

2005 R&D 100 Entry
Brian VanderHeyden

CartaBlanca


A High-Efficiency, Object-Oriented, General-Purpose
Computer Simulation Environment

Unlocks business-proven Java
efficiency for scientific computing

Integrates Extreme Programming's JUnit
testing for efficient team development

Uses advanced numerical algorithms

Designed for complex multiphase and
fluid-structure interaction problems

 **Los Alamos**
NATIONAL LABORATORY
EST. 1943

2005 R&D 100 Entry

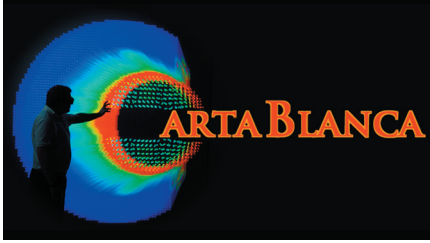
CartaBlanca: A High-Efficiency, Object-Oriented, General-Purpose Computer Simulation Environment

Brian VanderHeyden

ABOUT THE COVER

CartaBlanca, the first Java-based simulation software package, can be used to simulate explosions of different kinds. Shown is a simulation of an exploding cylindrical blast container; these containers are used to dispose of an improvised explosive device (IED). The simulation shows the motion of the broken IED hemispherical case, blast wave through the gas (air), and deflection and deformation of the cylinder. The colors show the different local particle pressures. These results are being visualized at Los Alamos National Laboratory's RAVE (Reconfigurable Advanced Visualization Environment) facility, where scientists can see the data in a three-dimensional mode that provides greater insight into the details of the simulation.

The Regents of the University of California have rights in this submittal under their contract with the DOE for operating Los Alamos National Laboratory. Distribution and use of the submittal except for purposes of award review must receive prior approval from The Regents of the University of California.



Executive Summary

CartaBlanca: A High-Efficiency, Object-Oriented, General-Purpose Computer Simulation Environment

Features

CartaBlanca brings the tremendous efficiency of the Java programming language to the world of scientific computing. The first of its kind, CartaBlanca is a state-of-the-art, object-oriented simulation software package poised to offer next-generation modeling and simulation capabilities to scientists in a wide-ranging number of disciplines. Written in the “developer friendly” Java language, it enables computer code developers to simulate complex nonlinear effects such as airflow through a turbo booster, blast effects on buildings, or heat transfer along a semiconductor, to name but a few of its many applications. Because CartaBlanca is a Java-based software package, the code is much easier to use, manipulate, and modify than are codes based on such programming languages as FORTRAN or C++. CartaBlanca takes advantage of the improved execution speed offered by the HotSpot™ compiler, which allows performance on par with FORTRAN and C++. CartaBlanca opens up the field of physical modeling to a much broader set of programmers because Java is one of the most common and efficient business software languages. CartaBlanca is modular and allows for rapid software application or simulation code prototyping; strong, extensive compiler checking; plug-and-play module insertion for modeling physical systems; solutions with consistent results; and integrated unit and regression testing.

Applications

- Aerospace engineering
- Animation and special effects
- Computational fluid dynamics
- Fluid/solid interactions
- Automotive design
- Weapon/target interactions
- Pharmaceutical processing
- Homeland defense

Benefits

- Provides accurate, physics-based computer simulations in Java
- Provides faster and lower-cost development
- Allows for easily modified and integrated code
- Runs on most hardware platforms without modification, from single PCs and Macs to parallel-processing supercomputers
- Increases software developer productivity
- Allows state-of-the-art simulations for complex reactive flows

2005 R&D 100 Award Entry Form

1. Submitting organization

Organization Los Alamos National Laboratory
Address Mail Stop B216
City, State, ZIP Los Alamos, NM 87545
Country USA
Submitter W. Brian VanderHeyden
Phone (505) 667-9099
Fax (505) 665-5926
E-mail wbv@lanl.gov

AFFIRMATION: I affirm that all information submitted as a part of, or supplemental to, this entry is a fair and accurate representation of this product.

(Signature) W. B. VanderHeyden

2. Joint submitter

N/A

3. Product name

CartaBlanca: A High-Efficiency, Object-Oriented, General-Purpose Computer Simulation Environment

4. Brief product description

CartaBlanca is the first Java-based, object-oriented simulation software package that provides simulation prototyping research code to use in modeling a wide variety of physical systems such as chemical reactors, projectile/target interactions, and urban bomb blasts.

5. Eligibility: When was this product first marketed or available for order?

Month December
Year 2004

6. Principal developer

Name W. Brian VanderHeyden
Position Technical Staff Member
Organization Los Alamos National Laboratory
Address Mail Stop B216
City, State, ZIP Los Alamos, NM 87545
Country USA
Phone (505) 667-9099
Fax (505) 665-5926
E-mail wbv@lanl.gov

7. *Product price*

\$10,000 per license (preliminary price)

8. *Do you hold any patents or patents pending on this product?*

YES Patent pending, and this technology is copyrighted.

9. *Primary function*

Modern scientific and engineering simulation projects have become large-scale and complex endeavors. Very often, simulation projects involve the modification of existing software to produce new capabilities. These projects typically require constant and ongoing attention from a variety of code developers. In fact, on some scientific projects, code developers outnumber the actual users for a significant portion of the lifetime of the project. As such, it is useful for scientific and engineering code developers to use modern productive software languages to produce a “developer-friendly” simulation code. It is just as important for such code to be developer-friendly as it is to be user-friendly. This approach is different from the mainstream approach where user-friendly interfaces are wrapped around less-developer-friendly legacy code written in FORTRAN or C++. Developers need to collaborate with each other and simultaneously use the codes to develop large-scale simulations. CartaBlanca is built for just such a purpose.

CartaBlanca is an object-oriented, nonlinear simulation and prototyping software package whose primary function is to assist code developers in solving a wide range of hydrodynamics and fluid/structure interaction problems. Because CartaBlanca is written entirely in Java, it provides scientists and engineers with developer-friendly software to use in producing large-scale computational models. CartaBlanca allows users to solve a variety of nonlinear physics problems, including multiphase flows, interfacial flows, solidifying flows, and complex material responses. CartaBlanca makes use of the powerful, state-of-the-art Jacobian-free Newton-Krylov method to solve nonlinear equations in a flexible unstructured grid finite-volume scheme. CartaBlanca couples the particle-in-cell (PIC) method—a technique used to model discrete objects—with its multiphase flow treatment to model how fluids interact with solid materials that can undergo deformation, damage, and failure.

Because of Java’s marketplace strength, a wealth of third-party off-the-shelf software components is available for use. One notable example is the JUnit testing framework that allows users to write repeatable tests and a series of typical problems. CartaBlanca uses this third-party software to allow developers to see where the “bugs” are or where their code breaks down. In fact, because CartaBlanca is Java based, many other Java-based components can be used with

the software package. Examples include existing databases and computer-aided design tools.

The Java language includes a facility for spawning “threads” or processes that run simultaneously and can communicate with each other, but are controlled from the same program. CartaBlanca’s software design uses this ability to enable data-parallel, shared-memory, and distributed-memory computations on a wide variety of unstructured grids with triangular, quadrilateral, tetrahedral, and hexahedral elements. This design allows CartaBlanca to handle complex geometrical shapes and mathematical domains. Java’s simplicity greatly simplifies both code maintenance and modification because the same code is used for both serial and parallel calculations. Java’s object inheritance allows developers to construct a hierarchy of physics-systems objects, linear and nonlinear solver objects, and material response behavior objects. This hierarchy simplifies code development while fostering software reuse. CartaBlanca’s multithread nature permits easy portability to networked computers with the help of third-party distributed shared-memory systems such as JavaParty (<http://www.ipd.uka.de/JavaParty>).

Finally, CartaBlanca employs Java’s powerful swing-graphics facility to provide an extensive, user-friendly graphical interface, or GUI, for problem specification and data input.

Details of the theoretical approaches used in the development of CartaBlanca are provided in the appendix in “CartaBlanca—A Pure-Java, Component-based Systems Simulation Tool for Coupled Nonlinear Physics on Unstructured Grids” (excerpt) and “Implementation and Performance of a Particle in Cell Code Written in Java” (excerpt). The appendix also includes letters of recommendation and examples of CartaBlanca’s structure and capabilities.

10A. List of competitors

CFDLib—A Los Alamos library of computer codes written in FORTRAN 77. These codes are capable of solving a wide range of computational fluid dynamics problems (<http://www.lanl.gov/orgs/t/t3/codes/cfdlib.shtml>). The library is available (for a fee) from the Energy Science and Technology Center (ESTSC) (phone 865-576-2606, estsc@adonis.osti.gov).

CHAD (Computational Hydrodynamics for Advanced Design)—Another computer code developed at Los Alamos National Laboratory. CHAD has been intensively used by the automobile industry and in defense projects. It has been written to take full advantage of parallel computers, specifically for chemically reactive flows. For information, contact Manjit Sahota (sahota@lanl.gov).

FLUENT—A commercial computational fluid dynamics software package developed and marketed by Fluent, Inc., of Lebanon, New Hampshire. Fluent, Inc., is the world's largest provider of computational fluid dynamics software and consulting services. The FLUENT software is written in the C++ language and employs an unstructured grid capability.

10B. Comparison matrix

Parameters	CartaBlanca	CFDLib	CHAD	FLUENT	Comments
Written in JAVA?	Yes	No	No	No	Java is a robust, type-safe, modern computing language. Java has been demonstrated to be a more efficient language for development when compared with conventional languages such as C++ and FORTRAN. Because of a very strong and growing marketplace presence, Java leads to a huge number of programmer tools. Java is also becoming a competitive language for both business and scientific applications.
Object-Oriented?	Yes	No	No	No	Being object-oriented makes a computer language developer-friendly and enhances programmer productivity. CFDLib, CHAD, and FLUENT use conventional procedural computer programming style and, as a result, do not benefit from the efficiency features of object-oriented Java.
Built-In Thread Parallelization?	Yes	No	No	No	CartaBlanca uses a built-in thread parallelization for shared-memory machines. Third-party programs are available to allow the execution of a Java code in a distributed memory system. In FLUENT, CHAD, and CFDLib, the use of parallel code in shared-memory or distributed-memory machines requires heavy, separate reprogramming and maintenance of both serial and parallel codes.
Allows Easy Code Testing?	Yes	No	No	No	Because of the wealth of third-party software packages available, off-the-shelf testing facilities such as JUnit can be used with CartaBlanca in compiling and testing their code.
Graphic User Interface (GUI)?	Yes	No	No	Yes	A GUI can be written easily using Java's swing library. This ability makes the program very user-friendly, eliminating the need for text-based inputs except for the mesh files.
Reflection?	Yes	No	No	No	Reflection, a feature unique to the Java language, allows information about classes and data structures to be retrieved and manipulated by name within an executing Java program. Objects can be queried as needed for program information. The object can also be used to save the information settings to the directory, and it also allows for drop-in components.
Extreme Portability?	Yes	No	No	No	A Java code is portable even without the source codes. Codes written in other languages must always be recompiled and, frequently, have the source modified in moving from one machine to another.
Multiphase Flow?	Yes	Yes	No	Yes	CartaBlanca incorporates true multiphase flow, which involves the integration of multiple momentum equations for each phase in the multiphase flow and allows slippage between the phases. One simple example of a multiphase flow would be air bubbles in water. Multiphase flow simulation is a difficult technology but one with many applications in industry and homeland defense.
Uses Jacobian-free Newton-Krylov?	Yes	No	No	No	CartaBlanca uses a Jacobian-free Newton-Krylov solver, a mathematical technique that provides robust, fully coupled solutions for nonlinear systems of equations.
Allows Particle/Fluid Interactions?	Yes	Yes	No	No	CartaBlanca implements particle/fluid interactions using the PIC method and the material point method (MPM). This is the most advanced feature available in CartaBlanca. By using the PIC method in combination with MPM, CartaBlanca, like CFDLib, can simulate the interactions of fluids and fluid blast waves with solid materials, as well as the effects of impellers in mix tanks. All of these simulations can be developed without complicated moving meshes or associated mesh-tangling problems.

CartaBlanca is written in a widely used business computer language. CartaBlanca (Version 2.0) is a large-scale, scientific and engineering software application written in Java. CFDLib is written in FORTRAN 77, which is older technology. CHAD is written in FORTRAN 95, which is object-based but not object-oriented and requires greater effort from the developers. The inherent parallelism of the Java language leads to automatic parallel code on shared-memory machines.

CartaBlanca minimizes development efforts for simulation projects. Using object-oriented Java, code developers can produce a mature code more quickly, with better interaction between developers, and without sacrificing performance. Because Java is a clean, type-safe programming language, CartaBlanca can support a highly interactive and interdependent development team. Using CartaBlanca, developers can cut simulation code development time by a third.

CartaBlanca is user-friendly. Since Java supports GUIs, CartaBlanca is very user-friendly. Because of the GUI interface, both specifying the problem and inputting the data are much easier in CartaBlanca than in CFDLib, CHAD, or FLUENT.

CartaBlanca's speed is comparable to the competitors' speed. The performance speed of Java is now comparable to FORTRAN's, and because so many code developers work in Java, its performance will continue to improve. Currently, CartaBlanca achieves performance speeds equivalent to the speeds of similar C language codes. We expect CartaBlanca's performance speed to increase in the future because of the attention industry is paying to Java.

CartaBlanca, even in compiled form, is portable without any modification. Both CFDLib and CHAD must be frequently modified and always recompiled when transferred from one computer to another. Because CartaBlanca is written in Java, the code does not need to be modified or recompiled when it is transferred from one computer to another.

CartaBlanca uses a Jacobian-free Newton-Krylov solver. This solver is an improvement upon our competitors. The Jacobian-free Newton-Krylov solver enables much faster solution of the conservation equations than is possible with the relatively primitive solvers available in CFDLib and CHAD.

CartaBlanca uses the particle-in-cell (PIC) and material point method (MPM). CartaBlanca, like CFDLib, uses the PIC method and MPM, along with the multiphase flow formulation, to enable the simulation of complex fluid structure-interaction problems such as a blast between buildings.

11A. Principal applications

Aerospace and Defense—CartaBlanca is used to simulate projectile/target interactions for the Army. The PIC method in CartaBlanca was used to simulate the large deformations in both the projectile and target. Large-deformation simulations are easier to produce than those using conventional Lagrangian mesh approaches that suffer from mesh tangling, or those using a pure Eulerian approach, which suffer from excessive artificial diffusion or boundary smearing.

Chemical Process—CartaBlanca is used to simulate and optimize centrifugal contactor-separator devices for the recovery of actinides from a waste stream. CartaBlanca's multiphase flow and unstructured grid technology is key for developing these simulations.

Homeland Security—CartaBlanca is used to simulate the effects of a small nuclear blast between buildings. The calculation includes not only the effect of the blast wave through the atmosphere, but also the transmission of blast and damage in the building materials.

Heat Transfer/Phase Change—CartaBlanca is used to develop simulation algorithms using the Newton-Krylov solver for phase-changing flows in metals such as those encountered in industrial casting operations.

Reactive Flows—CartaBlanca is used to develop models for safe handling of high-explosive charges. These simulations use CartaBlanca's multiphase flow code with detonation chemistry.

11B. Other applications

Animation and Special Effects—Physical models are used in the entertainment field for special effects depicting realistic fire, water, air, hair, and moving objects. CartaBlanca can be used to simulate the movement of hair and other special effects for computer games, computer animation, video games, and the film industry.

Automotive Industry—CartaBlanca can be used to simulate reactive flows in combustion chambers. The PIC method can be used to simulate moving valves and pistons in an engine and show the effects of heat transfer in each of the moving parts.

Biomedical Industry—CartaBlanca can be used to simulate pharmaceutical centrifugal devices used to separate liquids and mass exchange operations. The software allows engineers to optimize device designs.

Oil and Gas Industry—Process equipment design for oil production fields, including separators for drilling fluids, can be simulated and optimized with CartaBlanca.

Environmental Industry—CartaBlanca can be used to design and optimize waste cleanup operations.

Nuclear Plant Safety Industry—CartaBlanca can be used to accurately simulate the multiple physics involved in a nuclear plant coolant accident.

12. Summary

Accurate, physics-based computer simulations are widely used in all branches of science and engineering. When a new airplane turbine is designed, physics-based computer simulations are used to predict the performance of the finished part and to make changes before the aerospace company manufactures a prototype. In addition to being important in the technical disciplines, physical models are also used in the entertainment field, i.e., film industry, video games, and computer games. Physics-based computer simulations are the drivers behind special effects depicting realistic fire, water, air, hair, and moving objects. Simulation software for these special effects can be quite complex. Very often, simulation projects involve the modification of existing software to produce new capabilities. Furthermore, program or project goals change frequently as funding priorities evolve or as research developments lead to new branches of investigation. As such, code development is often the bottleneck on these projects. This bottleneck becomes a substantial incentive for software development packages that are developer-friendly—easy for scientific and engineering software developers to modify and extend their code.

CartaBlanca is the first scientific software package to employ the full power of Java, a language that continuously derives benefits from the extensive community of developers and supporters. CartaBlanca is a modern flexible software package for large-scale computational method and physical model development. CartaBlanca allows scientific and engineering code developers to produce and maintain less costly code that is not only user-friendly, but also developer-friendly.

Java is a programming language that addresses deficiencies in the C++ object-oriented language. Because of features such as type safety, array-bounds checking, and simplified object-oriented syntax, Java is a more efficient programming language. According to G. Phipps in “Comparing Observed Bug and Productivity Rates for Java and C++” (*Software: Practice and Experience*, 1999; 29:345–348), Java typically has two to three times fewer bugs and 30%–200% more code development productivity than C++. Since its introduction in the mid-90s, Java’s superiority in terms of security, portability, and code-developer productivity has led to a rapid growth in the use of Java as the programming language of choice within the business community. So far, scientists and engineers have been reluctant to abandon old languages such as FORTRAN and C++ because of the small number of successful Java scientific

or engineering applications. CartaBlanca is the platform for future scientific code developers.

CartaBlanca contains advanced nonlinear solver technology through the Jacobian-free Newton-Krylov scheme. At the same time, CartaBlanca's GUI capabilities lead to an extremely user-friendly problem specification. CartaBlanca has a novel design and uses object-oriented Java to provide a component-like computer architecture for the solvers and for the physics and material response modules.

In summary, CartaBlanca represents a paradigm shift in scientific computing and takes advantage of the wealth of capabilities built into the Java programming language. In addition, CartaBlanca incorporates advanced numerical algorithms, such as the Newton-Krylov solution method and the multiphase particle-in-cell treatment, that allow scientists and engineers to simulate complex phenomena such as fluid/structure interactions and reactive flows with phase changes.

A DVD with a short clip showing CartaBlanca's capabilities accompanies this entry. The clip shows CartaBlanca's portability from PCs and Macs to parallel-processing supercomputers.

Organization Data

13. Chief executive officer

Name G. Peter Nanos, Jr.
Position Director
Organization Los Alamos National Laboratory
Address Mail Stop A100
City, State, ZIP Los Alamos, NM 87545
Country USA
Phone (505) 667-5101
Fax (505) 665-2679
E-mail nanos@lanl.gov

14. Contact person to handle all arrangements on exhibits, banquet, and publicity

Name Cindy Boone
Position R&D 100 Coordinator
Organization Los Alamos National Laboratory
Address Mail Stop C333
City, State, ZIP Los Alamos, NM 87545
Country USA
Phone (505) 667-1229
Fax (505) 665-3125
E-mail boone@lanl.gov

15. To whom should reader inquiries about your product be directed?

Name W. Brian VanderHeyden
Position Technical Staff Member
Organization Los Alamos National Laboratory
Address Mail Stop B216
City, State, ZIP Los Alamos, NM 87545
Country USA
Phone (505) 667-9099
Fax (505) 665-5926
E-mail wbv@lanl.gov

Appendix

List of Co-developers

Letters of Recommendation

CartaBlanca Structure and Capabilities

 CartaBlanca—Java-Based Solver Environment

 CartaBlanca's File Structure

 CartaBlanca Graphical User Interfaces

 CartaBlanca's Test Interfaces

 CartaBlanca Simulations

CartaBlanca—A Pure-Java, Component-based Systems Simulation
Tool for Coupled Nonlinear Physics on Unstructured Grids (excerpt)

Implementation and Performance of a Particle in Cell Code Written
in Java (excerpt)

Los Alamos Profile

A DVD is included in this entry.

List of Co-developers

Name Nely T. Padial-Collins
Position Technical Staff Member
Organization Los Alamos National Laboratory
Phone (505) 665-0931
Fax (505) 665-5926
E-mail nelylanl@lanl.gov

Name Duan Z. Zhang
Position Technical Staff Member
Organization Los Alamos National Laboratory
Phone (505) 665-4428
Fax (505) 665-5926
E-mail dzhang@lanl.gov

Name Qisu Zou
Position Technical Staff Member
Organization Los Alamos National Laboratory
Phone (505) 664-0109
Fax (505) 665-5926
E-mail qisu@lanl.gov

Name Giovanni M. Lapenta
Position Technical Staff Member
Organization Los Alamos National Laboratory
Phone (505) 667-4394
Fax (505) 665-7150
E-mail lapenta@lanl.gov

Name Stefano Markidis
Position Postdoctoral Staff Member
Organization Los Alamos National Laboratory
Phone (505) 665-7594
Fax (505) 665-7150
E-mail stefano@lanl.gov



2790 Skypark Drive, Suite 310 • Torrance, CA 90505-5345 • (310) 530-1008 • (310) 530-8383 Fax

28 January 2005

J0548WW012805

Dr. Brian VanderHeyden
Group Leader
Theoretical Division Fluid Dynamics Group
Mail Stop B216
Los Alamos National Laboratory
Los Alamos, NM 87545

Subject: Support for "CartaBlanca" – A high efficiency complex physics coding environment written in the Java programming language

Dear Dr. VanderHeyden and Members of the R&D 100 Award Committee:

ACTA is pleased to write this letter in support of your proposed research. We sought to team with Los Alamos National Laboratory on our Army Research Office (ARO) sponsored Small Business Administration (SBA) Small Business Technology Transfer (STTR) Project entitled "Advanced Computational Algorithms for Simulating Weapon-Target Interaction," because of LANL's reputation in this research area and the particular advantages of CartaBlanca for simulating material penetration problems involving Army munitions. We used the code in our Phase I feasibility study and were awarded a 24 month Phase II contract based on the success of our cooperative research effort.

We are particularly interested in the capability of CartaBlanca to simulate the penetration of structural materials as well as armor plate with exploding munitions. The multi-phase flow capability offered by CartaBlanca was amply demonstrated in Phase I where we simulated the penetration and detonation of an explosive material by an inert round.

In Phase II, we plan to validate material models in CartaBlanca using test data provided by the Army. Model validation and predictive accuracy assessment are among ACTA's specialties. We are currently under contract with the AFRL Munitions Directorate at Eglin AFB to develop high-fidelity physics-based (HFPB) fast-running models for weapon-target interaction involving Air Force munitions, and plan to pursue similar modeling efforts involving Army munitions, using validated material models in CartaBlanca.

Some other applications of CartaBlanca we plan to pursue during our Phase III (commercialization phase) include (a) MOUT (Military Operations in Urban Terrain) related problems such as warhead penetration of urban materials, secondary debris generation and their

Los Alamos National Laboratory

Page 2

January 28, 2005

effect on people nearby, (b) Breakup of space boosters, missiles and thermal ablation of resulting debris during their travel back to earth, and (c) modeling explosions of land mines and their effect on vehicles and occupants of affected vehicles.

CartaBlanca is written in java and therefore can be easily ported to many types of computers and operating systems. ACTA has both PC workstations and two Beowulf clusters running Linux. CartaBlanca's ability to run parallel in the Beowulf cluster is very useful for us when we plan to run large problems. CartaBlanca has the capability to automatically verify new implementations and changes to the code using JUnit. This is very important to us since this is a live program that gets updated frequently. CartaBlanca has a component design architecture that allows us to add constitutive models for new materials easily. We plan to port a complex concrete material model used in Dyna-3D to CartaBlanca next year.

We appreciate the collaborative relationship we have enjoyed throughout Phase I and look forward to the same in Phase II. We wish you well as you seek additional research funding for this very worthwhile endeavor.

Sincerely,

A handwritten signature in blue ink, appearing to read "T. Hasselman", with a long horizontal flourish extending to the right.

Timothy K. Hasselman, Ph. D.
Director, Engineering Mechanics Division



BP Amoco Chemicals Company
PTA Americas Business Unit
Research & Technology Department
150 W. Warrenville Road
Naperville, Illinois 60563-8460

Phone: 630-420-4658

Fax: 630-961-8265

January 27, 2005

Subject: Support for "CartaBlanca -- A high efficiency, object-oriented, general-purpose computer simulation environment" from the Theoretical Division of Los Alamos National Laboratory.

Dear Dr. VanderHeyden and the CartaBlanca Team:

I am writing to endorse your application for the 2005 R&D 100 Award Submission for "CartaBlanca -- A high efficiency, object-oriented, general-purpose computer simulation environment".

I have been involved with computer simulation and modeling in the chemical industry for several decades. It has been my pleasure to see the three dimensional high fidelity modeling from DOE National Laboratories such as Los Alamos makes its way from the national defense mission into helping with civilian R&D.

This type of simulation capability has already helped my company optimize several of our complex reactive flow unit operations and has resulted in substantial savings. Furthermore, I am glad to see you taking the utility of this technology to the next level with CartaBlanca.

CartaBlanca's use of Java will vastly improve portability and your object-oriented component-like design will help make this complex technology more adaptive and useful to a wide variety of needs. These features should help the petrochemical industry use advanced simulation much more effectively in the future.

It is for these reasons that I offer my support and endorsement of the CartaBlanca project for the R&D 100 award. Good luck!

Sincerely,

A handwritten signature in cursive script that reads "Christos G. Papadopoulos".

Christos G. Papadopoulos
Modeling & Simulation Advisor
Leader M&S Network Americas
BP Amoco Petrochemicals

Bayer HealthCare
Diagnostics Division



January 28, 2005

Subject: Support for Los Alamos National Laboratory's High Efficiency, Object-Oriented, Computer Simulation Software – CartaBlanca.

Dear Dr. VanderHeyden and the CartaBlanca Team:

The purpose of this letter is to endorse your application for the 2005 R&D 100 Award submission for the High Efficiency, Object-Oriented, Computer Simulation Software – CartaBlanca.

As a Senior R&D Engineer in the medical instrument development industry, I have been looking for a high efficiency and reusable computer simulation software for flow simulation so that we can: 1) better understand our medical system design (liquid delivery, mixing... etc.); 2) reduce development cycle time; 3) reduce total cost of our systems. These will allow us to develop more cost effective and better performing diagnostics healthcare solutions.

The commercial flow simulation tools on the market currently are very expensive and not efficient. In April, 2004 when I first came across CartaBlanca's website, I was very impressed by its capability, that is, Java Object-Oriented based and with rich capability in flow simulation.

In the industrial R&D settings, there are tens, if not hundreds of networked computers running either Microsoft Windows, Linux, or UNIX. During off hours, these computers usually go to idle state. With the Java language, CartaBlanca will be capable of solving the same flow model in parallel by utilizing these idle networked computers, regardless of operation system, during the night time and deliver results the next morning.

The Object-Oriented natural of CartaBlanca will also allow reuse of classes easily, hence, easy maintenance/development of new solvers.

With its unique feature in Java, Object-Oriented programming, and rich flow simulation capabilities, I will recommend CartaBlanca to anyone who performs complex flow simulation.

Sincerely,

A handwritten signature in black ink, appearing to read "Pei-Ying Hsieh".

Pei-Ying Hsieh, Ph.D.
Senior R&D Engineer
Bayer HealthCare
Diagnostics Division

Bayer HealthCare LLC
Diagnostics Division
511 Benedict Avenue
Tarrytown, NY 10591

Phone: 914-631-8000



Thomas J. Watson Research Center
P. O. Box 218
Yorktown, NY 10598

February 2, 2005

Subject: Support for "CartaBlanca -- A high efficiency, object-oriented, general-purpose computer simulation environment" from the Theoretical Division of Los Alamos National Laboratory.

Dear Dr. VanderHeyden and the CartaBlanca Team:

This letter serves to endorse your application for the 2005 R&D 100 Award Submission for "CartaBlanca -- A high efficiency, object-oriented, general-purpose computer simulation environment."

The Java programming language has been a spectacular success in web and business computing. Java's type safety, portability and efficient language structure have been the keys to this success. From a software engineering perspective, the Java programming language also provides an attractive platform for writing numerically intensive applications. Features such as built-in threads and components for graphical user interfaces along with true object-oriented programming capabilities make the language very attractive for 21st century scientific applications. IBM has been at the forefront of enabling the use of Java for numerically intensive computing. Our efforts have included the high performance Ninja compiler and research on array-bounds-check elimination and semantic inlining to name a few. To fully realize the goal of widespread use of Java scientific computing, we believe it crucial to have examples of practical applications that can serve as both a test bed and a pilot for the greater scientific and engineering community. CartaBlanca has served this role. It has demonstrated the feasibility of using Java along with its advanced features to produce a state-of-the-art scientific computing environment. We expect that the continued success of the CartaBlanca code project will encourage others to explore the use of Java for a wide variety of numerically intensive applications. This will foster a synergy between scientific and business computing that will yield great benefits for all. For these reasons, we unreservedly endorse the CartaBlanca project for an R&D 100 award.

Best Regards,

Manish Gupta

Manish Gupta

Senior Manager, Emerging System Software
IBM T. J. Watson Research Center

W. B. VanderHeyden
Group Leader
Theoretical Division Fluid Dynamics Group
TA-3, Bldg 200, Mail Stop B216
Drop Point 03020001S
SM-30, Bikini Atoll Road
Los Alamos National Laboratory
Los Alamos, NM 87545



pervasivetechlabs
AT INDIANA UNIVERSITY

January 24, 2005

Subject: Support for “CartaBlanca -- A high efficiency, object-oriented, general-purpose computer simulation environment” from the Theoretical Division of Los Alamos National Laboratory.

Dear Dr. VanderHeyden and the CartaBlanca Team:

This letter serves to enthusiastically endorse your application for the 2005 R&D 100 Award Submission for “CartaBlanca -- A high efficiency, object-oriented, general-purpose computer simulation environment.”

As you know, I have been dedicated to the cause of using Java for scientific computing applications. Just as the Java programming language has revolutionized web, business and telecommunications software, I believe that it also can provide the same benefits to scientific software. As Phipps’s study has shown (G. Phipps, *Software Pract. Exper.*, **29**, 345 (1999)), programming applications in Java in for business applications has led to substantially increased software developer productivity with a concomitant decrease in bug rates. It is for these reasons that I have served as the Chairman of the international Java Grande Forum (<http://www.javagrande.org/>), which is dedicated to the promotion of Java for high performance and scientific computing.

I am pleased to endorse the CartaBlanca project because it has been in the vanguard of Java Grande Projects. It has played a crucial role showing that real-world scientific computing Java applications are not only viable but also quite beneficial. Its complexity—unstructured 3 dimensional computational grids, multiphase fluid flow, fluid-structure interaction—has been particularly pleasing to see and to showcase in my review articles (G. Fox, *Computing Sci. Engng.*, **5**, 60 (2003)).

Furthermore, the seamless incorporation of the industry standard Junit off-the-shelf testing software into CartaBlanca furthers the cause of software re-use so crucial to the vision of vastly increased productivity that is behind much of the software engineering efforts today. The fact that CartaBlanca uses both single and multi-processor operational mode for a wide variety of applications is impressive. The re-use of the JavaParty environment from the University of Karlsruhe for porting from shared-memory-parallel to distributed cluster computing environments is another important example of software re-use and the power and portability of the Java programming language.

501 North Morton Street, Suite 224
Bloomington, Indiana 47404-3730
812-856-1242 Fax 812-856-1537

The use of object-oriented inheritance hierarchies in CartaBlanca to produce component-like, plug and play modules for alternative physics treatment and numerical solver strategies is another example of the novelty and power of the Java programming language. The relatively small size of the CartaBlanca team, when compared to more conventional software development projects, is good evidence of the productivity of the Java language. Analysis by Los Alamos documented the need for large software teams in the conventional Fortran/C++ environment.

Coupling these achievements discussed above, with your important collaborative work with Lapenta, Markidis and Budlimic on the Parsek test code, which gave crucially important side-by-side speed comparisons of Java, Fortran and C++, your work has gone a long way towards the realization of the Java Grande vision. Continued development and commercialization of the CartaBlanca code will help usher in a new era for scientific computing. It is for these reasons I give my unqualified support to your nomination for the R&D 100 Award.

Sincerely,

A handwritten signature in blue ink, appearing to read "Geoffrey C. Fox", with a stylized flourish extending to the right.

Geoffrey C. Fox

Professor of Computer Science, Informatics & Physics
Chairman, Java Grande Forum
Director, Community Grids Lab,
Pervasive Technology Labs at Indiana University



CENTER FOR HIGH PERFORMANCE SOFTWARE RESEARCH

KEN KENNEDY, DIRECTOR
JOHN AND ANN DOERR UNIVERSITY PROFESSOR

Via email: wbv@lanl.gov

January 31, 2005

W. B. VanderHeyden
Group Leader
Theoretical Division Fluid Dynamics Group
TA-3, Building 200, Mail Stop B216
Drop Point 03020001S
SM-30, Bikini Atoll Road
Los Alamos National Laboratory
Los Alamos, NM 87545

Dear Dr. VanderHeyden and the CartaBlanca Team:

It is my pleasure to endorse your application for the 2005 R&D 100 Award Submission for “CartaBlanca—A high efficiency, object-oriented, general-purpose computer simulation environment”.

Let me begin by summarizing my background. My research here at Rice University is exploring software support for high performance and parallel computing in science and engineering, scientific programming environments, and optimization of compiled code. My current work falls into four main project areas: implementation of efficient high-level domain-specific programming systems, application development tools for computational grids, research on compilers and tools for scalable scientific computing and compilation for high-performance uniprocessors. I am the Director for The Center for High Performance Software Research, which is a research development center that specializes in leading and managing cross-institutional, multidisciplinary research consortia, primarily in software for high performance computing. In addition, I currently direct or co-direct four ongoing multi-institutional consortia: the Los Alamos Computer Science Institute, the NSF-sponsored Virtual Grid Application Development System project, the

Gulf Coast Center for Computational Cancer Research and the Houston BioGrid consortium.

We at Rice have been closely following your progress on the CartaBlanca (and the associated Parsek) software project for several years now, as we regard it as one of the most important ongoing efforts to bring the power of object-oriented computing into the somewhat archaic world of high-performance scientific computing. We believe that modern software should exhibit the following characteristics: elegant design, ease of maintenance, extensibility, portability and last but not least, high performance. The software created by the CartaBlanca project shows all of these qualities, while further using Java as the implementation language, which has frequently been dismissed as too slow for high performance applications.

As you know, Zoran Budimlic and I have been pursuing the use of Java for high performance and scientific computing for some time now. Java offers increased programmer efficiency and language type-safety; and Java is widely used for web and business applications. In addition, Java follows the open-source trends that have yielded tremendous benefits in the form of software re-use.

At the same time, Java has not yet realized its full potential in the high performance and scientific computing arena. This is due not only to execution speed issues associated with earlier versions of the Java virtual machine and to compiler optimization problems with object-oriented code, but also to a lack of example success stories. We at Rice have been addressing the former problem by introducing our advanced telescoping Java compiler framework JaMake, which uses a novel approach for the compilation of highly object-oriented software. Your CartaBlanca effort has gone a long way towards addressing the latter problem by providing a real-world multi-physics application written entirely in Java, which solves complex multidimensional physics application problems. In addition, your work with the Parsek team has provided useful benchmarks between Java and the more traditional languages for scientific computing, C++ and FORTRAN. Furthermore, the Parsek code allowed us to test our JaMake compiler for scientific applications and gain unique insights.

The CartaBlanca project has had an enormous impact on the scientific community and has come a long way in reshaping the old-fashioned beliefs that one must sacrifice good design for the sake of performance, and the not so old, but still obsolete, thinking that Java and other high-level languages are unsuitable for high-performance computing. Efforts like this will change the way programmers think about high-performance software and help boost the productivity and quality of software design that is inadequate in most of the scientific and high-performance systems so far.

My sincere hope is that CartaBlanca's success will encourage more scientific computing using your paradigms and lead to a new wave of more efficient scientific computing applications. I therefore heartily endorse your application for an R&D 100 award for the CartaBlanca project.

Sincerely,

A handwritten signature in black ink that reads "Ken Kennedy". The signature is written in a cursive, slightly slanted style.

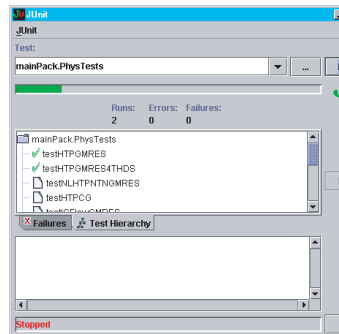
Ken Kennedy
John and Ann Doerr University Professor
Director, Center for High Performance Software Research

CartaBlanca – Java-based Solver Environment

- Newton-Krylov solver for implicit, non-linearly consistent solutions
- Multiphase flow
- Rapid prototyping
- PIC method
- GUI interface
- Integrated unit and regression testing

CartaBlanca

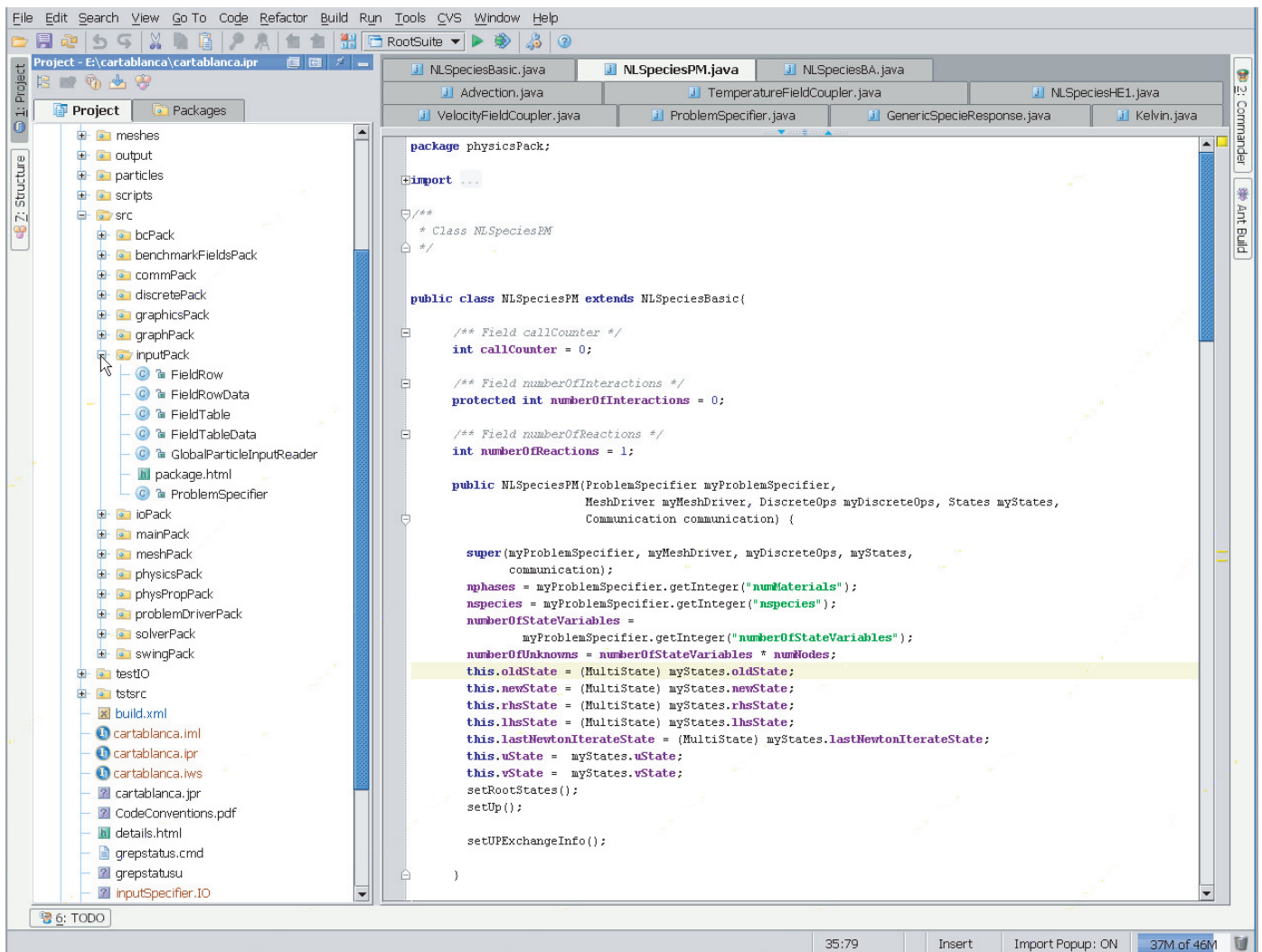
- Pure Java development environment
- Object-oriented design: developer productivity
- Phipps: Java 30-200% more productive
- Multiphase flow on 3-D unstructured grids
- JUnit for testing



Why Java?

- Simple, clean object-oriented language
- Strong typing, extensive compiler checking
- Commercially robust: good developer tools
- Portable, robust
- Built-in thread facility, networking
- JavaGrande addressing HPC
- New fast JVMs, native compilers
- C/C++/FORTRAN interoperability
- Growing pool of developers, will soon be most widely used
- Built-in GUI (Swing), graphics, database access (JDBC)

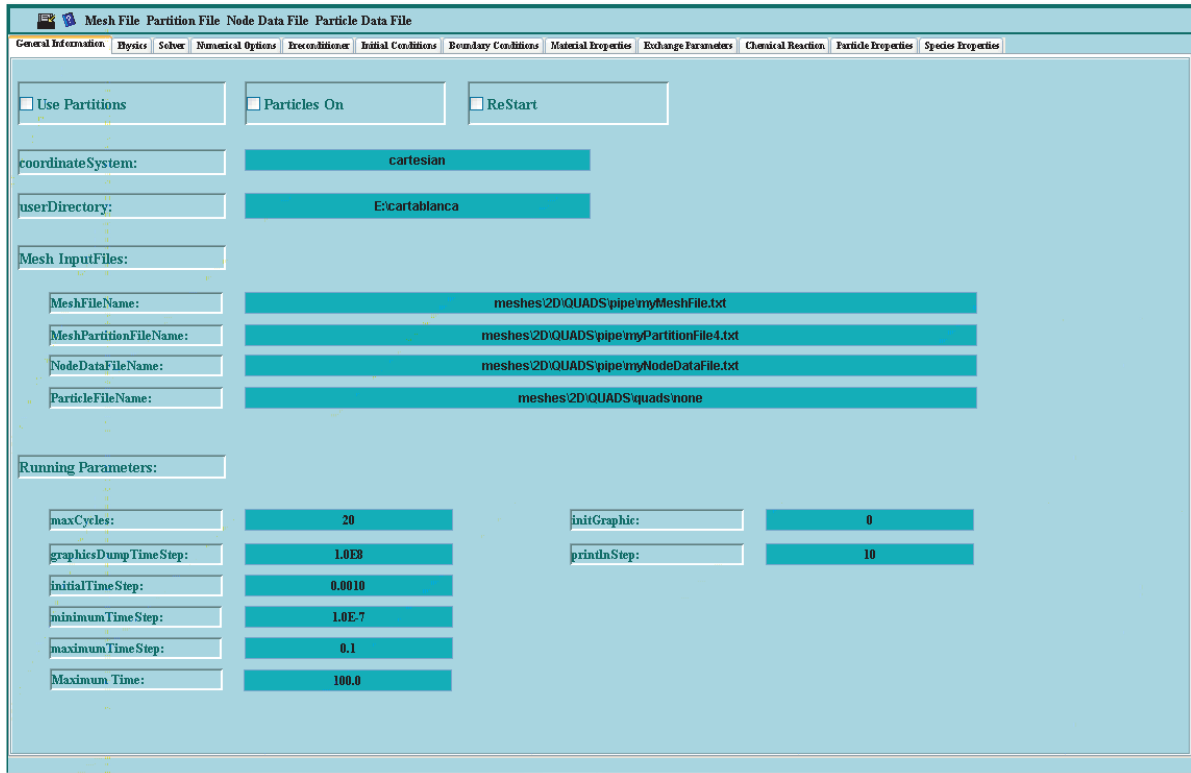
CartaBlanca's File Structure



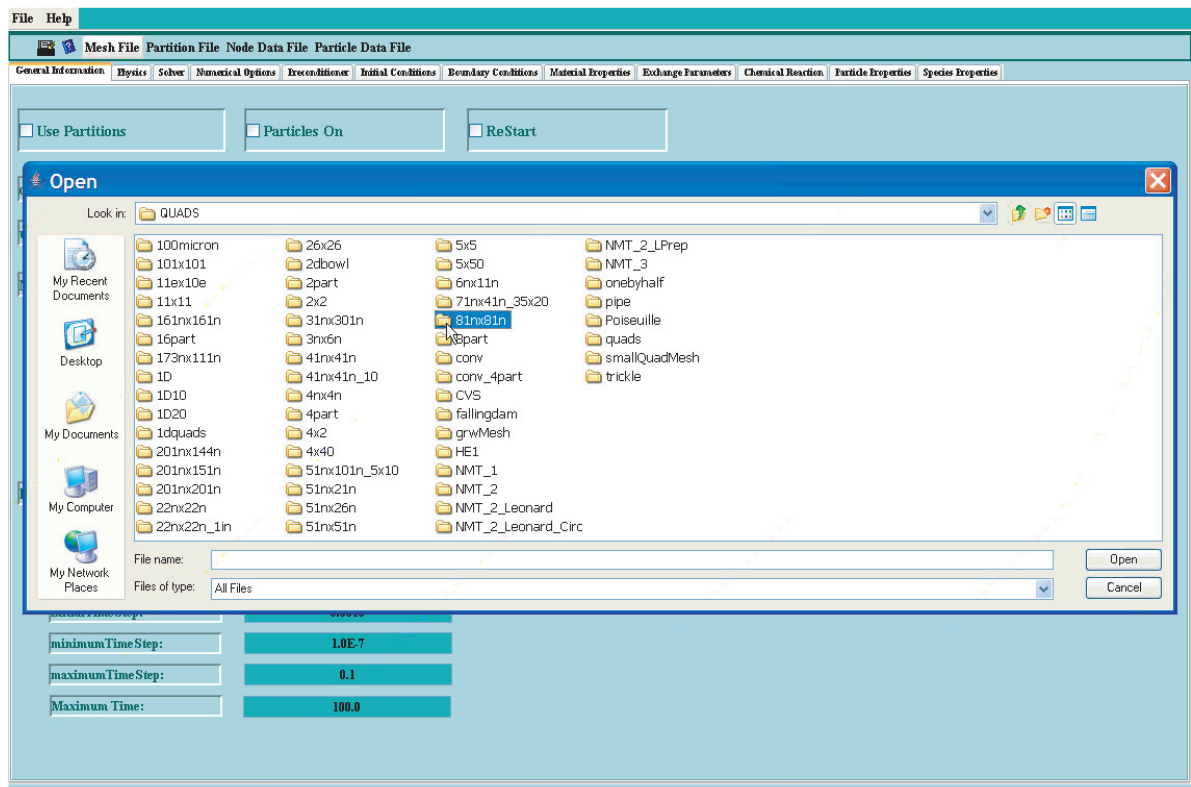
This figure shows an example of an edit session using Borland's JBuilder Java development environment to examine the CartaBlanca file structure. Because of the ubiquity of Java, the Foundation version of the Borland JBuilder environment shown here is freely available. This version is more than adequate for developing code with CartaBlanca. The same kind of capability for FORTRAN, for example, would cost several hundred dollars.

In the pane on the left, you see the directory structure of the CartaBlanca code. The user can easily interact with the objects using familiar point and click methods to drill down through the file hierarchy and to navigate in general. On the right, you see a part of the code for a species transport class in CartaBlanca. The color coding is automatic and helps the developer quickly identify the different types of variables and Java language components. JBuilder enables point and click interaction with the code in this pane. This ability is state-of-the-art and free because of Java's strong marketplace presence. This free software is part of the reason for Java's efficient development protocol.

CartaBlanca Graphical User Interface

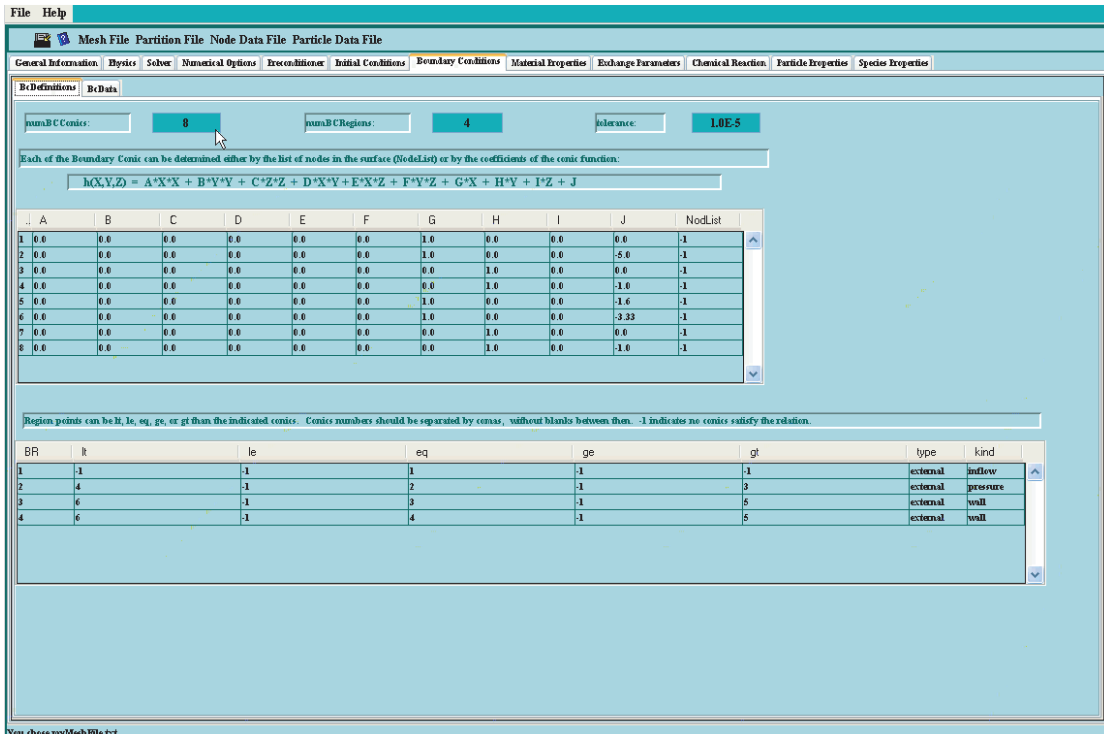


Main Page. Note multiple tabs along top to allow access to a variety of pages for specification of boundary conditions, material constants, etc.

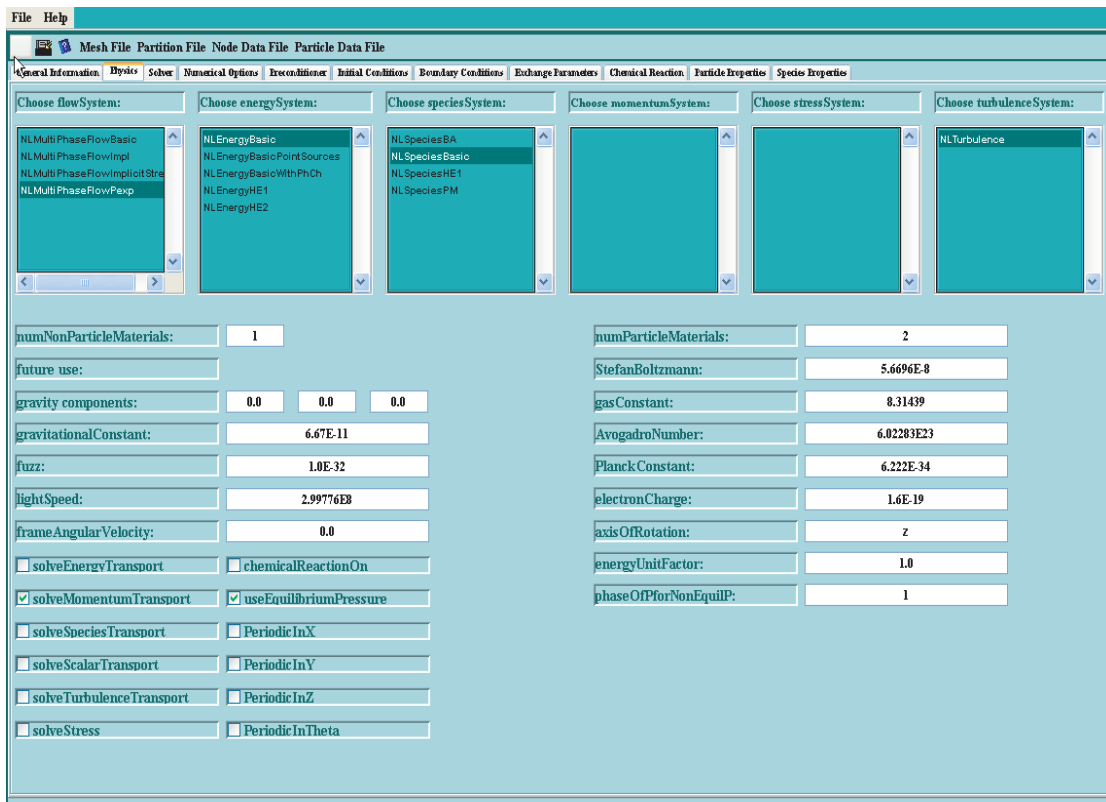


Here we see the interaction of the CartaBlanca GUI and the windows file system for point and click access to mesh files.

CartaBlanca Graphical User Interface



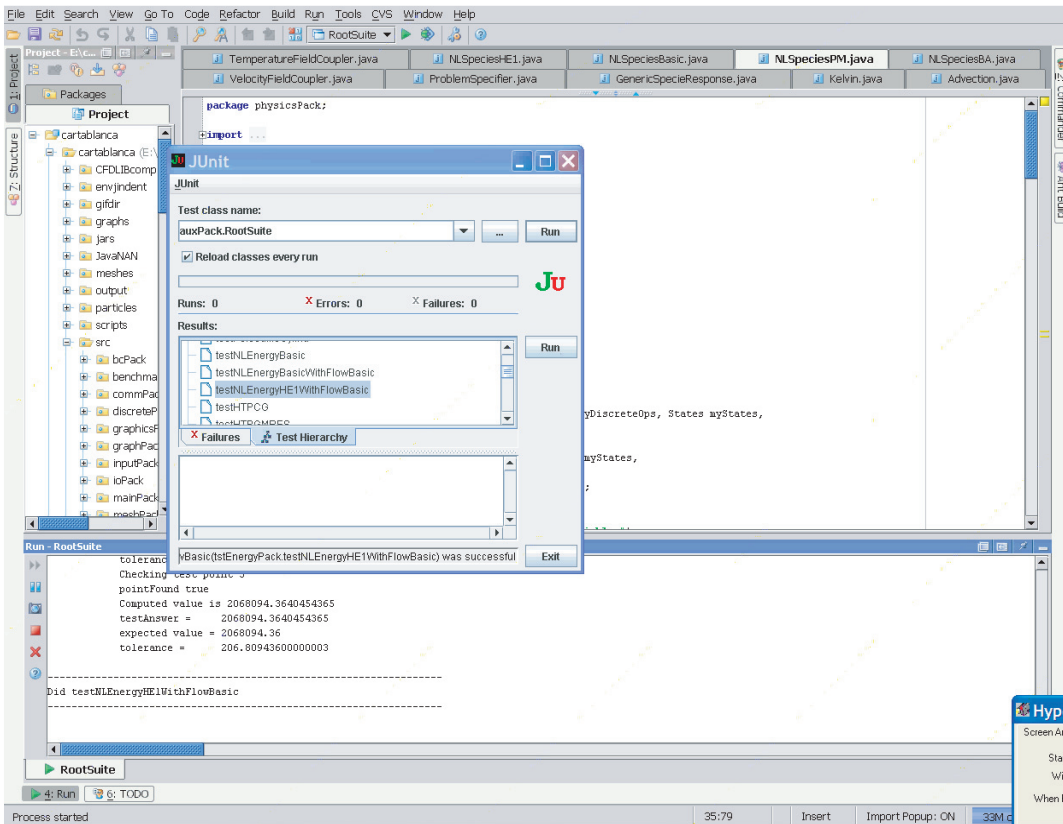
This figure is the boundary conditions page. In the top table, the user can prescribe general conic surfaces that serve as boundary sections for boundary conditions. In the second table, the user specifies which conic surfaces correspond to the boundary sections.



This figure is the general physics page. The bottom section contains constants and switches that control basic physics parameters. The top layer contains scroll panes that allow the user to select the type of flow, heat transfer, species transport, and other types of physics. If a developer adds a new instance of one type, Java's reflection mechanism will automatically pull it into the menu of choice.

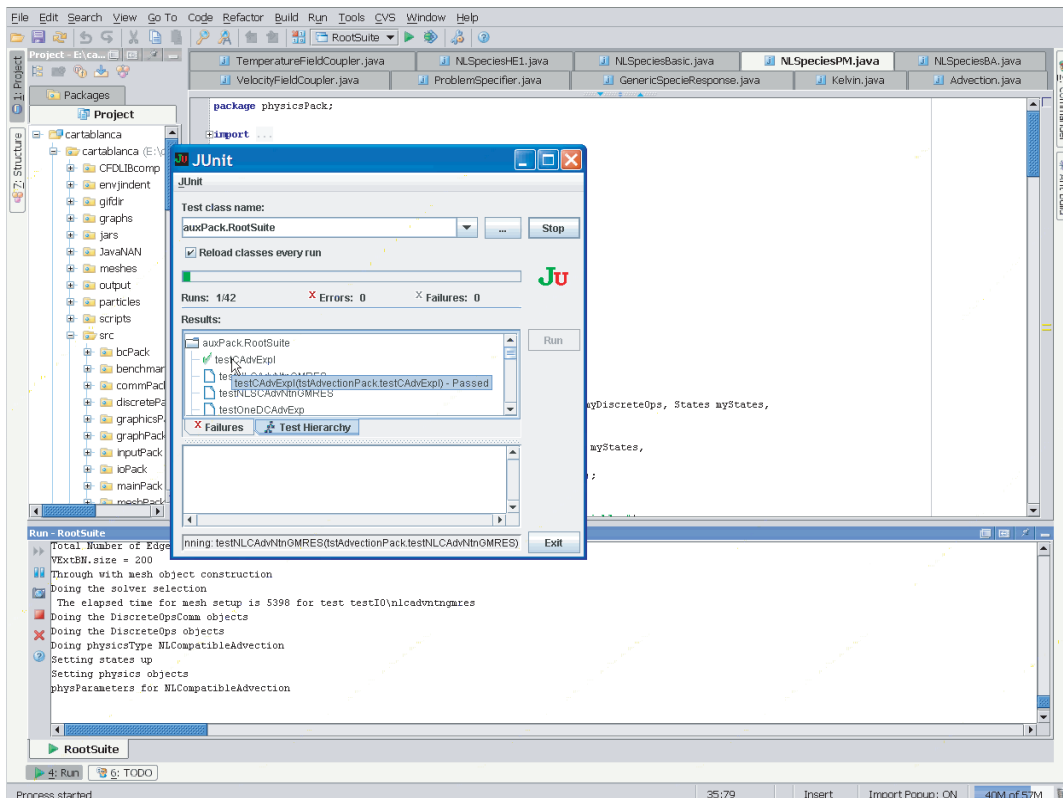
CartaBlanca's Test Interfaces

Single Test



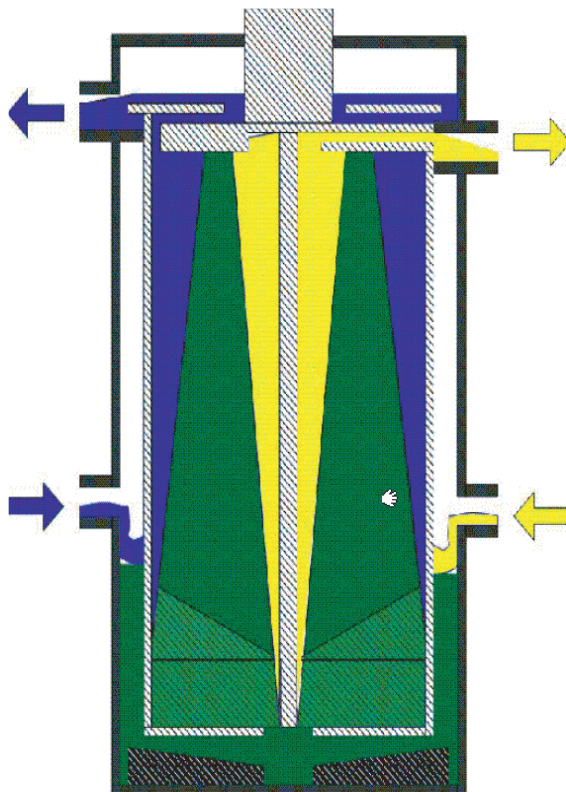
Here we see two examples showing the test suite for CartaBlanca. CartaBlanca uses the JUnit test package that drops seamlessly into CartaBlanca's framework. CartaBlanca's test suite allows the developer to make sure new modifications to the code have not broken previously installed capabilities. Nearly 50 tests are built into CartaBlanca to test many aspects of the code. These tests include solvers and advection algorithms, as well as tests of problems such as projectile target interactions. The tests are run either one by one, as shown in the single test, or automatically, as shown in the multiple test. The GUIs shown allow the developer to see the progress of the tests and to see where any failures occur. The tests are written to run quickly and to update automatically with any changes to the code. The use of JUnit along with the inherent efficiency of the Java programming language is one of the keys to the success of the CartaBlanca framework.

Multiple Test

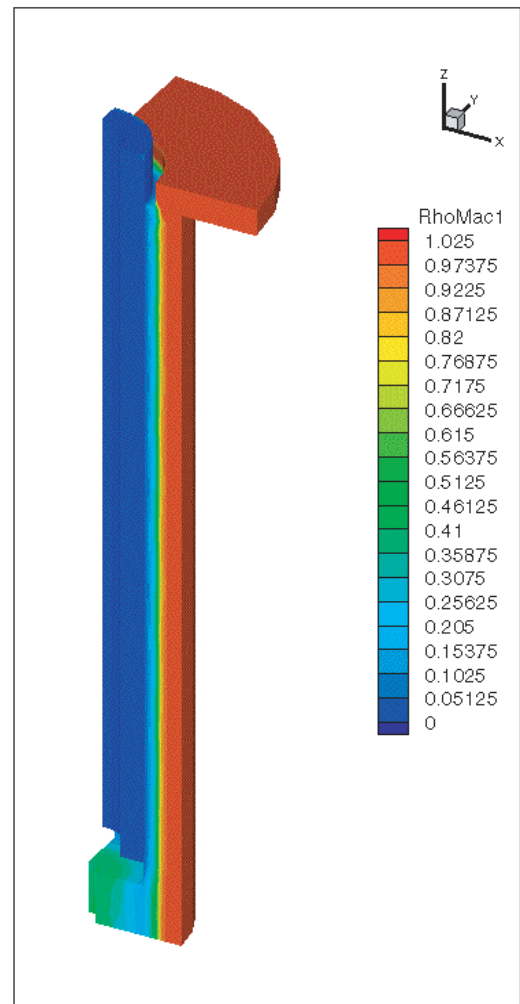


Phase Separation Simulation Using Annular Centrifugal Contactors

Annual centrifugal contactors consist of two concentric cylindrical zones. The spinning rotor and the stationary housing wall form the external zone, where some liquid or organic material are being mixed. After mixing, the flow mixture enters the inner rotating cylinder through an annular opening in the bottom. In this zone, the flows are separated by high centrifugal forces, and each liquid leaves the device through exit ports on top. CartaBlanca's multiphase-flow solver is used to simulate the hydraulics of the separation zone. The lighter fluid leaves through one small opening at the center of the outside wall of the central cylindrical region at the top. The heavier leaves through another small opening at the center of the outside wall of the external cylindrical region at the top.



Schematic of Centrifugal Contactor



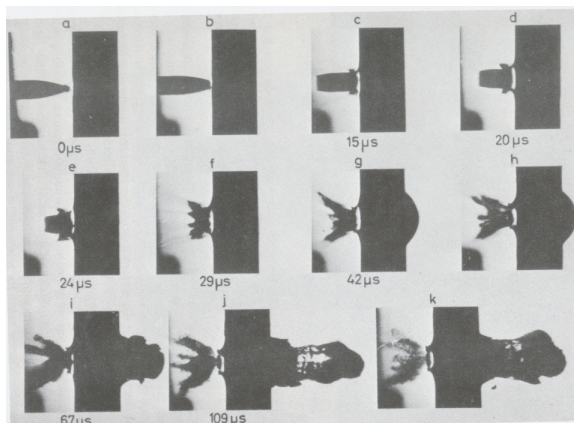
Quarter Section CartaBlanca Simulation of Contactor

Projectile/Target Simulation

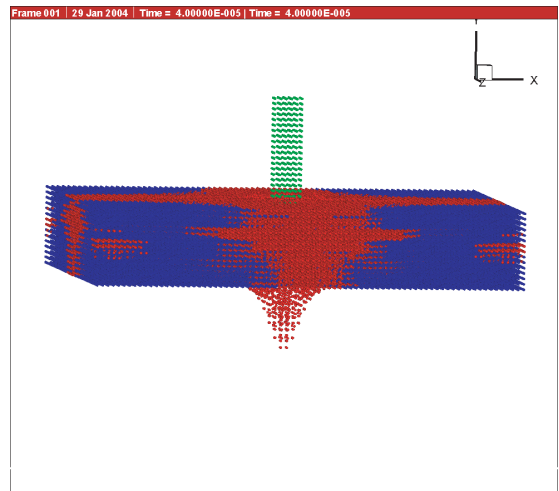
Simulations of projectile/target interactions are of interest to the armed forces. These interactions are challenging problems because of the high deformation and material damage that occurs, CartaBlanca addresses these problems by use of a combination of multiphase flow and the mesh-free particle method. It tracks the high deformation and damage without the mesh tangling experienced with conventional Lagrangian mesh code. CartaBlanca also handles the effects of gas evolution from a reactive armor problem using the same procedure.

3-D Bullet Brittle Plate

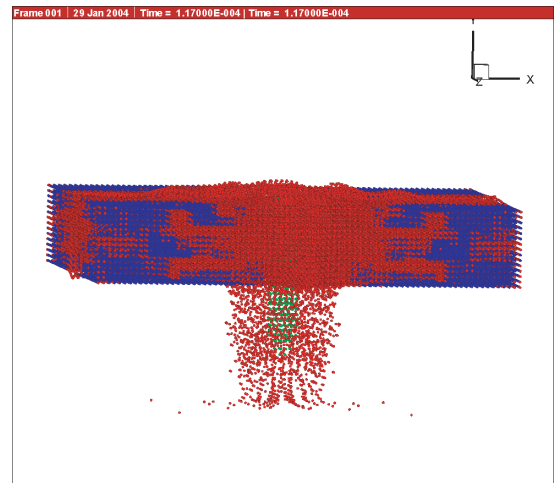
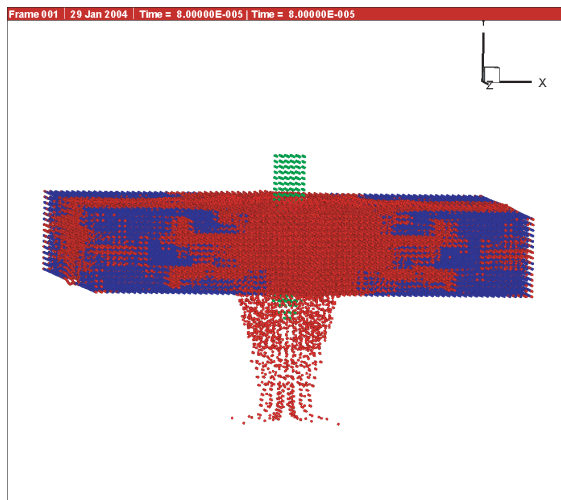
Awerbuch Experiment



3D – Brittle – 40 microseconds



3D – Brittle – 80 microseconds 3D – Brittle – 120 microseconds

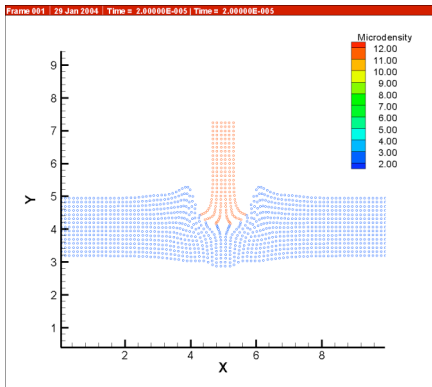


2-D and 3-D Bullet/Plate Interaction Simulation

This figure set shows a simple bullet/plate interaction calculation and compares the results in two- and three-dimensional simulations. The bullet is lead and the plate is aluminum. Both are represented as particles. The 3-D simulation shows some non-axisymmetric behavior resulting from the initial plate boundary conditions.

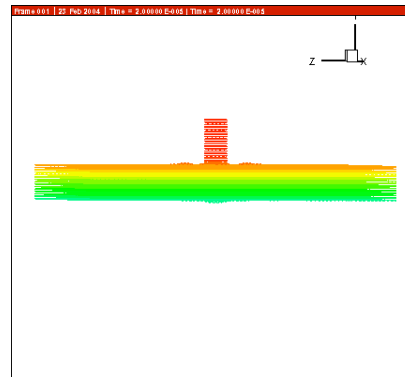
2-D Bullet/Plate Simulation
with Plastic Response

20 microseconds

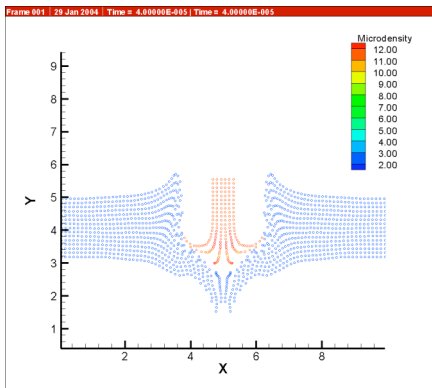


3-D Bullet/Plate Simulation
(Serial and Parallel)

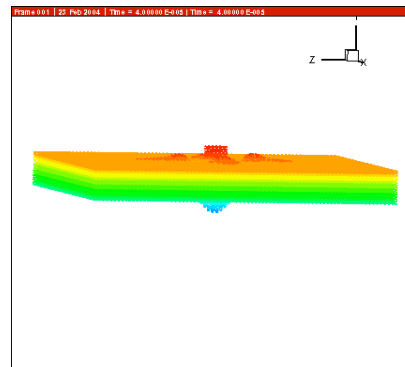
20 microseconds



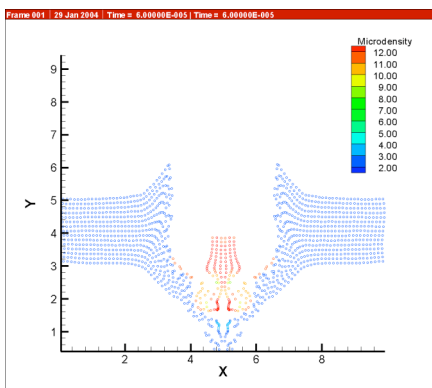
40 microseconds



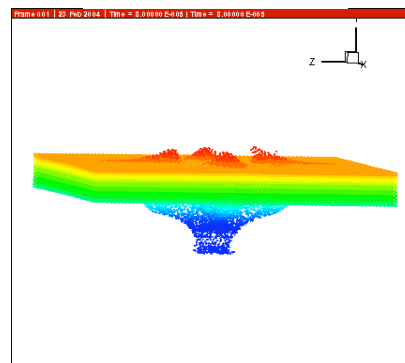
40 microseconds



60 microseconds



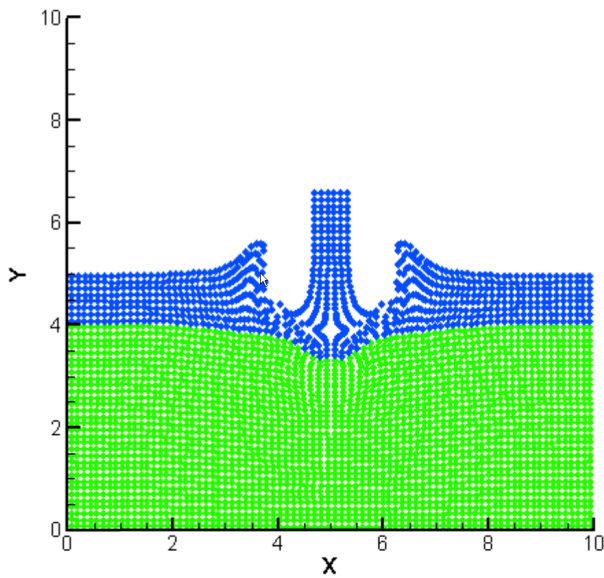
80 microseconds



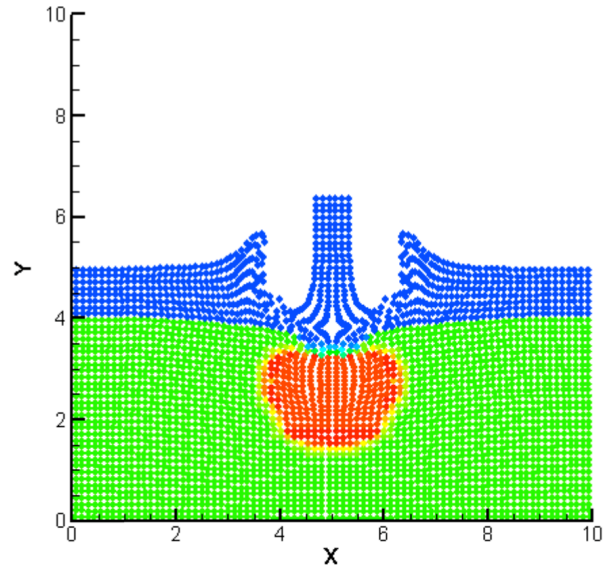
2-D 4-Phase Reactive Bullet Plate

Shown are the results from a 4-phase, two-dimensional calculation where a high explosive has been put at the bottom beneath the target plate. The calculation includes 1) lead bullet, 2) aluminum target, 3) gas, and 4) high explosive. The bullet penetrates the target plate and at 26 microseconds, the high explosive ignites. The orange color is the ignition front, which propagates through the material.

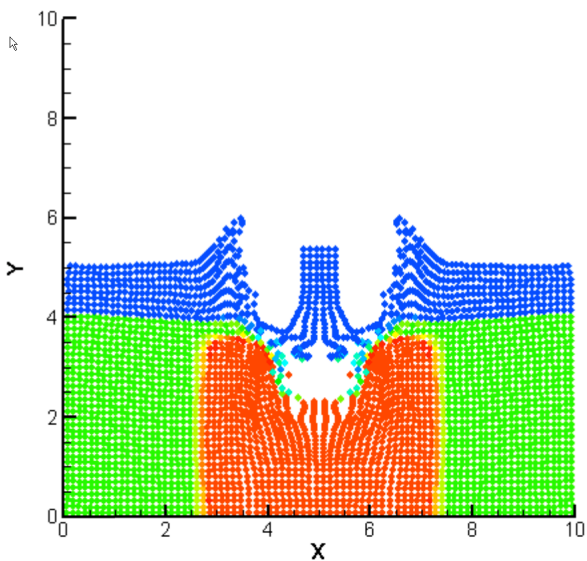
Reactive – 24 microseconds



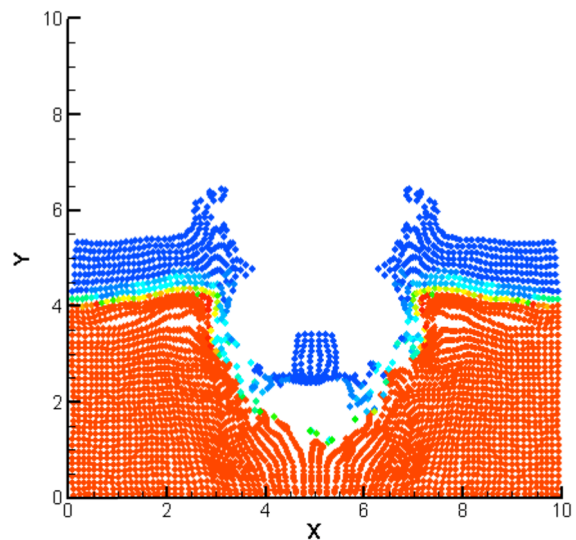
Reactive – 26 microseconds

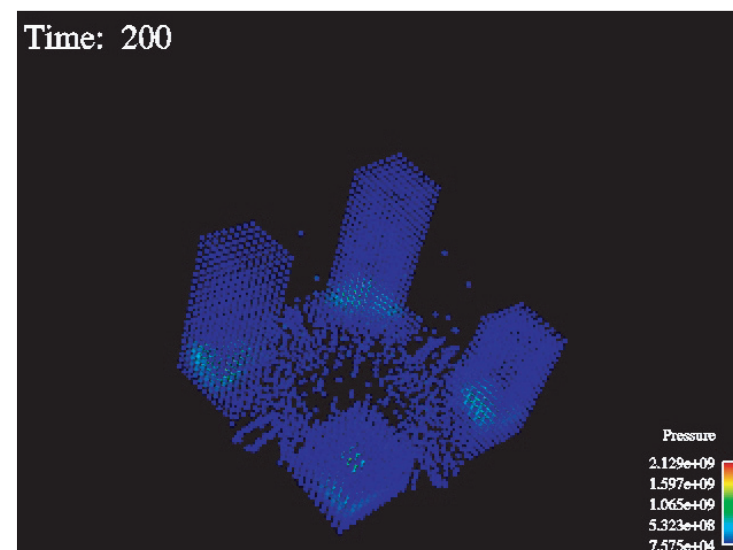
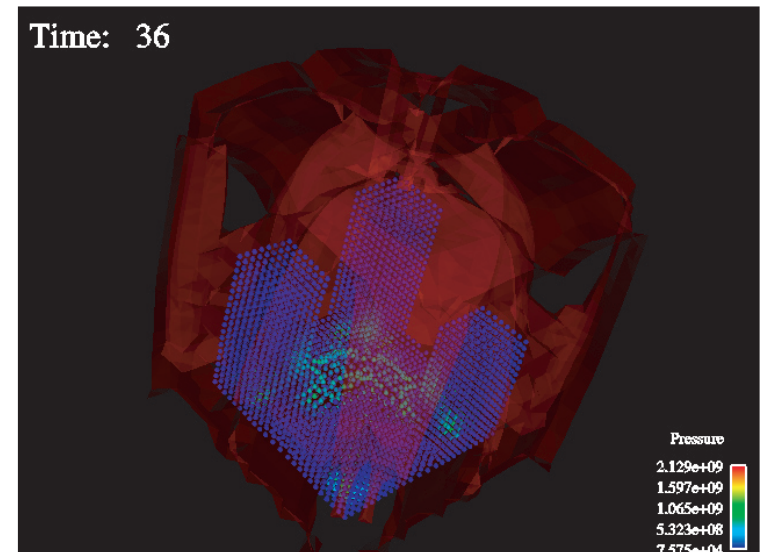
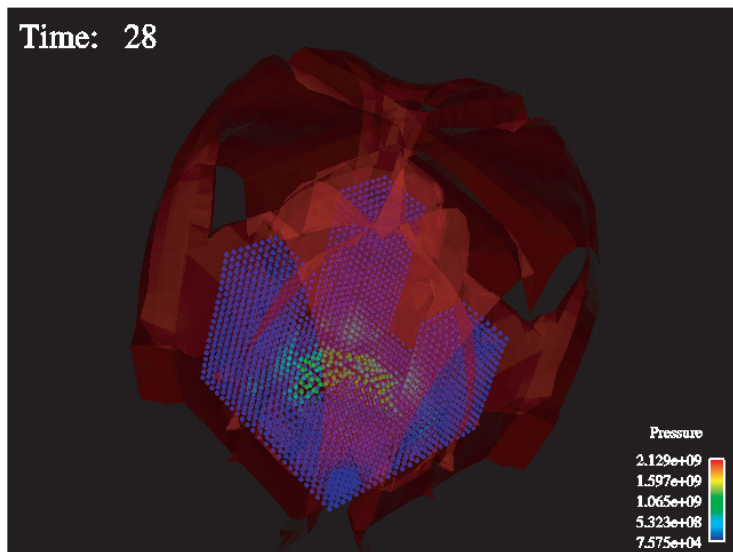
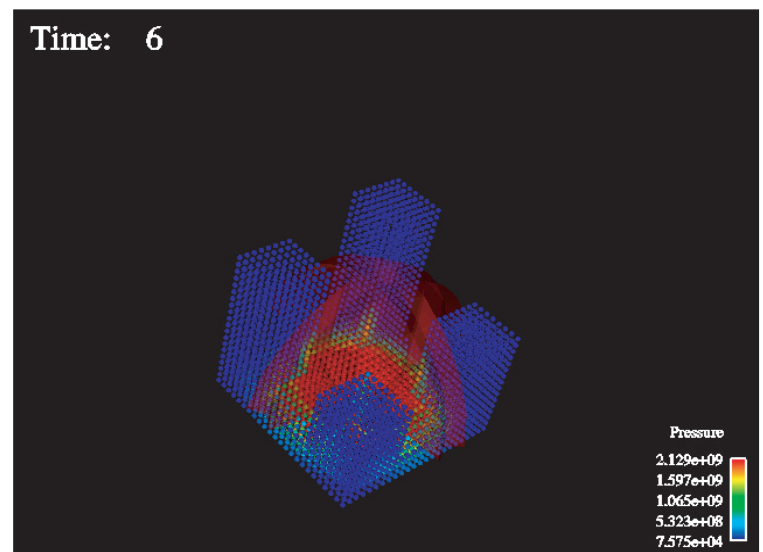
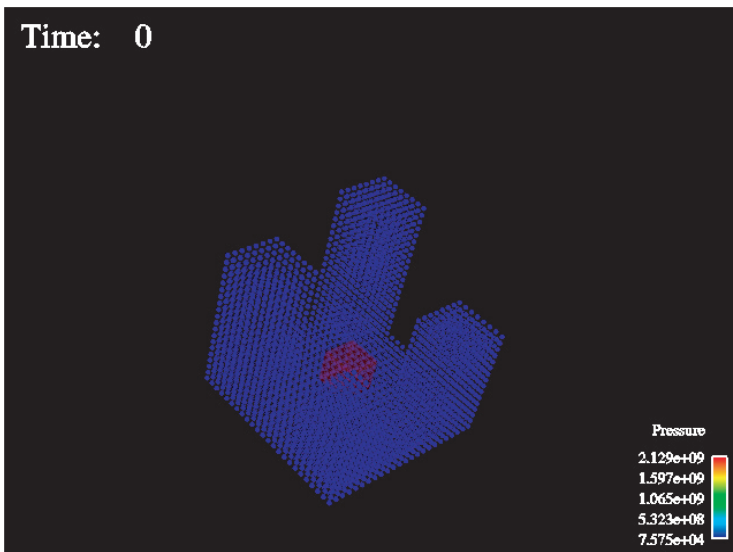


Reactive – 36 microseconds



Reactive – 56 microseconds





3D Multiphase-FLIP-MPM Simulation of Blast Between Buildings

- 100m cube
- 20m x 20m square buildings
- 20, 30, 40, 50 m tall
- ~3kt blast
- Elastic law with brittle fracture
- Multiphase flow

This sequence of snapshots from a CartaBlanca simulation shows the development of a blast and its effects on buildings in an urban setting. The application is analysis for homeland security. In the snapshots, the red isosurface shows the progression of the blast wave through the air. The blue particles make up the buildings and the concrete base. As the pressure wave interacts with the buildings, the material deforms and fails creating debris.

CartaBlanca— A Pure-Java, Component-based Systems Simulation Tool for Coupled Nonlinear Physics on Unstructured Grids—An Update

(Excerpt)

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

FINITE VOLUME METHOD

CartaBlanca is based on the finite-volume method, [9], for conservation equations. CartaBlanca adopts the node-based version of this scheme with edge-based connectivity. 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

$$\frac{d}{dt} \int_V q dV + \oint_A \vec{f} \cdot \vec{n} dS + \int_V s dV = 0, \quad (1)$$

where q is the density of some conserved quantity such as mass, momentum or energy, \vec{f} is the local flux of this conserved quantity due to a variety of mechanisms, \vec{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.

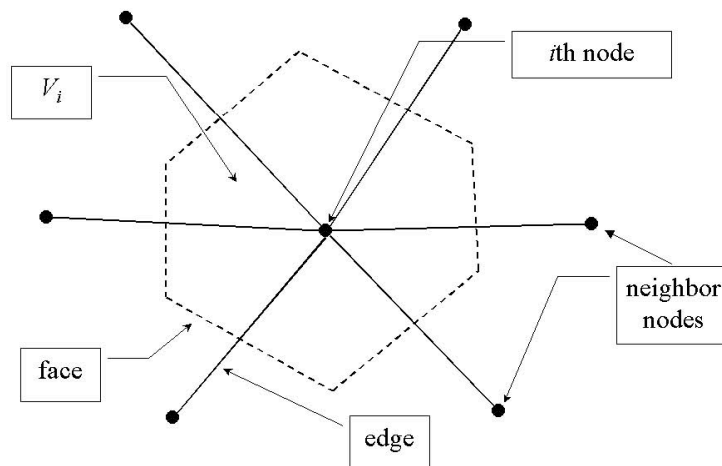


Fig. 1. Control volume for i th node.

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

$$q_i \equiv \frac{1}{V_i} \int_{V_i} q dV. \quad (2)$$

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

$$s_i \equiv \frac{1}{V_i} \int_{V_i} s dV. \quad (3)$$

Let \vec{f}_e be the average flux on the control volume face associated with edge e . Then, if we integrate the 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

$$q_i^{n+1} V_i^{n+1} - q_i^n V_i^n + \Delta t \left\{ \sum_{edges} \vec{f}_e \cdot \vec{n}_e A_e + s_i V_i \right\} = 0, \quad (4)$$

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 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.

JACOBIAN-FREE NEWTON-KRYLOV METHOD

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

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

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 Jacobian-Free Newton-Krylov method 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 parenthesis denotes the iterate level n . 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

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

along with the update

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

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

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

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. Furthermore, matrix-vector products can be approximated numerically using a directional difference formula,

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

where ε is some small scalar perturbation parameter. 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.

SOFTWARE

CartaBlanca is composed, at present, of twelve 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.

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.

Software Engineering

Our approach to team programming follows the lightweight processes advocated in the recent article by Fowler, [10]. 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, [15], testing facility into CartaBlanca. This has been useful in that any developer can perform tests easily on their local computing platform to make sure his 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 a 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.

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 S. J. Chapman, [5], for these operations, [28]. 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, [12]. We make use of the nonstandard

ExponentialFormat class provided in reference [12] for writing doubles to text files. At present, this is the only class contained in the I/O Package.

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 graphical user interface (GUI). Problem data is written to a 'ProblemSpecifier' class object. The 'ProblemSpecifier' object contains all input information from the GUI. It can be queried by as needed for information in the rest of the program. The 'ProblemSpecifier' object is serializable. This feature is 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.

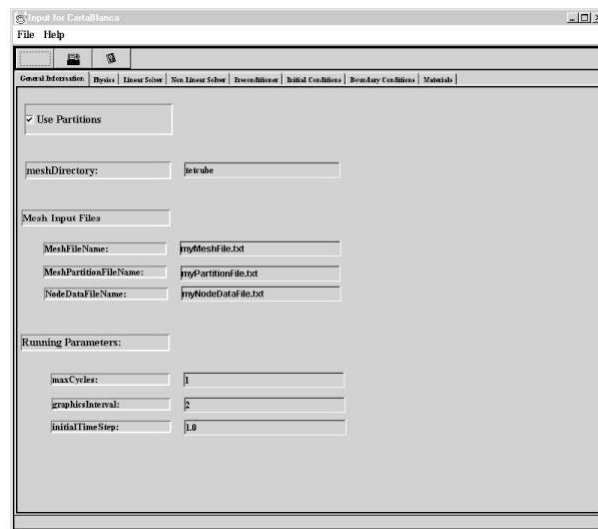


Figure 2. Snapshot of GUI. Tabs enable user to provide input on the various categories of input.

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 executing based on the information in the ProblemSpecifier object.

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 Chapter 5 of Oaks and Wong. The CyclicBarrier is used, for example, in discrete 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.

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 files, which follow the format used by the Metis mesh-partitioning program. The three files contain the mesh connectivity, the node coordinates and the partitioning of the mesh elements. Please see the Metis manual 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 the Figure, 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 the Figure, the three nodes along the partition boundary would be present in each partition as duplicates.

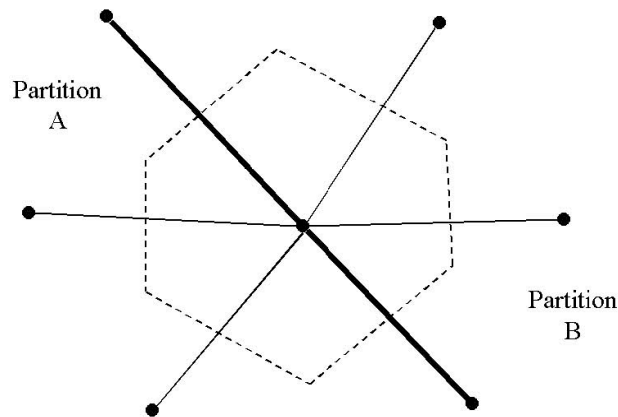


Figure 3. Partitioning in CartaBlanca. Meshes must be partitioned along node connections.

Figure 4 shows an example of an element-partitioned mesh for CartaBlanca.

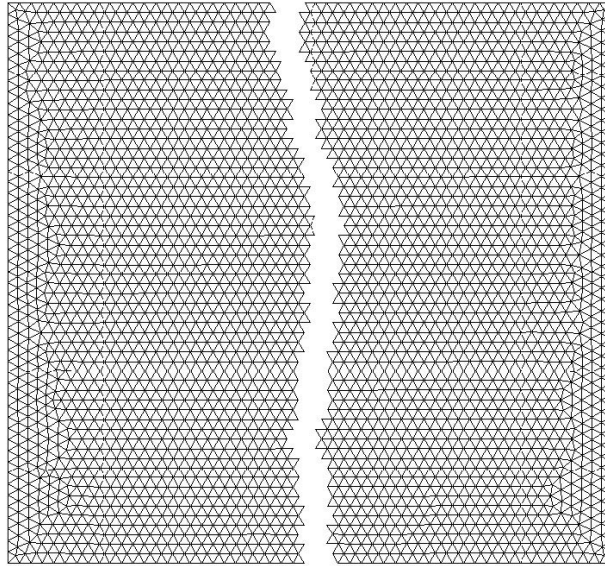


Figure 4. Two-dimensional 3000 node partitioned mesh. Partitions were generated using Metis.

The mesh partitioning shown in Figure 4 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 3-dimensional tetrahedral mesh.

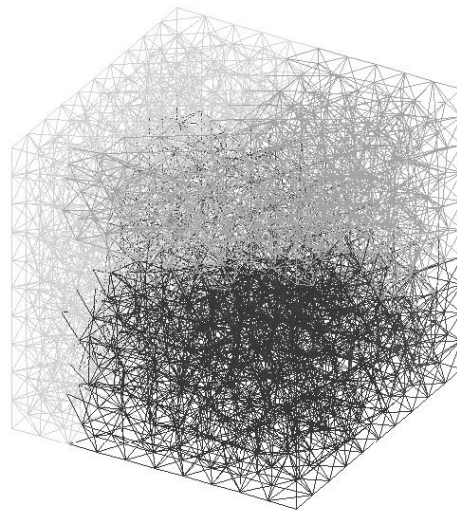


Figure 5. Three-dimensional tetrahedral element mesh. The shading denotes the 4 partitions that were computed by Metis.

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

- Read and store data from mesh input files. This includes element connectivity, node coordinates and element partitioning,

- Compute all required element and node geometrical information such as cell face areas and normal vectors for the global mesh,
- Compute all edge connectivity and geometric information from the element information for the global mesh,
- Link all nodes via edge elements,
- Setup up partition meshes with links between global and partition mesh objects including nodes, elements and edges.
- Set up connectivity between duplicate nodes on different partitions.

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.

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 insulates the physics developers from changes in the details of the materials properties input specifications.

Physics Package

The physics package contains classes that allow a developer to encode the conservation equations that he or she 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 AbsState 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.

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 a 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 a 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.

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 side 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.

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 a left and right hand side. The left and right hand side 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 provides a graphical overview of the physics class inheritance hierarchy that was generated directly from the Java source code using `GDPro` from `Embarcadero Technologies`.

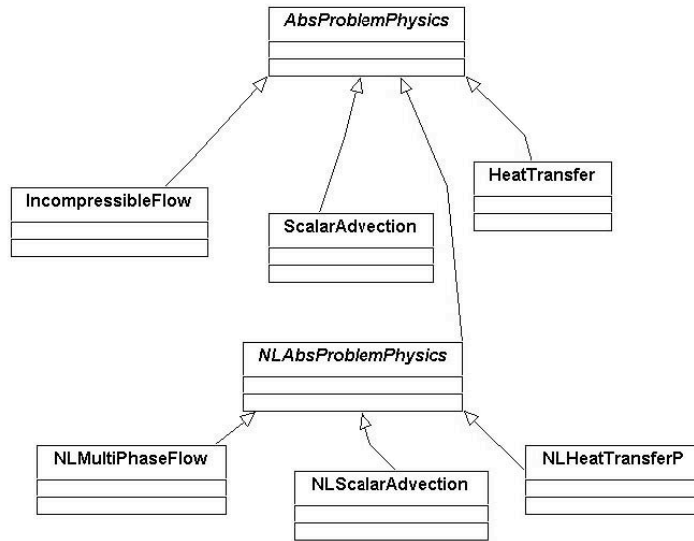


Figure 6. UML Class hierarchy diagram of the Physics package.

Solver Package

The solver package contains classes for linear and nonlinear solvers. 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 Krylov solver classes based on Conjugate Gradient and both the standard and flexible variant of Gmres, [24]. 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. The solver class inheritance hierarchy is shown in Figure 7.

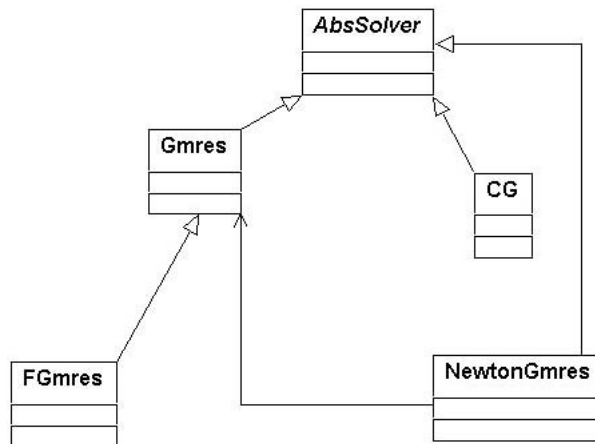


Figure 7. Solver package class hierarchy.

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.

Problem Driver 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.

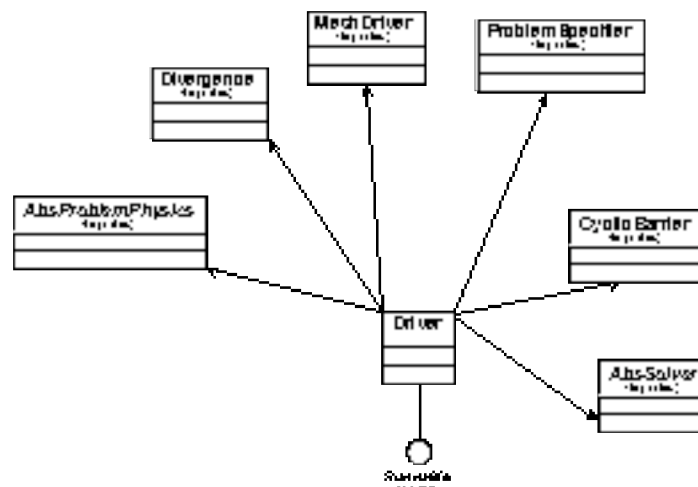


Figure 8. Association diagram for driver class.

As can be seen, the Driver class interacts with all the major CartaBlanca objects from an AbsProblemPhysics object to an AbsSolver object.

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 conservations equations in the physics classes.

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.

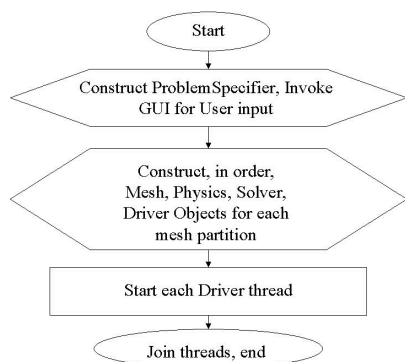


Figure 9. Flow chart for CartaBlanca main method.

Performance

Our last task here is to give some measure of the performance of CartaBlanca. To provide a basis of comparison, we used CFDLIB, [17], and CartaBlanca to perform simulations of the Martin and Moyce broken dam problem on both two- and three-dimensional Cartesian meshes. We furthermore ran both codes using the same treatment of advection mass fluxes, namely, 1st order accurate donor cell advection. Finally, we used the Conjugate Gradient method for the solution of the pressure equation in CFDLIB and for the preconditioning of the non-linear pressure residual in CartaBlanca. Thus, the two sets of calculations, while not identical, were as close to the same as possible in terms of the numerical methods used. Table 1 provides a comparison of the speed of the codes in terms of the so-called grind time--average wall-clock processing time per time step per node.

Table 1: Performance comparisons between CartaBlanca and CFDLIB on the broken dam problem using 1st order (donor cell) fluxes for advection. Table entries are 'grind times' defined as average wall-clock processing time in microseconds per time step per node.

Case\Code	Elements	Time Steps	CartaBlanca	CFDLIB
2D quadrilateral element mesh	1100	261	321	147
3D hexahedral element mesh	4400	279	779	400

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 Sun FORTRAN 77 compiler version 5.0 with optimization. As can be seen from the Table, 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. Finally, CFDLIB was written for structured grids and does not use indirect addressing, as does CartaBlanca.

Implementation and Performance of a Particle In Cell Code Written in Java (excerpt)

S. Markidis, G. Lapenta, W.B. VanderHeyden, Z. Budimlić

Skeleton Particle-in-Cell Algorithm

The simulation of systems where plasmas are present requires the description not only of the scale of interest but also of the smaller scales that affect the physics of the systems under consideration. For instance, simulation of coronal mass ejection from the Sun requires the description of large scale processes using a magnetohydrodynamic (MHD) model. However, the MHD models require to include models of dissipation processes that develop at microscopic scales. The calculation of dissipations requires more accurate microscopic kinetic models, beyond the fluid approach. At small scales dissipations are present not only as interparticle collisions but also through electromagnetic interactions of ions and electrons at the microscopic scales. A self-consistent description of astrophysical systems must be performed at the kinetic level using the Boltzmann equation for ions and electrons, the Maxwell equations for the electromagnetic fields and the Newton (or Einstein) equation for the gravitational field. However the cost of such direct approach would be prohibitive if attempted using the most common explicit methods currently in use. The standard approach is to represent the systems with reduced models such as the Hybrid, resistive MHD, Hall MHD or two-fluid model, where some or all species are approximated in the fluid limit. In all reduced models, ad hoc assumptions of the kinetic behavior are made, most commonly in the form of prescriptions for the higher order moments of the distribution (e.g., the pressure tensor) and for the dissipation processes (e.g., anomalous resistivity).

We follow a bolder approach. We adhere to the exact kinetic model with all the correct microscopic physics. To be able to bring such approach all the way to the large scales of interest we use two powerful techniques that can make the numerical simulation manageable within the existing computing resources: object orientation and implicit formulation. The implicit formulation is described elsewhere and its description is beyond the scope of the present paper. Here we use a simpler explicit PIC algorithm and focus only on the issue of object orientation of a plasma simulation code which is described next. Below, we report a simple version of the Particle In Cell method. The scheme considered here is a full-fledged plasma simulation method currently being widely used in the plasma physics community. Our goal here is not simply to use an artificially simple benchmark to test Java performances but to test Java in a realistic application. Our simplified algorithm consists of three parts: the interpolation scheme, the Poisson solver and the particle mover, as shown in Fig. 1.

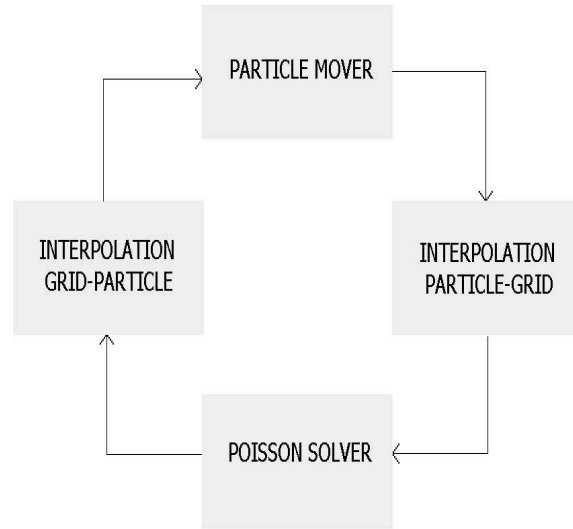


Fig. 1. An explicit electrostatic PIC algorithm.

Interpolation particle-grid

The density on the grid is calculated from the particles through the interpolation scheme defined by

$$\rho_i = \sum_{\rho} \frac{q_{\rho}}{\Delta x} W(x_i - x_{\rho}) \quad (1)$$

where i and p label grid nodes and particles, respectively, Δx is the space step while q_p is the particle charge. The classic Cloud-In-Cell (CIC) method is used for the interpolation functions:

$$W(x_i - x_{\rho}) = b_1\left(\frac{x_i - x_{\rho}}{\Delta x}\right) \quad (2)$$

where the b_1 is the first order b-spline function.

Field Solver

The Poisson equation for the electric potential Φ is:

$$\frac{d^2\Phi}{dx^2} = -\frac{\rho}{\epsilon_0} \quad (3)$$

where ϵ_0 is the dielectric constant. Equation (3) is solved using a finite difference scheme. Then the electric field E on the grid can be calculated by using the central difference discretization and solving the resulting linear system with Gaussian elimination.

Interpolation grid-particle

Given the electric field on the grid, the electric field on each particle can be calculated using the same CIC interpolation scheme:

$$E_\rho = \sum_i E_i W(x_i - x_\rho) \quad (4)$$

Particle mover

Particles are moved solving the Newton equations of motion for the particle position x_p and velocity v_p :

$$\frac{dv_\rho}{dt} = \frac{q_\rho}{m_\rho} E(x_\rho) \quad (5)$$

$$\frac{dx_\rho}{dt} = v_\rho \quad (6)$$

where m_p is the particle mass. Equations (5, 6) are discretized with the leapfrog finite difference scheme .

OBJECT ORIENTED IMPLEMENTATION

The biggest problem for any advanced plasma simulation code is to organize the complexity. Because plasma physics simulations are becoming more complex and because more physicists become involved in the writing of software, we need more sophisticated and easier development techniques. With an object-oriented framework, the computational physicist tries to organize the physical problem into objects that control the complexity of the simulation.

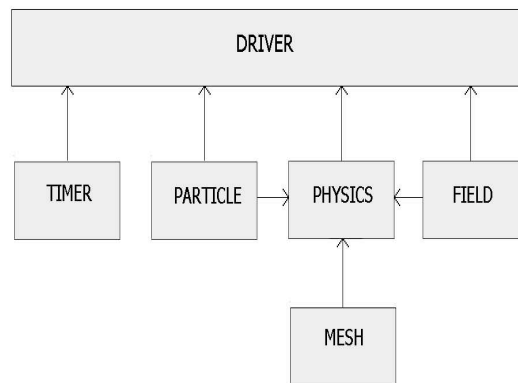


Fig. 2. Parsek Framework.

In designing Parsek, we have tried to follow two guiding principles. First, we have tried to use a full object-oriented programming from a physical point of view. Object orientation gives an elegant software design and results in a code that is easy to read and can be more effectively used by physicists who are less proficient in computer science issues.

Second, we have written the code to be as generic as possible so that the computer programmer can plug plasma physics into the proper program locations and develop a new code to study different plasma phenomena. So the programmer can extend the functionality of the code by adding models and algorithms of various level of complexity.

The algorithm discussed and tested in the present work is a skeleton version of a complete plasma simulation code. We use a PIC scheme that includes all the most important steps present in a complete code. The algorithm summarized in Sect. 2.1 follows the trajectories of a number of particles in force fields that are calculated self-consistently from charge, current and pressure densities created by the particles. Each time step in a PIC code consists of two major steps: the particle mover to update the particle positions and calculate the new charge and current densities, and the field solver to update the surrounding fields. Since particles can be located anywhere within the simulation domain but the surrounding fields are defined only on discrete grid points, the particle mover uses two interpolation steps to link the particle positions and the fields: a step to interpolate fields from the grid points to the particle positions and a step to interpolate the charge of each particle to grid points. The Parsek architecture is summarized in Fig. 2. The complete code listing is too long to be reported here, as is to be expected for a realistic simulation code that can study realistic physics problems. Parsek is composed of six separate classes:

- **Particle Object**

The Particle class describes an individual plasma particle, like an electron or ion, including its position and velocity. Basically Particle is organized as:

```
public class Particle {
    private double Position;
    private double Velocity;
    private double ElectricFieldOnParticle;
    ...
}
```

- **Field Object**

The Field class represents the electromagnetic field and its sources for a given point of the mesh. The Field object includes the charge density and the electric field for each grid node. It is written as follows:

```
public class Field {
    private double ChargeDensity;
    private double ElectricField;
    ...
}
```

- **Mesh Object**

The Mesh class contains several methods that describe mesh elements and boundary nodes. Furthermore, it provides methods to calculate discrete differential operators and to interpolate a discrete vector field onto a specified location in the mesh.

- **Physics Object**

The Physics class handles the particle mover phase, where the new particle position and velocity are determined by Newton's law, and the field solve phase, where the fields are updated solving Maxwell's equations.

- **Driver Object**

The Driver class describes the methods that handle the whole simulation. After initializing the arrays of particle and field objects with

```
Particle[]    myParticle    =    new    Particle[NumberOfParticles];  
Field[]      myField      =    new    Field[NumOfGridPoints];
```

the initial conditions for the particle velocities and positions are set. Once constructed, the simulation is advanced in discrete units of time. Fields are calculated from the sources, including the appropriate boundary conditions. At this point, the explicit method requires solving a linear system to determine the new electric fields. Next, the forces on particles are calculated by interpolating the fields to the particle positions. The forces are used to update the particle velocity, and subsequently the particle position. These procedures are repeated for each incremental time step.

- **Timer Object**

Finally, the Timer class calculates the time performance of the code.

ALTERNATIVE IMPLEMENTATIONS OF PARSEK

The object-oriented implementation of Parsek described above uses a fine-grained approach. The objects are chosen to correspond to the smallest units in the physical system under consideration: the particles and the mesh points. Alternative approaches are possible.

Previous studies have led the high performance computing community to reach two widely held beliefs.

First, programs written in Java are believed to be an order of magnitude, or more, slower than corresponding programs written in C or FORTRAN. To ascertain this point we have developed various FORTRAN and Java versions to compare their relative speed.

Second, fine-grained object orientation, either in C++ or in Java, is believed to be much slower than coarse-grained object orientation. Fine-grained object orientation can be loosely defined as the choice to define objects at the smallest scale of interest in the problem being considered. For plasma simulation this corresponds to the choice outlined in the previous section where the objects were chosen as single particles and single mesh points. The crucial feature of fine object-orientation is that the objects are small and large

arrays of them are required. The additional cost of handling arrays of objects is believed to result in a great penalty in terms of computing efficiency. Coarse-grained object orientation, instead, defines broader objects that include larger units of the system under consideration. For plasma simulations this corresponds to choosing objects composed by the whole grid or by whole populations of particles (such as all ions or all electrons). The crucial feature of coarse object orientation is that all relevant arrays are wrapped inside the objects and no arrays of objects are required. Previous studies have reported penalties of one order of magnitude when fine-grained object-orientation is compared with coarse-grained object orientation and only the traditional compiler techniques are used. To test this issue we have developed different versions of Parsek all written in Java but using different object orientation styles.

The two beliefs described above are often based on evidence obtained some years ago when the Java virtual machines and compilers were still in their infancy. Furthermore, often such conclusions were reached using simple methods not applied to any scientific problem. More recently, extensive benchmarks of Java using a suite of standard mathematical problems has shown that contrary to the commonly held beliefs, Java is almost on par with FORTRAN.

Here we intend to conduct all tests with the most modern compilers on the most modern computer architectures. And we will conduct all tests for a real problem of plasma simulation where the final answer is a significant plasma physics result. While most of the previous performance studies were conducted on benchmark problems, we will base our study on a realistic plasma physics simulation tool.

Below, we put the two beliefs described above to test using several alternative versions of Parsek both in Java and in FORTRAN 90. All versions are equivalent from the algorithmic point of view but are radically different in the choice of software architecture and programming language. Below we describe the various versions.

Coarse-Grained Object Oriented Parsek

Two approaches to object orientation are possible: a "coarse grained object-oriented" (referred to as LOO) and a "pure object-oriented" (referred to as OO) programming style. We have described the OO design in the section above. With a LOO technique the arrays that describe particles and fields are wrapped in two objects that represent the whole particle population and electrostatic field states. The coarse grained object-oriented Parsek is composed by 5 separate classes:

- **Particles Object**

The Particles class in the LOO framework acts as a container to store the characteristic data for N individual particulate elements. Each individual particle has several attributes, such as position and velocity. In the code, examples are:

```
private double[] Position = new double[NumberOfParticles];  
private double[] Velocity = new double[NumberOfParticles];
```

In a LOO code arrays are wrapped in a single object, and no array of objects is used. In the fully OO PIC code, instead, objects were single particles and arrays of objects were used. Moreover the Particles object contains methods to move and accelerate the particles, and to check if the particles are leaving the boundaries. Unlike the case

of the OO PIC code, in a LOO PIC code there is a direct interaction between the Particles and Mesh objects.

- **Fields Object**

A Fields object represents a discretization of a continuous field quantity over an underlying mesh. Internally, Fields data is stored essentially as an array, containing charge density, potential, and electric field values on the grid. In Java, it is written as follows:

```
private double[] ElectricField = new double[NumOfGridPoints];  
private double[] Potential = new double[NumOfGridPoints];  
private double[] ChargeDensity = new double[NumOfGridPoints];
```

The Field Solver is a method of this class.

- **Mesh Object**

It contains the information about the grid and methods to calculate the interpolation functions.

- **Driver Object**

It coordinates the other objects and controls the progress of the computational cycle.

- **Timer Object**

The Timer object calculates the timing performance of the code.

FORTRAN Style Parsek

Although Java is a full-fledged object oriented language, old-fashioned procedural programming remains possible using static classes. We have developed an additional Java version of Parsek that uses a "FORTRAN style" (FS) procedural program. All methods are static, arrays are passed directly as arguments and the data is accessed directly. The FORTRAN style code is procedural in only one class. It includes the usual Particle mover, Field Solver, and Interpolation stages.

FORTRAN 90 Parsek

To compare Java and FORTRAN 90 performances we wrote two additional versions of Parsek in FORTRAN 90. We have chosen to use modern FORTRAN90 features, including types, modules and array notation. Two versions have been written in FORTRAN 90: one with coarse-grained types and one with fine grained types.

Fine-grained types

The fine-grained data is stored as an array of elements of a defined type. The Particle type describes an individual plasma particle, like an electron or ion, including its position and velocity. Particle is organized as:

```
TYPE Particle  
  DOUBLE PRECISION :: Position  
  DOUBLE PRECISION :: Velocity
```



```
DOUBLE PRECISION :: ElectricFieldOnParticle END TYPE Particle
```

The Field type represents the electromagnetic field and its sources for a given point of the mesh. The Field type includes the charge density, the current density, the electric field for each grid node. Field is written as follows:

```
TYPE Field
```

```
DOUBLE PRECISION :: ChargeDensity
```

```
DOUBLE PRECISION :: Potential
```

```
DOUBLE PRECISION :: ElectricField END TYPE Field
```

Coarse-grained types

The coarse grained data is stored as a defined type that includes arrays to accommodate particle and field data. The Particles type describes an entire species of particles (composed of NumberParticles individual particles), like all electrons or all ions, including their position, velocity, mass, charge and the electric field acting on them. Particles is organized as:

```
TYPE Particles
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: Position
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: Velocity
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: Charge
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: Mass
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: ElectricFieldOnParticle
```

```
END TYPE Particles
```

The Fields type represents the electric field and its sources for the whole mesh (composed of NumOfGridPoints points). The Fields type includes the charge density, the current density, the electric field for each grid node. It is written as follows:

```
TYPE Fields
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: Charge Density
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: Potential
```

```
DOUBLE PRECISION, dimension(0 : NumberParticles-1) :: ElectricField
```

```
END TYPE Fields
```

Testing Environment

Table 1 presents the platforms used for measuring the performance. The Java environment is reported in Table 1 for each platform. The machines were relatively unloaded during each simulation, and several runs were made at each test condition with the average time recorded.

Table 1: Platforms Used for Measuring Performance

Platforms	Sun Blade 2000	Dell Precision 520	Dell Latitude C840	Dell Latitude C840
Processor	UltraSPARC-III+	Xeon	Pentium 4	Pentium 4
Processor Speed	1.2 GHz	1.9 GHz	1.6 GHz	2.0 GHz
Memory	512MB	4GB	512MB	1GB
Operating system	SunOS 5.8	RedHat Linux 8	Windows 2000	Windows XP
SUN Java build	1.4.1-02-b06	1.4.1-b21	1.4.2-b28	1.4.1-b21

Java Performance

Table 2 and Figures 3 and 4 compare the execution time for the different implementations of Parsek. All tests are performed on the SUN platform. In Table II the second, third and fourth columns show the performance of Parsek, developed in Java with different object orientation techniques. Each row presents the time performance in milliseconds, for simulations with a different number of particles showed in the first column. Figure 3 illustrates graphically these results. Figure 6 describes the dependency of the timing performance on the number of particles. Clearly, the timing shows the relative slowness of the fine-grained object-oriented programming solution. While FS and LOO show a difference in performance of only a few percent, the OO Parsek is on average 1.5 times slower than the others. However, it must be stressed that, in our PIC code, the fine-grained object-orientation is not as penalizing as reported in previous studies.

Recently "The Center For High Performance Software" of the Rice University has developed JaMake, a Java compilation environment that uses advanced program analysis and transformation techniques. JaMake is able to improve the performance of a fine grained object oriented program to bring it almost to the same speed as a procedural or coarse-grained object oriented program. Although the JaMake package is still under development, we have succeeded in testing the performance of the OO version of Parsek when compiled using JaMake. The last column of Table 2 shows the timing performance of the OO version of Parsek compiled with JaMake. The results clearly show the elimination of the additional costs of fine grained object orientation.

Table 2: Execution times, showing the almost 1.5 times speedup of FS and LOO over the pure OO. Tests on the SUN platform shown in Table I.

num.particles	OO(ms)	LOO(ms)	FS(ms)	OO-JaMake (ms)
1000	210	128	118	167
2000	413	233	233	281
5000	1032	551	605	603
10000	1999	1063	1224	1206
20000	4012	2143	2419	2517
50000	10273	5799	6164	6258

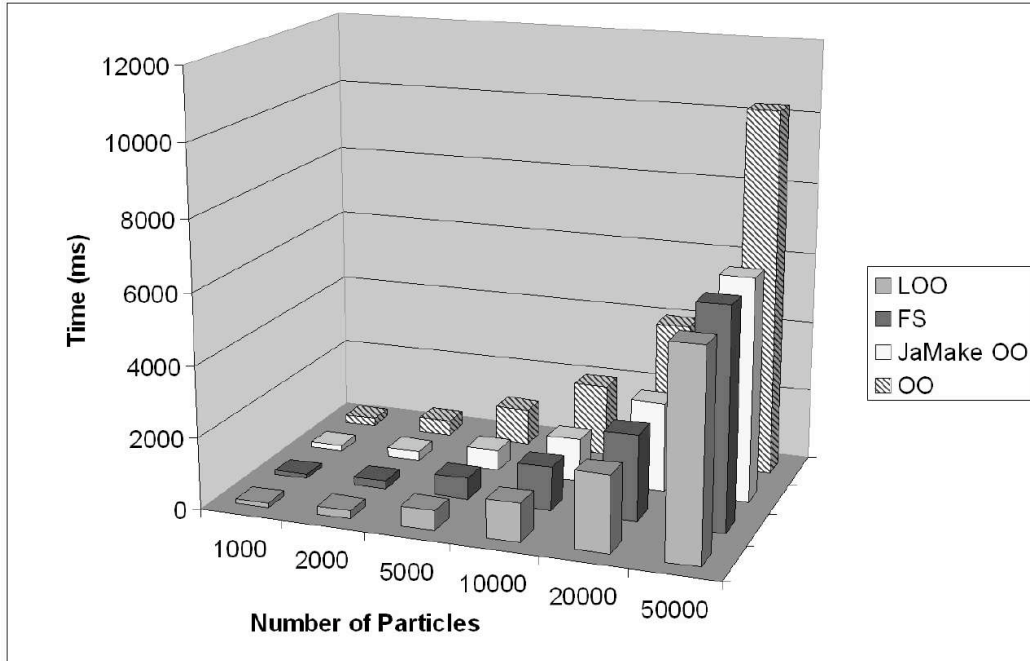


Fig. 3. Execution times for FS, LOO, OO Parsek codes in Java. The performance data is reported in Table 2.

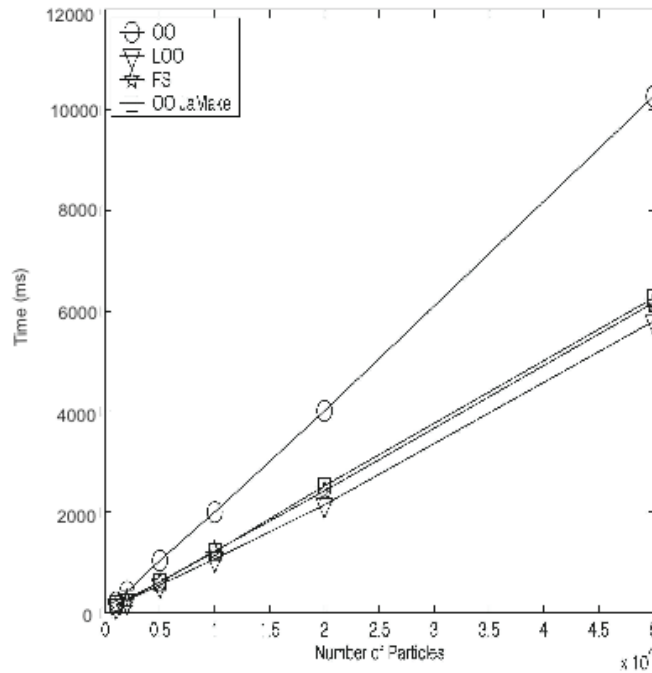


Fig. 4. Execution times: increasing the number of particles, the execution time increases linearly. The performance data is reported in Table 2.

Java vs FORTRAN 90

Table 3 and Figure 5 compare the three Java implementations of Parsek with the two implementations in FORTRAN 90. Different platforms and operating systems are considered. All tests are conducted using 50,000 particles. In Table III the six last columns show the performance of Parsek, developed in FORTRAN 90 and in Java with different programming techniques. The last column considers the fine-grained OO version in Java compiled with JaMake. Each row presents the time performance in milliseconds, for simulations running under different operating system showed in the first column. The version of Java used is listed in Table I for each platform. For the two FORTRAN implementations, we use the Lahey FORTRAN 95 compiler (version 5.7 for the Windows platform and version 6.1 for the Linux platform) with maximum optimization on the Linux and Windows platforms. The Sun FORTRAN 90 version 6.2 with the compiler option -O3 is used for the SUN platform. Figure 7 illustrates graphically these results.

Clearly, on the SUN platform, FORTRAN and Java performances are comparable, with some Java implementations even outrunning both FORTRAN implementations. Conversely, on the INTEL platforms, the coarse grained FORTRAN version remains about a factor of two faster than the fastest Java, but the fine grained FORTRAN version is actually slower than some Java implementations. The direct comparison between Java and FORTRAN requires further comments.

Table 3. Execution times for a simulation with 50,000 particles under different operating system shown in Table 1

O.S.	F90 fine(ms)	F90 coarse (ms)	JavaFS(ms)	JavaLOO(ms)	JavaOO(ms)	JavaOO-JaMak
Windows2000	3765	1680	3255	3956	5800	3625
Linux	2742	1130	2874	3420	3960	4250
SUN	7430	6485	6164	5799	10273	5386

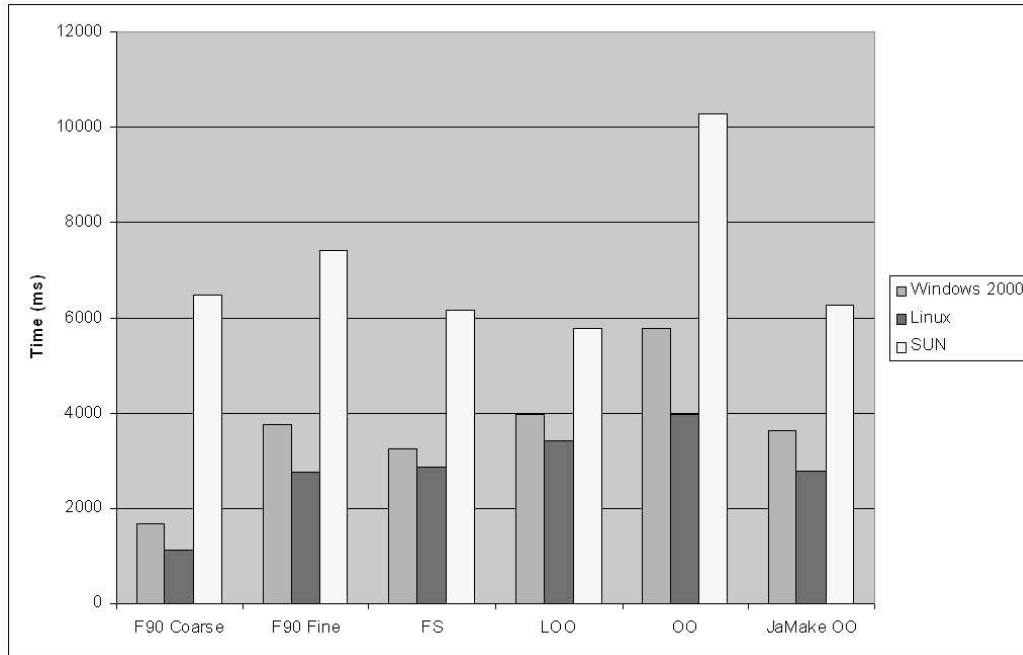


Figure 5. Execution times for FS, LOO, OO Java implementations and the two Fortran90 implementations. The performance data is reported in Table III.

First, on the Windows 2000 platform we tested also the Compaq FORTRAN compiler that resulted in considerably slower execution. On the Linux platform we also tested the ABSOFT compiler version 7, which was also slower but by a smaller margin.

Second, the two FORTRAN implementations perform significantly different on the two INTEL platforms. Coarse-grained typing results in a improved handling of the cache since operations conducted on an array of quantities (such as the particle positions) are closer in memory and are loaded in the cache all together in a block.

Third, we have repeated the tests above with a different number of particles, reaching virtually identical conclusions.



Situated on more than 43 square miles in northern New Mexico, Los Alamos National Laboratory has more than 7400 regular full-time employees and an approximate annual budget of \$1.7 billion. We are operated by the University of California for the National Nuclear Security Administration of the US Department of Energy.

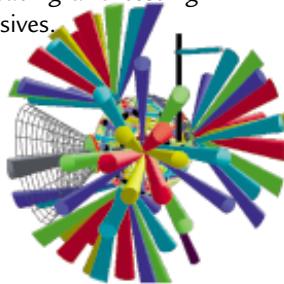
Over our 58-year history, our primary mission has been to apply science and technology to problems of national security. At first our mission was dedicated principally to

stockpile stewardship, but an ever-changing world has expanded our mission to cover other global threats, such as computer hacking and biological terrorism.

The Laboratory also conducts basic and applied research that addresses societal issues, such as developing alternative energy sources, designing the world's first functional quantum computer, and tracking down the most common ancestor of the HIV-1 strains responsible for AIDS. A vital facet to all our work is R&D collaboration with private industry. The following are five key R&D areas.

Maintaining Our National Security.

The Laboratory continues to work on resolving nuclear weapons issues and on deciphering emerging technological challenges posed by the nuclear weapons stockpile. To accomplish this mission, the Laboratory applies an array of science and technology, from theoretical and computational physics to fabricating and testing explosives.



Scientists are developing hohlraums to achieve thermonuclear ignition, a process for the controlled release of great amounts of energy.

Developing Environmental Solutions.

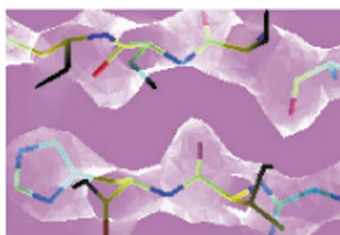
The Laboratory's expertise in this area spans a variety of environmental technologies, from waste minimization to environmental restoration and waste management. The principal goal of these and other programs is to maintain a safe and healthy environment for present and future generations.



Radiation-tolerant materials like this could provide safe nuclear waste storage for thousands of years.

Understanding the Complexity of Life.

At Los Alamos, scientists have united biological, physical, and computational sciences in an effort to better understand biological complexity. With such knowledge, we will develop technologies that address a number of critical issues, such as detecting, identifying, and defeating diseases and determining how genes function in the cell and the whole organism.



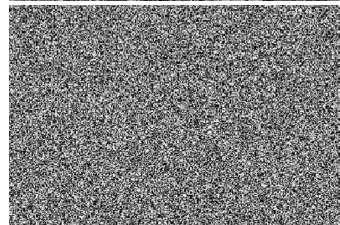
Researchers are using density contour maps to solve novel protein structures, many of which have medical applications.

Defeating Global Threats.

Los Alamos researchers are developing technologies that defend the world against a number of international threats, such as the proliferation of nuclear weapons, chemical and biological agents, and information terrorism and computer infiltration.



The Blue Mountain Supercomputing Platform helps researchers maintain the safety and reliability of the US nuclear stockpile.



Using a quantum cryptography process developed at Los Alamos, a sender can transmit an image (top) that is encrypted (middle) in such a way that only the intended receiver can decrypt it (bottom).

Developing Supercomputers.

In collaboration with industry, the Laboratory continues to develop faster and more sophisticated supercomputers to handle the extraordinarily complex calculations required to study the dynamics of nuclear weapons, global climate and ocean changes, or oil flow through underground rock.

Visit the LANL web site at <http://www.lanl.gov/worldview/>