods, of course, can be overridden in the derived classes to provide additional functionality.

4.9.3. *NLAbsProblemPhysics class*

In the case of nonlinear physics problems, the matrix–vector multiply evaluation has to be provided in a generic fashion following Equation (9). The NLAbsProblemPhysics class of CartaBlanca extends the AbsProblemPhysics to provide this behavior. In this class of objects, the developer must encode the governing equations into methods that return the full nonlinear residual equation in the form of left- and right-hand sides. The left- and right-hand sides correspond to the implicit and explicit parts of the governing equations. These objects invoke these nonlinear *get* methods from the overridden linear *get* methods from AbsProblemPhysics class using Equation (9) to produce a linear matrix–vector multiply evaluation. Since NLAbsProblemPhysics inherits from AbsProblemPhysics, all other behavior, such as pre-conditioning, is also available.

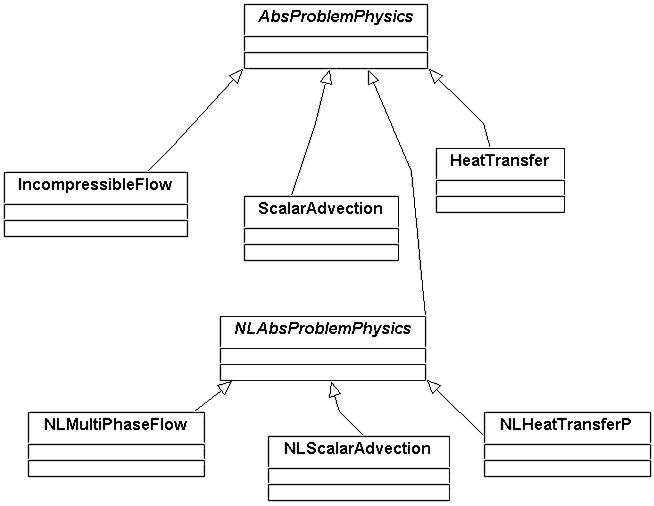


Figure 6. The Unified Modeling Language (UML) class hierarchy diagram of the physics package.

Figure 6 provides a graphical overview of the physics class inheritance hierarchy that was generated directly from the Java source code using GDPro from Embarcadero Technologies.

4.10. Solver package

The solver package contains classes for linear and nonlinear solvers. The solver class inheritance hierarchy is shown in Figure 7. As was the case for the classes in the physics package, an abstract solver class, AbsSolver, is provided as a parent for all solvers. Currently, this class has been extended to provide users with Krylov solver classes based on Conjugate Gradient and both the standard and flexible variant of Gmres [23]. The implementation of the flexible variant of Gmres, FGmres, is a classic case of code re-use. To obtain FGmres, we simply overrode three of the eight methods in Gmres. While Gmres contains 600 lines, FGmres contains only 122.

In addition, an 'explicit' solver is provided for fully explicit calculations which essentially bypasses any solution method at all and simply returns the right-hand side as the solution. Finally, a Newton–Krylov (JFNK) solver is provided for nonlinear problems. Each of these solvers communicates directly with physics objects through method invocations.

It is worth noting that Gmres, Fgmres and CG are linear system solvers while NewtonGmres is a nonlinear system solver. All solvers inherit from AbsSolver, which provides the basic interface for interacting with physics objects. NewtonGmres includes a Gmres linear solver. This is an example of a 'has a' relation. The Gmres solver is used to find the solution of the linearized system in each of the Newton iterations.

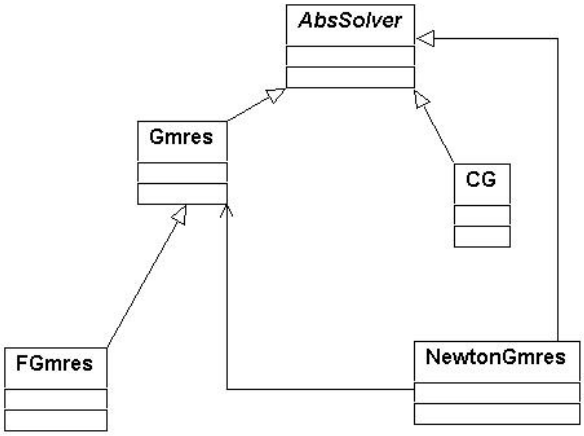


Figure 7. The solver package class hierarchy.

4.11. Graphics package

The Graphics package, at present, contains only one class that can be used to produce Tecplot format output text files. The class interacts with the abstract state class so that it automatically knows about new state variables, etc. Eventually, this class will be extended to allow for additional plot file output formats. We also envision direct use of Java graphics.

4.12. ProblemDriver package

The ProblemDriver package contains the Driver class, a top-level driver for solving physics problems on each mesh partition. The Driver class implements Java's Thread-class Runnable interface. This enables data-parallel computation in CartaBlanca with each thread corresponding to a particular mesh partition. Figure 8 shows a UML association diagram for the Driver class. As can be seen, the Driver class interacts with all the major CartaBlanca objects from an AbsProblemPhysics object to an AbsSolver object.

4.13. Boundary conditions package

Boundary conditions are required for the complete specification of all but the simplest physical problems. In order to build a layer of abstraction between the core physics classes and the user interface for boundary conditions, we have introduced a separate boundary conditions package. At present, this package contains only one class, which implements boundary conditions. This class takes user input data from the ProblemSpecifier object and provides methods for setting boundary fluxes for use in the conservation equations in the physics classes.

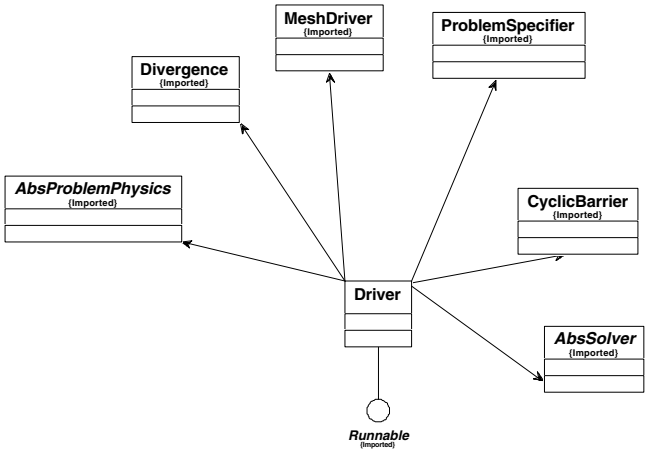


Figure 8. The association diagram for driver class.

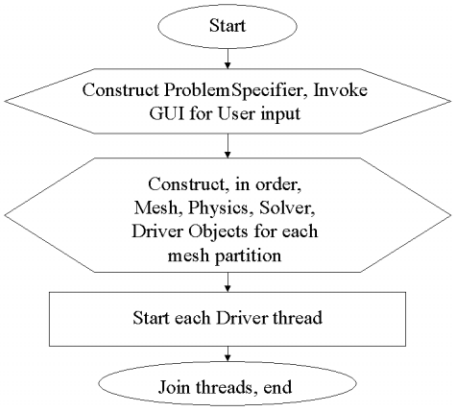


Figure 9. A flow chart for the CartaBlanca main method.

4.14. Main package

The main package consists of several classes that contain the public static main method that drives the entire simulation. The class PhysMain contains a main method that instantiates all high-level objects and invokes the start method for all of the Driver objects for each mesh partition. This sequence is depicted in Figure 9. The relation between the main method in the PhysMain class to top-level objects is depicted graphically in Figure 10.

The main method in class PhysMain spawns multiple threads for each of the mesh partitions. Execution on the threads is controlled by the Driver objects, which invoke the Solver objects.

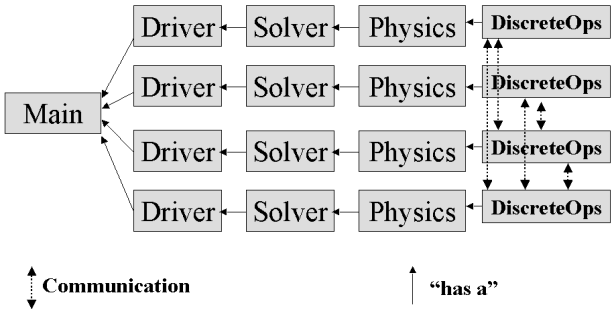


Figure 10. The relation of the main method in PhysMain to top-level objects.

The Solver objects interact with the Physics objects which make use of the discrete operation objects. Inter-thread communication is handled in and among the discrete operation objects. All of these objects contain the MeshDriver objects.

5. RESULTS

Here we provide information on compilation, run-time performance and some preliminary simulation results from CartaBlanca. Computations were performed on a dual 500 MHz Pentium III processor SGI 320 PC running a Windows NT operating system, on a single processor Sun Ultra 60 workstation running the Solaris 2.7 operating system and on a four-processor SGI Origin 200 workstation running the IRIX operating system. The entire software package compiles to bytecode in a matter of a few seconds.

5.1. Scalar advection

The first two examples are of explicit scalar advection calculations done on the triangular element grid shown in Figure 4. The results of the calculations are shown in Figures 11 and 12. These calculations were performed using two threads in parallel on our dual-processor SGI PC. Each thread operated on a separate mesh partition.

Each calculation started with a pulse of concentration in the lower left-hand corner of the domain. In the first example, the concentration pulse has a Gaussian spatial distribution, while in the second example the pulse was a spatial step function with circular shape. This material was advected towards the top right according to the conservation equation

$$\frac{\partial c}{\partial t} + \frac{\partial c u_i}{\partial x_i} = 0 \tag{10}$$

where c is the concentration, u_i is the ith component of velocity, t is time and x_i is the ith spatial coordinate. There is an implied summation over the repeated index in the second term. In these example

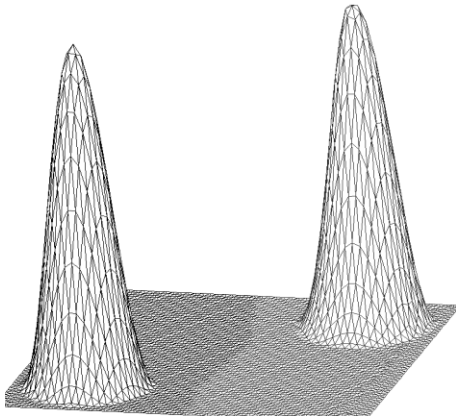


Figure 11. Results from a continuum scalar advection simulation on a two-dimensional triangular mesh.

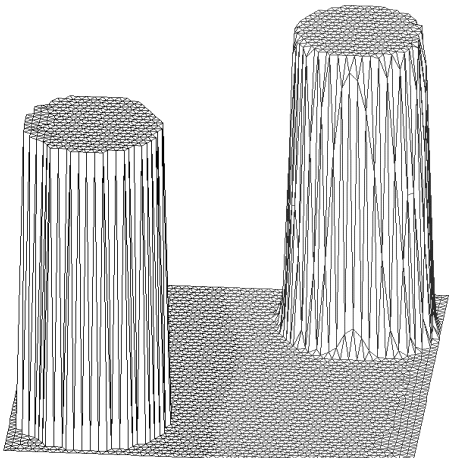


Figure 12. Results from a interface-tracking scalar advection simulation on a two-dimensional triangular mesh.

calculations, the velocity was a constant with component values of one in the horizontal and vertical directions. The final conditions of the concentration pulses are shown in the top-right of Figures 11 and 12. Note that in the case of Figure 11 we used the continuous field version of CartaBlanca's advection facility while in Figure 12 we used the interface-tracking version of the advection facility [24].

The calculations used CartaBlanca's explicit solver to advance the solution of Equation (10) over 120 time cycles. Although Equation (10) is relatively simple, it is commonly known that it is difficult to

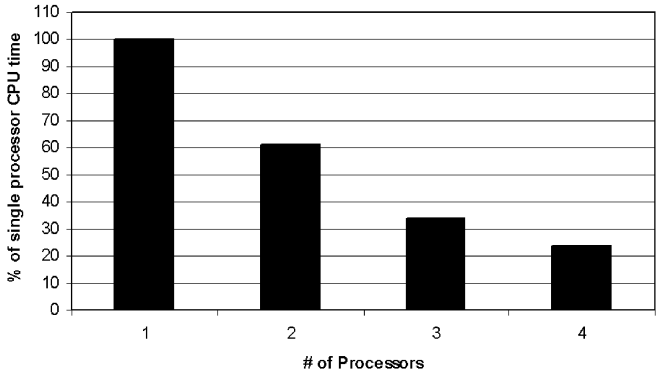


Figure 13. A preliminary example of the scaling behavior of CartaBlanca on the linear heat transfer problem. Values are the wall clock time for the problem solution scaled by that for a single processor calculation.

obtain accurate numerical solutions for this equation that minimize the artificial numerical smearing of such concentration pulses and also avoid the creation of artificial extrema, especially on triangular grid meshes. Also, doing such calculations using interface tracking is also a very difficult task. These examples demonstrate that CartaBlanca has advanced advection algorithms that can be used by developers for a wide variety of applications.

5.2. Implicit heat transfer

The second example is a simple implicit heat transfer calculation performed on similar triangular element meshes. CartaBlanca's flexible Gmres solver was used without preconditioning to compute the temperature field as a function of time according to the equation

$$\frac{\partial T}{\partial t} = \frac{\partial}{\partial x_i}\left(\alpha \frac{\partial T}{\partial x_i}\right) \tag{11}$$

where T is the temperature and α is the thermal diffusivity, which was set to one for the example problem. Zero-gradient boundary conditions were used which correspond physically to a perfectly insulated box. The final state is, therefore, a uniform temperature of one-half. CartaBlanca computed the correct solution on a 46 000-node triangular element mesh, using both a single mesh partition and multiple mesh partitions running on separate threads in parallel.

Figure 13 shows the scaling behavior for this heat transfer problem on a four-processor SGI Origin 200 workstation. These are very preliminary results and are likely to change as we perform more tests on the parallel performance of CartaBlanca. They do show, however, that CartaBlanca appears to scale well for this problem.

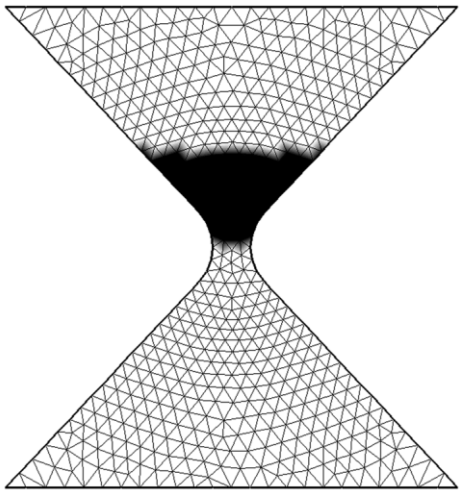


Figure 14. Initial conditions and the mesh for the hourglass problem. The dark region is 60% sand and 40% water by volume.

5.3. Multiphase flow

We can also compute multiphase flow using CartaBlanca. The details of the governing equations and the numerical algorithm are beyond the scope of this publication. Those interested in the governing equations for multiphase flow should consult, for example, [25]. Our numerical algorithm is closely related to the one used in CFDLIB [10]. We plan to publish a detailed exposition of our algorithm in the near future. In this paper we demonstrate our ability to compute multiphase flows on complex geometries and show the performance of CartaBlanca compared to the CFDLIB FORTRAN code. Our first example calculation is of the flow of sand grains and water in an hourglass. Our second example is a computation of a so-called broken dam experiment due to Martin and Moyce [26].

5.3.1. Hourglass flow

We show here some results from computations performed on a triangular element mesh in the shape of an hourglass. Figure 14 shows both the mesh and the initial conditions for the calculation. The mesh contains 1018 elements and 578 nodes. The overall dimensions of the problem are 10 cm by 10 cm. The width of the neck of the hourglass is 1 cm. The contours indicate the initial distribution of materials. The dark region in the middle of the hourglass is filled with 60% (by volume) sand and 40% water. The remainder of the space is filled with water. At time zero, gravity is 'turned on' and the sand begins to fall through the hourglass throat. The computation of this example problem was performed with CartaBlanca on our SGI PC using a single mesh partition. Figures 15 and 16 show the time evolution as calculated by CartaBlanca. The computation took about 15 minutes on the PC.

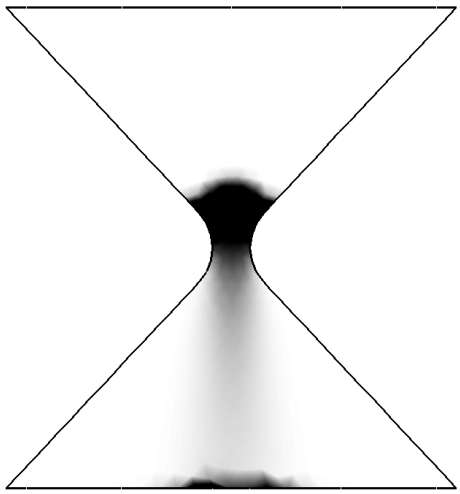


Figure 15. The hourglass flow problem at 0.2 s, real time.

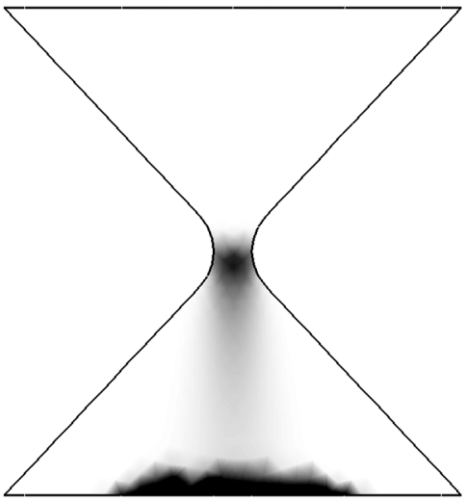


Figure 16. The hourglass flow problem at 0.36 s, real time.

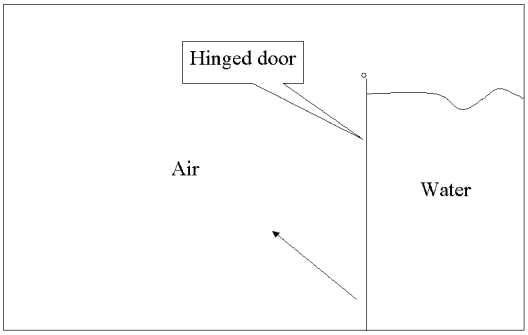


Figure 17. A schematic side-view diagram of the broken dam experiment.

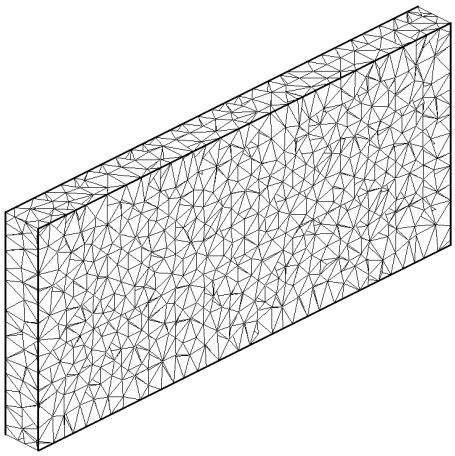


Figure 18. Tetrahedral element mesh for the broken dam problem.

5.3.2. Broken dam flow

The final example calculation is of a broken dam experiment performed by Martin and Moyce [26]. Martin and Moyce constructed a small box with dimensions of 28.57 cm by 12.57 cm by 2.3 cm out of plexiglass. Inside the box, they installed a hinged door that served as a dam for water at one end. The door was hinged so that it could be opened quickly to allow water to fall freely under the action of gravity. The experimental set-up is shown in Figure 17.

Martin and Moyce recorded the time and position of the front of water that moved across the bottom of the apparatus. We have used CartaBlanca to successfully simulate the Martin and Moyce broken dam experiment on a variety of two- and three-dimensional meshes composed of quadrilateral, triangular,

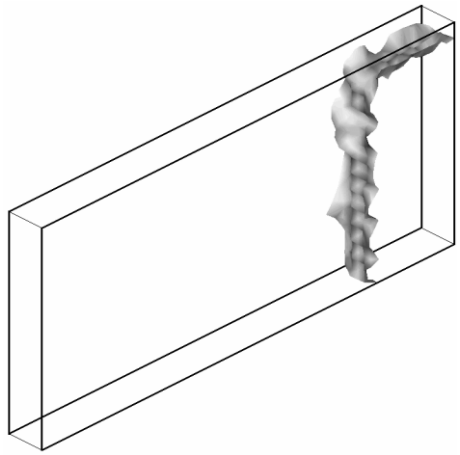


Figure 19. The initial condition for the broken dam experiment simulation. The isosurface shows the interface between water and air. Water is to the right and below the interface.

hexahedral and tetrahedral elements. Figure 18 shows, for example, the tetrahedral element mesh we used. This mesh consisted of 10 076 elements and 2205 nodes.

Figure 19 shows the initial condition of the interface for the calculation on the tetrahedral element mesh. The 'bumpiness' of the interface is an artifact of the irregular tetrahedral mesh and our, at present, simplified way of initializing materials in CartaBlanca. Currently, we simply assign to each node a volume fraction of either 0 or 1 according to its position relative to the air–water interface. In the future, we will implement the ability to assign fractional volume fractions to nodes that intersect the interface using a particle interpolation technique.

In Figures 20–24 we show the progression of the air–water interface after opening of the dam door to the point in time at 0.2 s where the water reaches the far end of the box.

In Figure 25 we show a quantitative comparison of the water front position between the experiments of Martin and Moyce and the calculations using CartaBlanca. The CartaBlanca results are a little ahead of the Martin and Moyce data and may signal the need for improvements in the physical models incorporated into CartaBlanca. (The CartaBlanca simulation neglected viscous and surface tension effects.)

5.4. Performance

Our last task here is to give some measure of the performance of CartaBlanca. To provide a basis of comparison, we used CFDLIB [10] and CartaBlanca to perform simulations of the Martin and Moyce broken dam problem on both two- and three-dimensional Cartesian meshes. Furthermore, we, ran both codes using the same treatment of advection mass fluxes, namely, first-order accurate donor cell advection. Finally, we used the Conjugate Gradient method for the solution of the pressure equation in

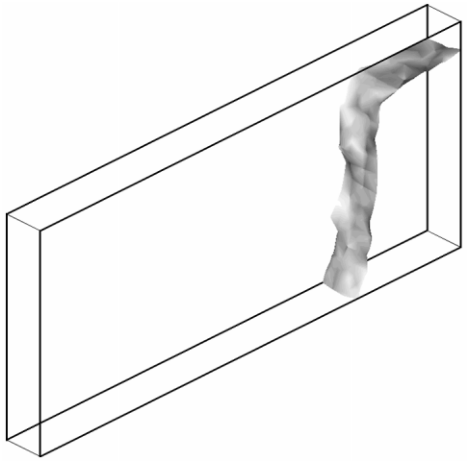


Figure 20. The interface at 0.04 s.

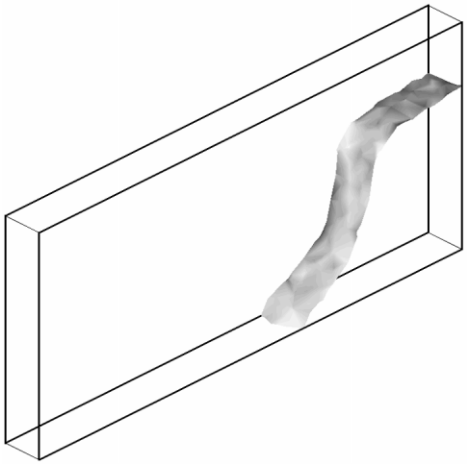


Figure 21. The interface at 0.08 s.

CFDLIB and for the pre-conditioning of the nonlinear pressure residual in CartaBlanca. Thus, the two sets of calculations, whilst not identical, were as close to being the same as was possible in terms of the numerical methods used. Table I provides a comparison of the speed of the codes in terms of the so-called grind time—the average wall-clock processing time per time step per node.

All calculations were performed on our Sun Ultra 60 workstation running Solaris 2.7. For the CartaBlanca calculations, we used Sun JDK 1.3.1 with the HotSpot JIT. For the CFDLIB calculations, we used the Sun Fortran 77 compiler version 5.0 with optimization. As can be seen from the table,

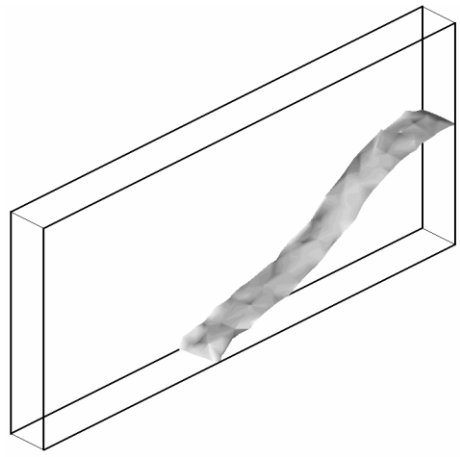


Figure 22. The interface at 0.12 s.

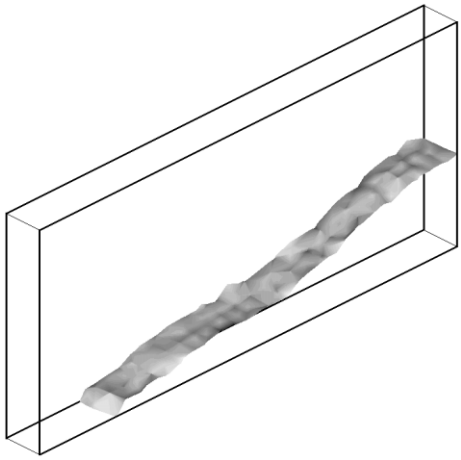


Figure 23. The Interface at 0.16 s.

CartaBlanca achieved 46% of the speed of CFDLIB in the two-dimensional case and 51% of CFDLIB in the three-dimensional case. While this is not a perfect side-by-side comparison of Java and Fortran, it is a reasonably close comparison. The results are quite pleasing to us when we consider that CFDLIB is a highly optimized Fortran code, which has many man-years of effort behind it and a worldwide user base. Furthermore, CFDLIB is recognized as a fast multiphase flow code by our users. CFDLIB was written for structured grids and does not use indirect addressing, as does CartaBlanca.

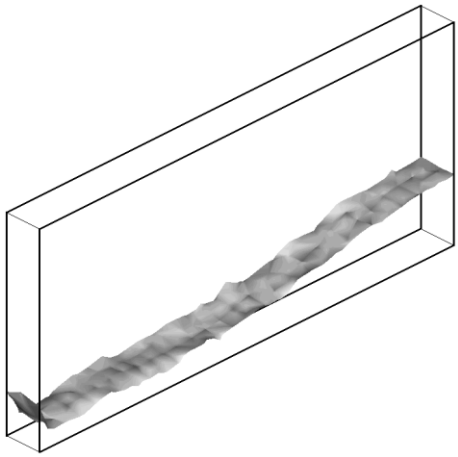


Figure 24. The interface at 0.2 s.

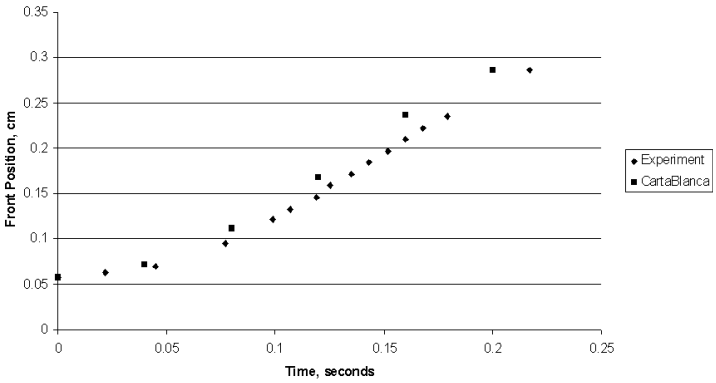


Figure 25. Comparison of the CartaBlanca computation and the data of Martin and Moyce for the water front position as a function of time.

We believe that the favorable speed comparison between CartaBlanca and CFDLIB is due to two factors. First, our basic design choice discussed in Section 4.1, wherein most our objects are things which exist over entire mesh partitions, has allowed us to avoid excessive overhead from operations like run-time type identification, etc. If, for example, we had used objects extensively at the computational node level, we would have seen much poorer performance due to excessive overhead. The second factor accounting for our favorable speed comparison is the use of the modern Just-In-Time (JIT) compiler. Turning off the JIT compiler typically results in an order of magnitude loss of execution speed for

Table I. Performance comparisons between CartaBlanca and CFDLIB on the broken dam problem using first-order (donor cell) fluxes for advection. Table entries are 'grind times' defined as the average wall-clock processing time in microseconds per time step per node.

Case\code	Elements	Time steps	CartaBlanca	CFDLIB
Two-dimensional quadrilateral element mesh	1100	261	321	147
Three-dimensional hexahedral element mesh	4400	279	779	400

CartaBlanca. These results are consistent with those reported by Boisvert *et al.* [8]. These two factors are related. Our restrained use of objects enables effective optimization from JIT compilers. At the same time, we still benefit from the use of object-oriented programming in terms of code organization and code re-use.

6. CONCLUSIONS AND FUTURE PLANS

Since we are still in the development phase of this project, it is not appropriate to provide final assessments of the design and performance of CartaBlanca. In a qualitative sense, we can say that CartaBlanca has already been a useful tool for algorithm development since we have been able to do the research on our new advection scheme exclusively within CartaBlanca [24]. That is, CartaBlanca has provided us with sufficient usability and performance that we did not choose to go 'off-line' to some other program or development tool to do one of our main jobs, algorithm research and development. Therefore, in this sense, we can already claim some 'bottom-line' success.

The performance comparison of CartaBlanca with CFDLIB is quite encouraging. We hope that with further improvements to the CartaBlanca software and with improved Java compilers and Virtual Machines we will see the gap close.

We plan to pursue a number of promising leads in the near future. The following is a partial list covering some of the major possible directions.

- To fully investigate the use of Java native-code compilers. The GNU compiler, GCJ, is particularly attractive in that it provides the option of turning off array-bounds checking. This could provide significant speed-up [27].
- To incorporate fluid–structure interactions to the nonlinear multiphase flow class for solidifying flow and elastic–plastic flow simulations.

- To extend CartaBlanca to distributed-memory architecture for cluster-based computing. We are considering the use of JavaParty [28] or Jackal [29,30] as a means to extend CartaBlanca to distributed-memory systems.

ACKNOWLEDGEMENTS

We gratefully acknowledge the support for this work from the Department of Energy and the Los Alamos Computer Science Institute (LACSI).

REFERENCES

1. Brown PN, Saad Y. Hybrid Krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comput.* 1990; **11**(3):450–481.
2. Ferziger JH, Peric M. *Computational Methods for Fluid Dynamics*. Springer: New York, 1999.
3. VanderHeyden WB, Dendy ED, Padial-Collins NT. CartaBlanca—a pure-Java, component-based systems simulation tool for coupled nonlinear physics on unstructured grids. *Proceedings for the ACM 2001 Java Grande/ISCOPE Conference*. ACM Press: New York, 2001.
4. Karypis G, Kumar V. *METIS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN. http://www-users.cs.umn.edu/~karypis/metis/index.html [20 September 1998].
5. JUnit. http://www.JUnit.org/.
6. Schatzman J, Donehower R. High-performance Java software development. *Java Report* 2001; **6**(2):24–41.
7. Shirazi J. *Java Performance Tuning*. O'Reilly: Cambridge, 2000.
8. Boisvert RF, Moriera J, Phillipsen M, Pozo R. Java and numerical computing. *Computing in Science and Engineering* 2001; **3**(2):18–24.
9. Reinholtz K. Java will be faster than C++. *ACM Sigplan Notices* 2000; **35**(2):25–28.
10. Kashiwa B, Padial NT, Rauenzahn RM, VanderHeyden WB. A cell-centered ICE method for multiphase flow simulations. *Numerical Methods in Multiphase Flows, ASME* 1994; **185**:159–176.
11. Addessio FL, Baumgardner JR, Dukowicz JK, Johnson NL, Kashiwa BA, Rauenzahn RM, Zemach C. CAVEAT: A computer code for fluid dynamics problems with large distortion and internal slip. *Los Alamos National Laboratory Report LA-10613-MS, Rev. 1*, May, 1992.
12. Telluride. *Los Alamos National Laboratory Report LA-UR-99-1664*, 1999. http://public.lanl.gov/mww/HomePage.html.
13. O'Rourke PJ, Sahota MS. CHAD: A parallel, 3-D. Implicit, unstructured-grid, multimaterial, hydrodynamics code for all flow speeds. *Los Alamos National Laboratory Report LA-UR-98-5663*, October 1998.
14. Selmin V. The node-centered finite volume approach: Bridge between finite differences and finite elements. *Comput. Methods in Appl. Mech. Engrng.* 1993; **102**(1):107–138.
15. Hauser J, Ludewig T, Gollnick T, Winkelman R, Williams R, Muylaert J, Spel M. A pure Java parallel flow solver. *AIAA Paper 99-0549*.
16. Hatakeyama M, Watanabe M, Suzuki T. Object-oriented fluid flow simulation system. *Computers and Fluids* 1998; **27**(5):581–597.
17. Kelly CT. *Iterative Methods for Linear and Nonlinear Equations*. SIAM: Philadelphia, 1995.
18. Fowler M. Put your process on a diet. *Software Development Magazine* 2000; **2**(12):32–36.
19. Chapman SJ. *Java for Engineers and Scientists*. Prentice-Hall: Upper Saddle River, NJ, 2000.
20. Harold ER. *Java I/O*. O'Reilly: Cambridge, 1998.
21. Eckstein RM, Loy M, Wood D. *Java Swing*. O'Reilly: Cambridge, 1998.
22. Oaks S, Wong H. *Java Threads*. O'Reilly: Cambridge, 1999.
23. Saad Y. *Iterative Methods for Sparse Linear Systems*. PWS Publishing: San Fransisco, 1995.
24. Dendy ED, Padial-Collins NT, VanderHeyden WB. A general purpose, finite-volume advection scheme for continuous and discontinuous fields on unstructured grids. *Journal of Computational Physics*. In press.
25. Drew DA, Passman SL. *Theory of Multicomponent Fluids*. Springer: New York, 1999.
26. Martin JC, Moyce WJ. An experimental study of the collapse of liquid columns on a rigid horizontal plane. *Philosophical Transactions of the Royal Society* 1952; **A244**:312–324.

27. http://gcc.gnu.org/java/index.html.
28. Philippsen M, Zenger M. JavaParty: Transparent remote objects in Java. *Concurrency—Practice and Experience* 1997; **9**:1225–1242.
29. Veldema R, Hofman RFH, Jacobs C, Bhoedjang RAF, Bal HE. Jackal, a compiler-supported distributed shared memory implementation of Java. Submitted for publication. http://www.cs.vu.nl/rveldema/jackal-2001.ps [2001].
30. Veldema R, Hofman RFH, Bhoedjang RAF, Bal HE. Runtime optimizations for a Java DSM implementation. *Proceedings for the ACM 2001 Java Grande/ISCOPE Conference*. ACM Press: New York, 2001.

