American Institute of
Aeronautics and Astronautics

AIAA-2004-1091

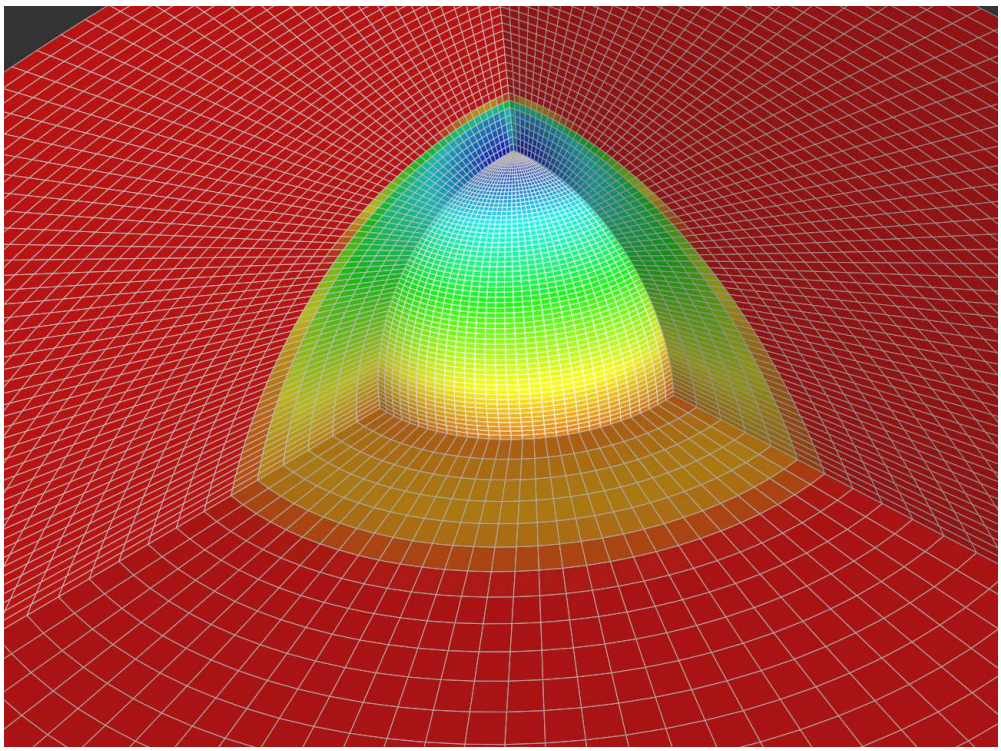
JUSTGRID[1]

A PURE JAVA HPCC GRID ARCHITECTURE FOR MULTI-PHYSICS SOLVERS USING COMPLEX GEOMETRIES.

Thorsten Ludewig[2], Jochem Häuser[*], Torsten Gollnick[*] and Hans-Georg Paap[**]*

[*] Univ. of Applied Sciences and Dept. of High Performance Computing
Center of Logistics and Expert Systems (CLE) GmbH, Salzgitter, Germany

[**] HPC Consultant, Barbing, Germany



42ND AIAA
AEROSPACE SCIENCES MEETING AND EXHIBIT, RENO, NEVADA
5-8 JANUARY 2004

1 This name is due to Dr. Jean-Luc Cambier, U.S. Air Force Propulsion Directorate, EAFB
2 This paper is part of the PhD work of the first author.

30. December 2003

Abstract

After the *Earth Simulator*, built by NEC at the Japan Marine Science and Technology Center (JAMSTEC) on an area of *3,250* m^2 (50m×65m), began it's work in March 2002 with the outstanding performance of *35,860 Gflops* (40 Tflops peak) [1], numerous scientists opted in favor of the high-performance computation and communications (HPCC) roots, suggesting to *build again Cray type vector supercomputers* that dominated scientific computing in the mid seventies. On the other hand, other major industrial players in supercomputing, for instance, IBM, Sony, and Toshiba, announced a new processor design for the upcoming *Sony Playstation 3* called *Cell* and, according to these companies, a Cell processor should deliver about one trillion floating-point calculations per second (Tflop). Cell would be roughly a 100 times faster than the current Pentium 4 chip running at 2.5 GHz. Cell will most likely use between four and 16 general-purpose processor cores per chip [2]. If one can provide enough main memory for such a processor, a formidable and inexpensive parallel system linked by a very high bandwidth interconnect could outperform the Earth Simulator. The size of such a cluster would fit in the average kitchen and the off-the-shelf technology would cost only a fraction of the Earth Simulator. It should be remembered that the computer games industry is responsible for the revolution in high end 3D graphics cards that convert any PC into a most powerful graphics workstation. It should be obvious, despite the computational power of the Earth Simulator, that this definitely is *not* the road of HPCC for general scientific and engineering computation.

"I hope to concentrate my attention on my research rather then how to program", says Hitoshi Sakagami, a researcher at Japan's Himeji Institute of Technology and a Gordon Bell Prize finalist for work using the Earth Simulator [1].

We fully agree with this statement, and this is one of the major reasons that we have chosen *Java* as our high performance computing language. Programming vector computers is a difficult task, and to obtain acceptable results with regard to announced peak performance has been notoriously cumbersome. On the other hand, Cell like systems with many processors on a single chip need to be programmed in a *multi threaded* way. *Threads* are a substantial part of the Java programming language. Java is the only general programming language that does not need external libraries for parallel programming, because everything needed is built into the language. In addition, there are major additional advantages of the Java language (object oriented, parallelization, readability, maintainability, programmer productivity, platform independence, code safety and reliability, database connectivity, internet capability, multimedia capability, GUI (graphics user interfaces), 3D graphics (Java 3D) etc.) which were discussed in detail in [6-11]. In the current paper, we present the next stage in the design of an internet capable parallel Java based multi-physics solver. The overall task is to build a fast, efficient, portable, and platform independent collaborative scientific and engineering parallel multi-physics simulation environment for visualization and distributed computing over the Internet, using a three tier client-filter-server approach. The applications in this paper are taken from aerospace.

1 The Java Ultra Simulator Technology (JUST) Grid

The latest version of Java is a serious competitor to the traditional HPCC programming languages like FORTRAN or C/C++ [10, 11]. The single-processor performance of a Java code is now on par with C++, and the speedup on common symmetric multi processor (SMP) machines is excellent.

Runtime (2GHz, Pentium4, 1GB Memory)	time in s
Sun JVM 1.4.2_02 (-server)	2,12
GNU gcc version 3.3.1 (-O3 -mcpu=pentium4)	3,16

Table 1: Sequential matrix multiplication using a 30 times 30 matrix doing 10000 iterations on a Linux Pentium 4 PC

There exists the ***JUSTGrid*** framework which is a software platform providing the possibility to simplify as well as to speed up solver development. Among the additional features of **JUSTGrid** are the handling of complex 3D grids, both structured and unstructured.

JUSTGrid is the basis for ***JUST*** (Java Ultra Simulator Technology) that is a revolutionary computing software that dramatically improves the ability to quickly create new kinds of software systems across the whole field of science

and engineering, embedded in an Internet-based environment. ***JUSTGrid*** is a software platform and virtual computing environment that enables scientific and engineering computation of large-scale problems in an Internet-based computational grid environment, integrating computer resources at different, geographically distributed, sites. ***JUSTGrid*** enables the user to create his simulation software at the client site at run time by using a Java based browser GUI. The solver package composed by this GUI is sent in binary form to the server site, replacing the default simulation solver package.

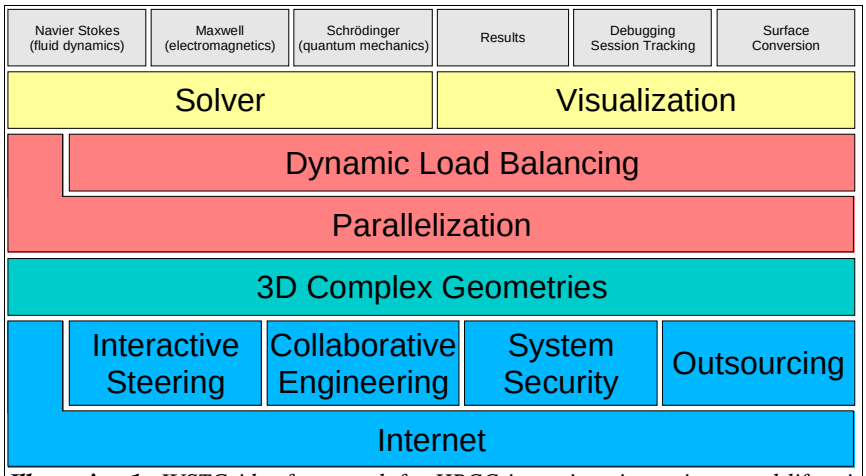


Illustration 1: *JUSTGrid a framework for HPCC in engineering, science and life sciences.*

JUSTGrid is a completely Java based software environment for the the user/developer of HPC software. ***JUSTGrid*** takes care of the difficult tasks of handling very complex geometries (aircraft, spacecraft, biological cells, semiconductor devices, turbines, cars, ships etc.) and the parallelization of the simulation code as well as its implementation on the Internet. ***JUSTGrid*** builds the computational Grid, and provides both the *geometry layer* and *parallel layer* as well as an interface to attach any arbitrary solver package to it. ***JUSTGrid*** is implemented on the client site, where the user resides, and on the compute server where the computations are to be performed. It also can access one or more data servers, distributed over the Internet. A default solver package resides on the server site. For instance, this may be a fluid dynamics solver. If the client decides that it will use this solver, the necessary data needs to be collected and sent to the server. In case a totally different solver is needed, e.g., a solver for Maxwell's equations to compute, for instance, the electromagnetic signature of a ship or aircraft or to simulate the trajectories of an ionized plasma beam of an ion thruster, the correct solver object has to be sent from the client to the server at run time. As described above, the new solver is created through the GUI at the client site at run time. This solver object is sent in binary form to ensure code security. If the solver object is written in Java, the Remote Method Invocation (RMI) class is used, if not, the Common Request Broker Architecture (CORBA) or the Java Native Interface (JNI) is employed to integrate so called legacy solvers. The server does not need to know anything about the solver as long as the solver interface is correctly implemented. The parallelization is entirely based on the Java thread concept. This thread concept has *substantial advantages over the PVM or MPI library parallelization* approach [6-7, 12-13], since it is part of the Java language. Hence, no additional parallelization libraries are needed.

JUSTGrid also provides a third layer, the *solver package layer*, to be implemented on the client site. This layer is a Java interface, that is, it contains all methods (functions in the context of a procedural language) to construct a solver whose physics is governed by a set of conservation laws. An interface in the Java sense provides the overall structure, but does not actually implement the method bodies, i.e., the numerical schemes and the number and type of physical equations. This *JavaSolver-Interface* therefore provides the software infrastructure to the the other two layers, and thus is usable for a large class of computational problems based on finite volume formulation. It is well known that the Navier-Stokes equations (fluid dynamics), Maxwell's equations (electromagnetics, including semiconductor simulation) as well as Schrödinger's equation (quantum mechanics) can be cast in such a form. Thus, a large class of solvers can be directly derived from this concept. The usage of this solver package, however, is not mandatory, and any solver can be sent by the client at run time. All solvers extend the ge-

neric solver class, and in case a solver does not need to deal with geometry, the generic solver class is used directly instead of the conservation law solver class.

JUSTGrid provides the coupling to any existing solver, but freeing this solver from all the unnecessary burden of providing its own geometrical and parallel computational infrastructure. Because of Java's unique features, ***JUSTGrid*** is completely portable, and can be used on any computer architecture across the Internet ,as long as a Java Runtime Environment (JRE) is provided.

The online view of the solution progress gives an impression of what is going on in during a computation and should be used as a steering aid. It is not meant as a replacement for visualization software like TecPlot™ or Ensight™.

2 *JUSTGrid* Simple Frontend

This ***JUSTGrid*** simple frontend is a rapid prototype to demonstrate the simplicity of a well designed GUI for a 2D mono block Euler solver. It converts GridPro™ and TecPlot™ grid files into a validated XML file format storing additional information: description, physical parameters and boundary conditions. The XML file with its corresponding DTD (Document Type Definition) along with the result of the computation is automatically stored as a ZIP-file into the user's file system with the file extension GRX. The ZIP-file format is a well known format on a wide range of computer systems (UNIX/Linux, Windows, MacOS, etc.) and can be extracted with tools like Java's JAR, UNZIP or WinZIP. This frontend acts also as a control center for the Euler solver. Dealing with the Java Media Framework one has the possibility to render video files from the solution domain during the computation, employing the integrated video player to display the solution video in real time. The integrated video player acts like a normal video player, for instance, the usual commands, ***forward, pause and reverse playing*** are available. In ***JUSTGrid*** remote data visualization along with data compression and feature extraction as well as remote computational steering is of prime importance. Since ***JUSTGrid*** allows multiple sessions, multiuser collaboration is needed. Different visualization modules are needed, but here a computational fluid dynamics (CFD) module, allowings the perusal of remote CFD data sets is being developed, based on the Java3D standard.

In large simulations, grids with millions of cells are computed, producing hundreds of megabytes of information during each iteration. Depending on the numerical scheme, several thousand iterations may be needed either to converge to a steady state solution or to simulate a time-dependent problem. Hence, a fast connection is needed to move data to the client where it can be analyzed, displayed or interacted with in order to navigate the parallel computation on the server. Therefore a visual interactive package, termed the ***JUST VVTK*** (Virtual Visualization Toolkit) is provided (illustration 17 on page 14).

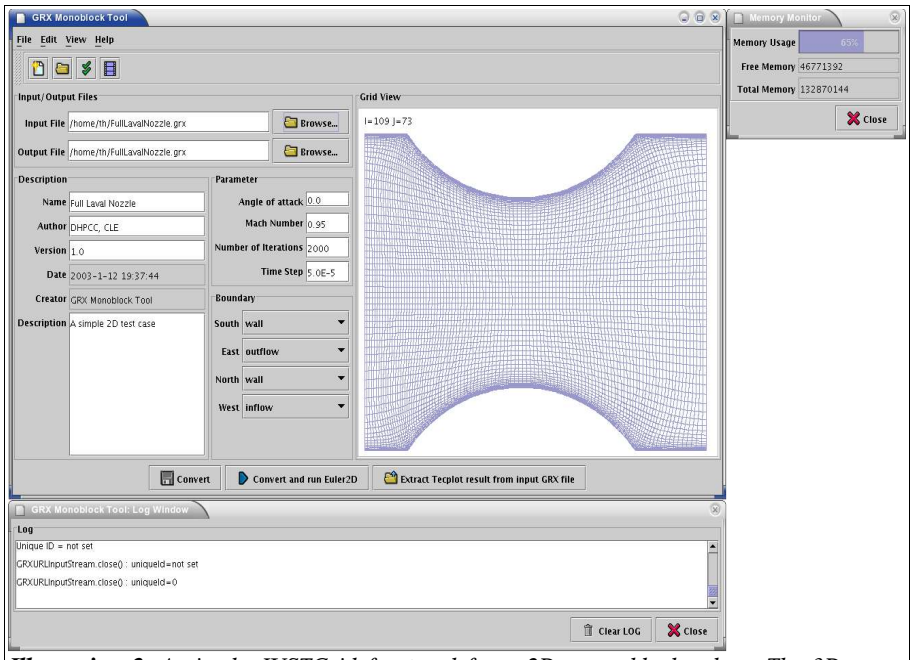


Illustration 2: *A simple JUSTGrid front-end for a 2D mono block solver. The 3D mono block solver is being developed. Upon testing, this solver is merged with the parallel infrastructure of the JUSTGrid.*

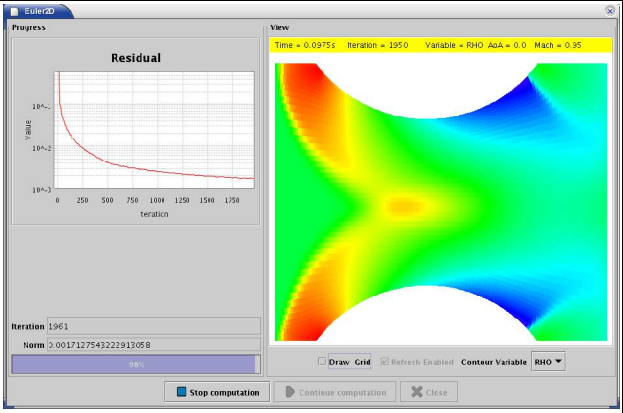


Illustration 3: *Online view of the solution progress (video production).*

3 JUSTGrid Features at a glance

- **Replace the default solver with your own solver (mathematics).**
  The design of **JUSTGrid** allows to replace every default computation relevant class (Solver, Cell, SessionHandler, BoundaryHandler, ...) on the server except the Session and the Master Implementation.
- **Exchange the solver online during the computation.**
  The exchange of a specific class on the **JUSTGrid** server can be initiated while the computation is running without a restart cycle.
- **Dynamic load balancing for free on SMP Architectures.**
  Dealing with multithreaded architectures transfers the responsibility for the load balancing from the applications to the operating systems. Modern operating systems like Sun Solaris are very efficient in distributing the thread load on the available system CPUs.
- **Simple geometrical model for the programmer.**
  JUSTGrid frees the programmer from dealing with complex geometries. The programmers view on a cell is always in a mathematical universe where every edge has a normalized length 1. The transformation from the physical- to the mathematical- coordinate system is done by **JUSTGrid**.
- **Simple Solver API (interface)**
  One thing we kept in mind during the whole design process it was Einstein he said: Make it as simple as possible but not simpler. For Example if one like to write his own multi-block solver he has to implement only one method named `solve`. For other types of solvers there are only a few more methods to implemented. Illustration 5 shows an UML class diagram of the JUSTGrid solver interfaces.
- **OnlineVisualization on demand**
  JUSTGrid provides access to all computational data in the solution domain at any state of the computation. Illustrations 3 and 14 showing online visualizations of the solution domain.
- **Collaborative engineering**
  Via a unique Session-ID multiple clients are able to connect to the same compute session on the server. As an example: if an engineer wants to ask an expert about the correctness of his running computation the engineer sends the Session-ID to the expert, he could now connect to the compute session and visualize the running computation online and give his comment back to the engineer.
- **Multiple sessions on one server.**
  The **JUSTGrid** server is able to run as many sessions as you want; it is only limited by the available resources on the server system.
- **Application and network security**
  Java has a very smart security architecture that protects your code and your data from unauthorized access or modification. **JUSTGrid** benefits from the application security features and uses the network security layer for the client/server communication.
- **Loaders and writers for structured and unstructured grids and TecPlot™ data files are available.**
  Data files could be stored on the client as well on the server side.
- **Modern object oriented software architecture**
  The object oriented architecture allows to benefit from techniques like inheritance, data encapsulation and message passing. These are some of the features that make the code more robust and maintainable.

4 JUSTGrid Software Architecture

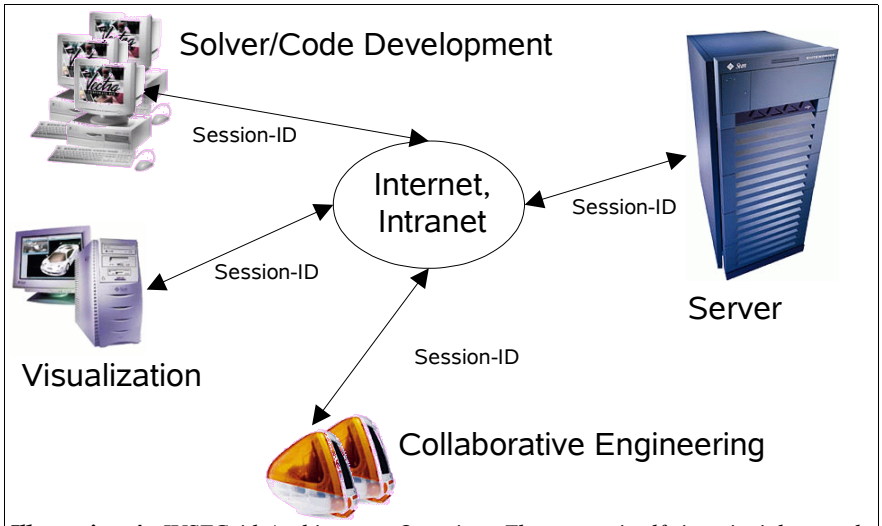


Illustration 4: *JUSTGrid Architecture Overview. The server itself, in principle, can be distributed over the internet.*

With a distributed computing system, for example an engineer at a workstation running a simulation on a supercomputer, the engineer would like to see the computation just as if it where happening on the workstation. The Java Remote Method Invocation (RMI) is one way to do this: the engineer (client) manipulates objects with a user interface, but the actions he performs (the *method invocation*) are actually performed on objects on the supercomputer (server). This transparent distribution of the computation and steering are vital if we are to provide both the immediacy of a workstation code with the computational power of the supercomputer.

If the client establish the first connection to the server and requests a new session a random 64Bit Session-ID is created on the server and send back to the client. Every further action to the session is bound to that Session-ID. With a valid Session-ID many clients are able to connect to the same session and gives the engineers the possibility to steer or visualize the computation from many different clients (collaborative engineering). Another advantage of **JUSTGrid** is if the internet connection to the server goes down a client can easily reestablish the connection to server when the internet connection becomes up using the Session-ID. During the internet connection is down the computation does not stop and no data will be lost. Additionally to the communication with RMI **JUSTGrid** has also implemented a streaming server for large data files because of RMI is packed oriented and inefficient for large continuos data sets.

4.1 Solver Interface Class Diagram (UML)

Illustration 5 shows the class diagram of the `JpSolver` interface. This simple interface has to be implemented with the numeric for the problem to be solved. The **JUSTGrid** automatically starts one solver per block or domain (illustration 8 on page 9) and handles the boundary update after each iteration. As default a loose synchronization is performed that means if all neighboring blocks of a current block are become ready with the actual iteration the boundary update for the current block will be done by the **JUSTGrid** `JpBoundaryHandler`. After the boundary update the solver (block) starts for the next iteration. This synchronization strategy increases dramatically the efficiency of the dynamic load balancing.

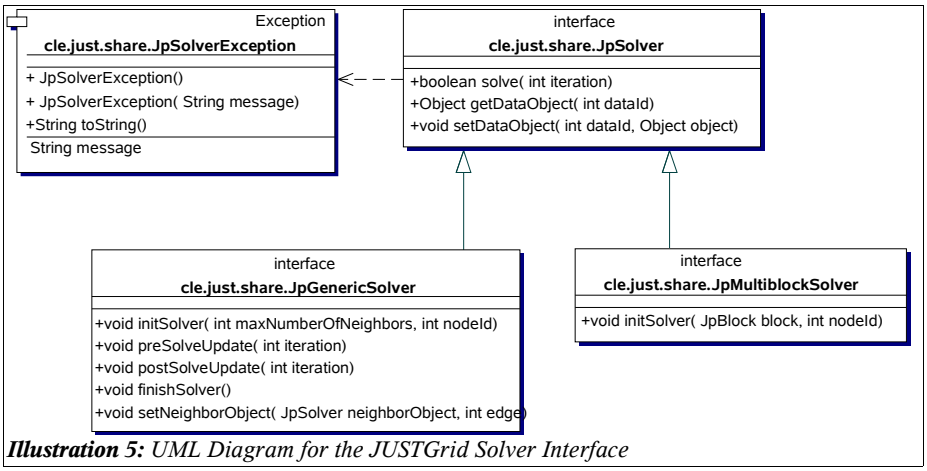


Illustration 5: *UML Diagram for the JUSTGrid Solver Interface*

4.2 Server Class Diagram

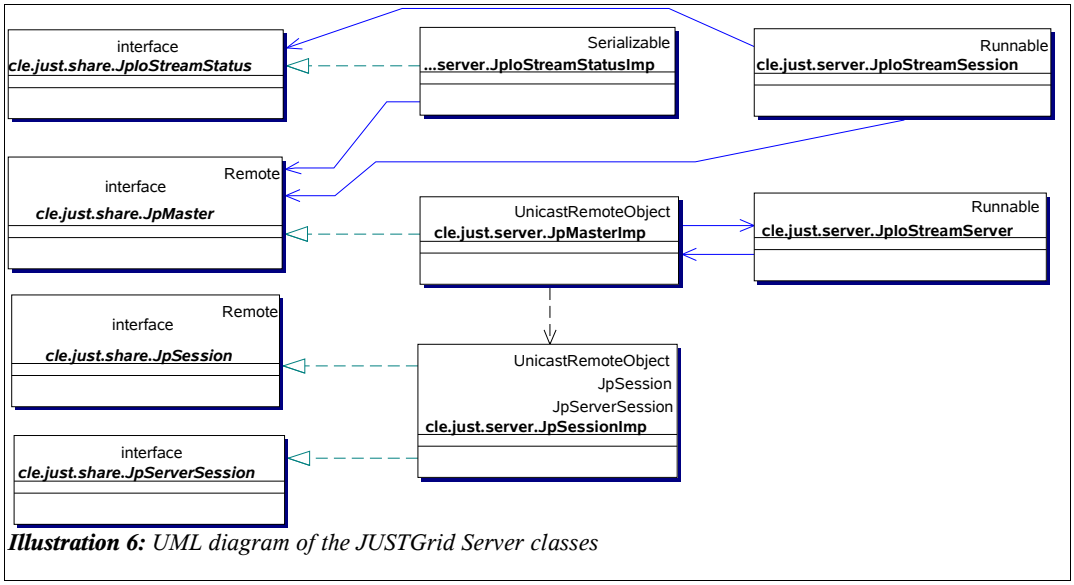


Illustration 6: *UML diagram of the JUSTGrid Server classes*

The central class on the server side is `JpMasterImp`. It provides the remote view of the client onto the server implementing the `JpMaster` interface. It is responsible for the creation, monitoring and deletion of the servers sessions and io stream severs. The `JpIoStreamServer` is used to transport large amount of data sets between client and server where the packet oriented RMI (Remote Method Invocation) is inefficient. Before the `JpMasterImp` is started on the server machine the `rmiregistry` has to be started on the same computer. The `rmiregistry` is a java based naming service that associates a name (String) with an remote object and is part of the Java 2 Standard Edition. This registry gives the client the ability to get in contact with a registered `JpMasterImp` object.

4.3 Session Class Diagram

In illustration 7 the `JpSession` provides all functionality's for interactive steering of a computation like initialization, data transfer, start and stop. The `JpSolverHandler` is responsible for the initialization and monitoring of all compute nodes. that are represented by the `JpNodeImp` class. It is also dealing with the `JpBoundaryHandler` which is responsible for the boundary update. If the computational problem requires a totally different initialization or data transfer, e.g. integration of legacy FORTRAN code, the default `JpSolverHandler` and `JpBoundaryHandler` can also be replaced on demand in the **JUSTGrid** framework like the solver or the cell implementation.

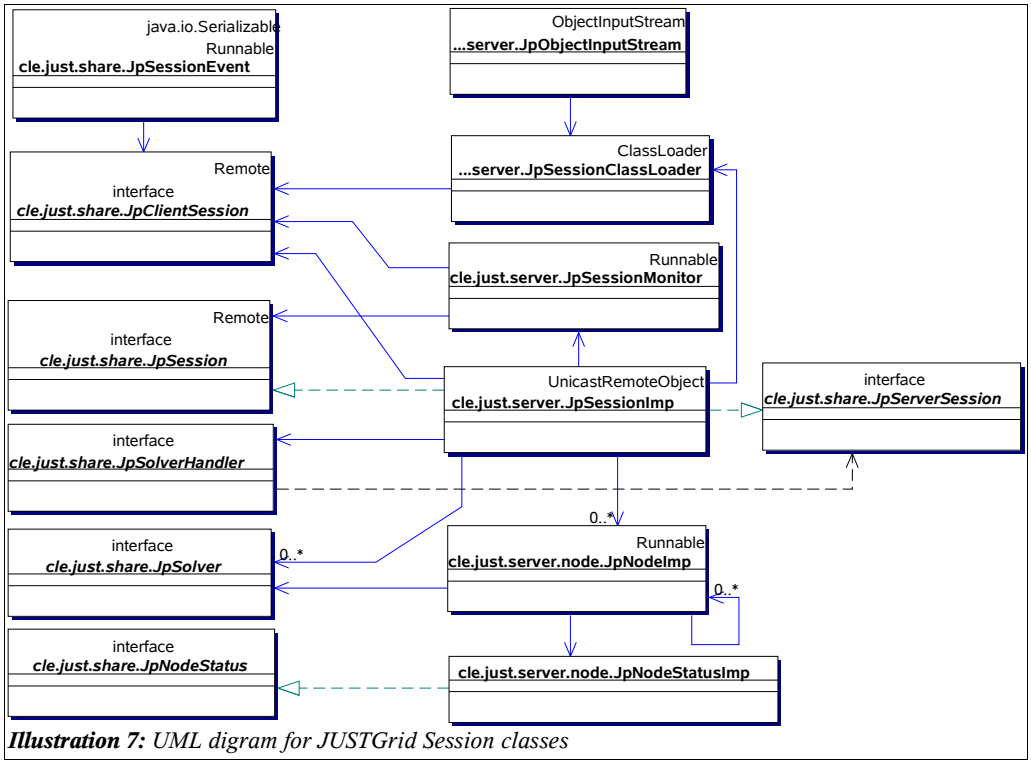


Illustration 7: *UML digram for JUSTGrid Session classes*

4.4 Multiblock Class Diagram

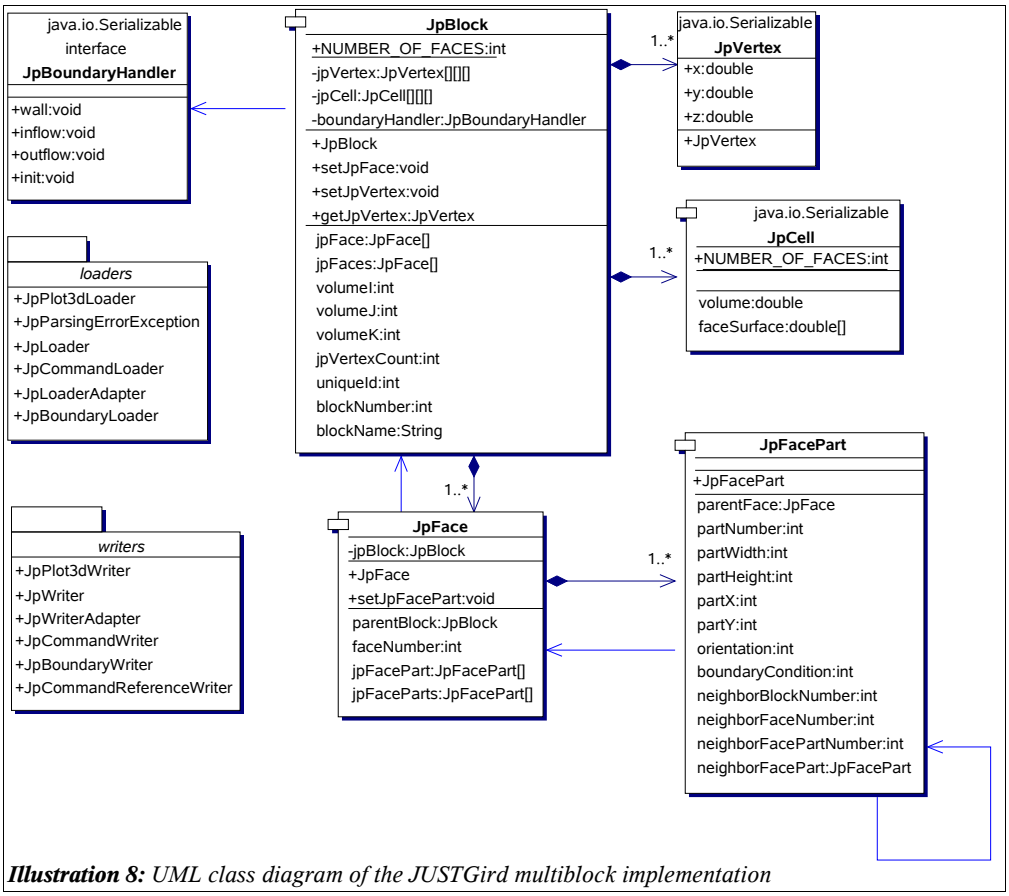


Illustration 8: *UML class diagram of the JUSTGird multiblock implementation*

The illustration 8 shows the class diagram of the JUSTGrid multiblock implementation including the loader- and writer- packages and its containing classes.

5 Threads

The thread concept as the basic parallelization strategy, delivers an unlimited number of options to speed up parallelization, since fine tuning by threads on all levels of parallelization (i.e, domain decomposition, numerical algorithm, loops etc.) of a computation is possible.

5.1 What are Threads?

Multithreaded programs extend the idea of multitasking one level further such that individual programs (processes) will appear to perform multiple tasks at the same time. Each task is called a *thread* which is the short form for thread of control. Programs that can run more than one thread at a time are said to be *multithreaded*. A thread consist of three parts : a virtual CPU, the code to be executed and the data the code works on. (see Illustration 9)

5.2 Threads in the JUSTGrid framework

In **JUSTGrid**, each subdomain is run within its own thread, and is completely independent of all other threads (macroscopic parallelization). The mapping of threads onto the set of processors as well as thread scheduling is entirely left to the underlying operating system.

The issues of static and of dynamic loadbalancing are not the concern of **JUSTGrid** running on a single SMP machine. The situation is different for cluster computing.

Within the thread for a singel subdomain, additional threads can be started, for instance to parallelize the numerical algorithm itself (microscopic parallization).

Illustration 12 in Section 6.1.1 on page 10 give a good impression about the small amount of scheduling overhead dealing with threads. A straightforward rule is to use a minimum of 4 threads per system processor but not to exceed than 512 threads per processor.

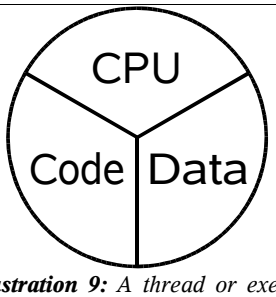


Illustration 9: *A thread or execution context*

6 Java Performance

6.1 Scaling

Every benchmark in the single threaded and also in the multi threaded benchmark was done 8 times in the same Java runtime environment.

6.1.1 Simple Numeric Benchmark

In this ultra-simple program, many identical threads are used for simple arithmetic computing multiplication's and division's. It is an embarrassingly parallel problem, meaning that the threads do not have to communicate, and thus there is no need for thread synchronization.

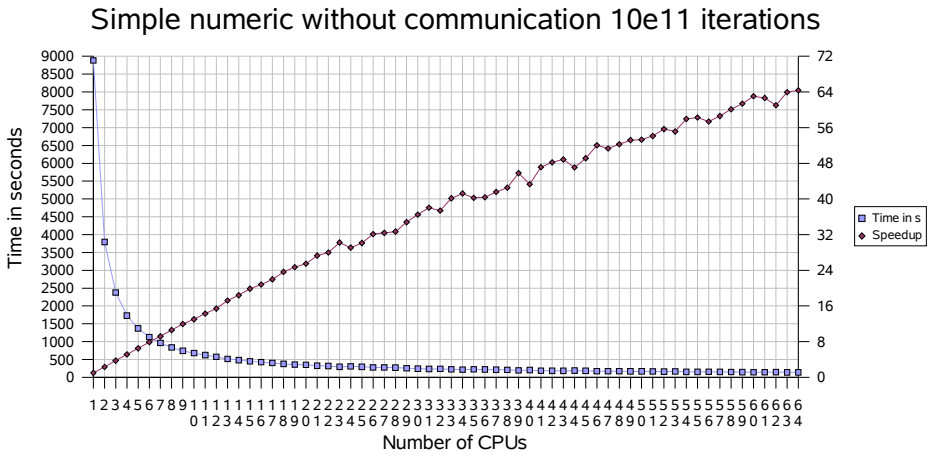


Illustration 10: *Simple numeric benchmark on a Sun Microsystems Enterprise 10000 with 64 UltraSPARC II CPUs and 256GB main memory.*

The code computes a fixed number of multiplication's and division's and it splits the work among a variable number of threads. These threads then are mapped to the processors by the operating system, relieving the user of the need to employing any kind of message passing library as well as a load balancing algorithm. The code runs on any kind of platform as long as a Java virtual machine is available.

The purpose of this code is to determine whether multi-threading produces a parallel (linear) speedup on the target machine.

The performance losses at about every 8 CPUs noticed in illustration 10 might be a behavior of the hardware architecture of the Sun Microsystems Enterprise 10000 server.

6.1.2 Multi-threaded Simple Numerics

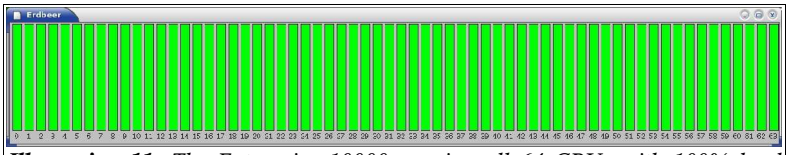


Illustration 11: *The Enterprise 10000 running all 64 CPUs with 100% load during the computation.*

Illustration 12 shows that even in case of some 512 threads per CPU the overall computing time rises only slightly

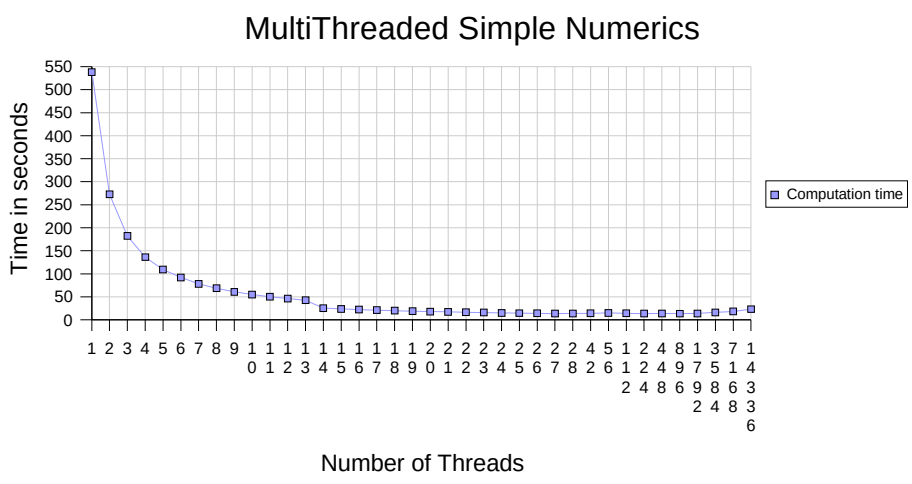


Illustration 12: *This benchmark shows the very small amount of overhead using threads. This benchmark was done on a Sun Microsystems Enterprise 6000 with 28 CPUs.*

6.2 Dynamic load balancing

6.2.1 Mandelbrot Benchmark

This code tests the self-scheduling of threads. Computation of the well-known Mandelbrot set utilizes a 2D grid in the complex plane, and an independent iterative calculation takes place at each grid point, where the number of iterations varies greatly from point to point. We partition the grid into blocks; we want each block large enough that thread overhead will be much less than the computational work associated with the block, and we want the blocks small enough that there are many blocks for each processor. As explained above, each processor takes a block from the pool, computes it, then gets another block.

Although this program is still embarrassingly parallel, it exhibits a new feature, namely the computational load depends on the position within the solution domain, which is a rectangle in this case. Dynamic load balancing would be needed to run such an application successfully on a large parallel architecture. Using PVM or MPI, the user has to provide a sophisticated algorithm to achieve this feature, requiring a lengthy piece of code. Using the Java thread concept, dynamic load balancing is provided by the operating system.

The parallel Mandelbrot concept is simple. There is a nonlinear mapping in a finite region of the complex plane (rectangular). Since the mapping can work on any finite region without interference to any other region being computed, the parallelization strategy is a simple 1D domain decomposition. This is as far as the idea goes. The final threaded code, however, shows a fair degree of complexity. This example can be used to demonstrate the effect of dynamic load balancing achieved by the Java thread concept.

Illustration 13 shows a linear speedup is achieved with that configuration.

6.3 Java vs. C++

6.3.1 Matrix multiplication

6.3.1.1 Sequential Matrix Multiplication

A sequential (1 thread) matrix multiplication using a 30 times 30 matrix doing 10000 iterations on a single processor Pentium 4 PC running Linux.

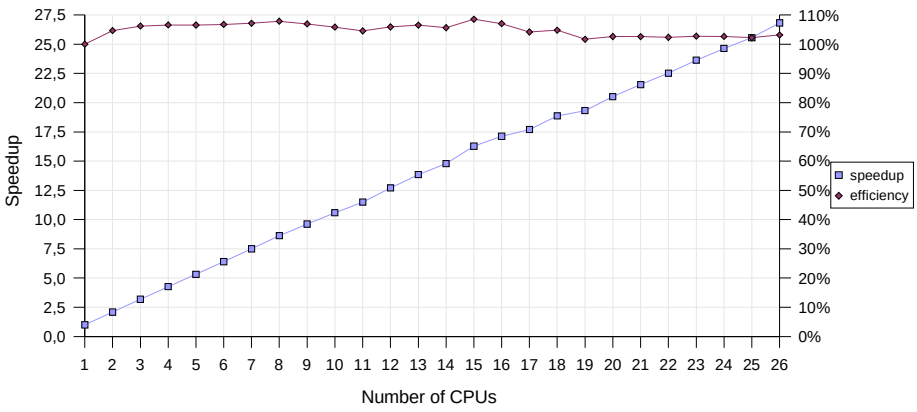


Illustration 13: *Mandelbrot Set dimension 7200 x 4800, max iterations 5000 running 400 threadson a Sun Microsystems Enterprise 6000 with 28 processors.*

Exactly the following source was used for both benchmarks. (C++ and Java)

```
// get start time here
  for( n=0; n<maxIterations; n++)
  {
    for( i=0; i<dim; i++ )
    {
      for( j=0; j<dim; j++ )
      {
        for( k=0; k<dim; k++ )
        {
          c[i][j] += a[i][k]*b[k][j];
        }
      }
    }
  }
// get end time here
```

`a` and `b` are the source and `c` the destination matrix. `dim` and `maxIterations` aren't constant variables so the compilers are not able to do an unroll loop optimization.

Runtime (2GHz Pentium 4, 1GB Memory)	1 run	2 run	3 run	4 run	5 run	6 run	7 run	8 run
GNU g++ -O3 -mcpu=pentium4 -march=pentium4 -Wall (Version 3.3.1)	3,15	3,19	3,22	3,16	3,15	3,17	3,16	3,16
Intel icc -O3 -mcpu=pentium4 -march=pentium4 (Version 8.0)	3,23	3,23	3,25	3,23	3,23	3,23	3,23	3,25
Sun Java HotSpot Client VM (Version 1.4.2_02-b03)	3,86	3,88	3,90	3,90	3,90	3,90	3,89	3,90
Sun Java HotSpot Server VM (Version 1.4.2_02-b03)	3,55	3,51	2,12	2,12	2,12	2,12	2,13	2,12

Table 2: A sequential (1 thread) matrix multiplication using a 30 times 30 matrix doing 10000 iterations on a single processor Pentium 4 PC running Linux.

The most important result of this bench is the enormous speed improvement after the two warmup phases in the Sun Java HotSpot Server VM. This runtime is about 1.5 times faster then the compiled C++ binary.

Due to a Linker error we could not use the `-fast` option with the Intel compiler.

6.3.1.2 Multithreaded Matrix Multiplication

Runtime	time in s
1.1.8_14	516,94
1.2.2_08	38,97
1.3.0_03 Server	37,47
1.3.1_02 Server	21,69
1.4.0_01 Server	19,51
1.4.1_02 Server	17,31
C++ - GCC	26,65
C++ - Forte 6u1	17,26

Table 3: Multithreaded matrix multiplication using a 100 times 100 matrix doing 10000 iterations with 400 threads on a 26 CPU Sun Microsystems Enterprise 6000.

In this configuration the C++ and the Java runtimes are on par.

7 Internet based HPCC and Security

These days we sometimes are talking about internet based HPCC as the holy grail for all of our HPC problems. But only a few people are talking about internet based SecureHPCC for our sensitive data.

Today the Java programming language with the Java Runtime Environment is the best way to get a SecureHPCC over the Internet. Java grows up with the Internet (World Wide Web) boom and it was developed always with the security problems in mind. Unlike the other popular programming languages ('C/C++', FORTRAN, ...) and development environments the overall system security was always one of Java's key features.

7.1 Java Language Security

Like even all OOP (Object-Oriented-Programming) languages Java has different access modifier/level (`private, default, public`) for data protection. Attributes that are declared as `final` must not be changed and Java does not have pointers like 'C/C++' with the possibility to access arbitrary data at any memory location.

7.2 Java Security Model

The Java 2 SDK security architecture is policy-based, and allows fine-grained access control. When code is loaded, it is assigned "permissions" based on the security policy currently in effect. Each permission specifies a special access to a particular resource, such as "read" and "write" access to a particular file or directory, or "connect" access to a given host and port.

7.3 Java Secure Socket Extension (JSSE)

The Java Secure Socket Extension (JSSE) enable secure Internet communications. It includes functionality for data encryption, server authentication, message integrity, and optional client authentication. A service can provide the secure passage of data between a client and a server running any application protocol using JSSE.

8 Results from JUSTGrid

Illustration 3 on page 6 shows a Lava Nozzle with the Euler 2D code.

As a reference sample to check the correct communication (boundary update) of the JUSTGrid we computed a 3D cone with the JUST Euler

3D solver and CFD++ shown in the Illustrations 15 and 16.

Illustration 14 shows a 3D online visualization (client) of a sphere during the computation with the JUST Euler 3D solver (server) demonstrating the high performance client server communication and visualization of **JUSTGrid**.

Another visualization with the JUSTGrid Virtual Visualization Toolkit (VVTK) is shown with Illustration 17 on page 14. It displays the surface of the EXTV the European eXperimental Test Vehicle.

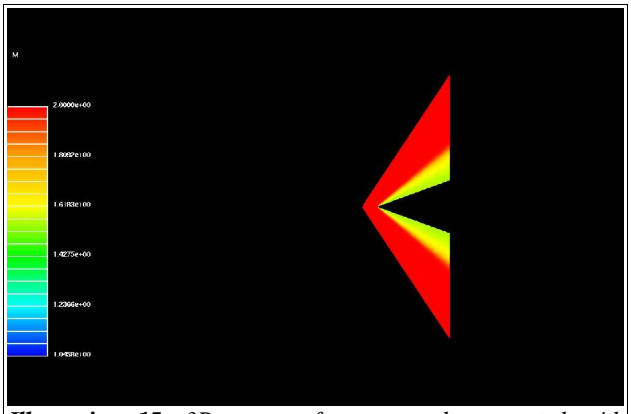


Illustration 15: *3D cone reference result computed with CFD++, Mach-number distribution, AoA 0, Mach 2.0,*

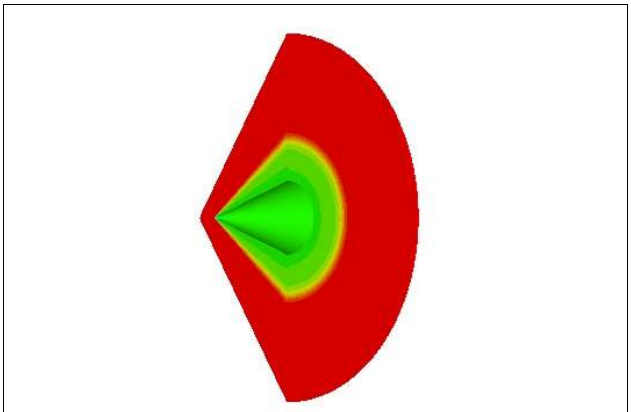


Illustration 16: *3D cone, computed with JUST euler 3D, Mach-number distribution, AoA 0, Mach 2.0*

9 Conclusions and Future Work

9.1 Achieved or shown

- With **JUSTGrid** a modern, well structured, easy to use and extensible framework for HPCC is provided.
- Collaborative engineering, online visualization and advanced security features are implemented in **JUSTGrid**.
- The code developer is freed from dealing with complex geometries, dynamic load balancing and inter block or domain communication.
- We obtain dynamic load balancing for free, if the CPU workload is high enough and the number of compute threads is about 4 times higher than the number of processors. These conditions can be easily fulfilled with our computational problems in mind.
- A numerical framework for a system of hyperbolic conservations laws is installed, based on the integral form of the conservation equations
- Today Java has become **THE** modern object oriented language for HPCC, providing excellence performance (at least as fast as C++) and outstanding parallelization features, as well as internet and security features already available in the core system.

9.2 TODO

- The communication layer of JUSTGrid has to be extended to cluster computing.

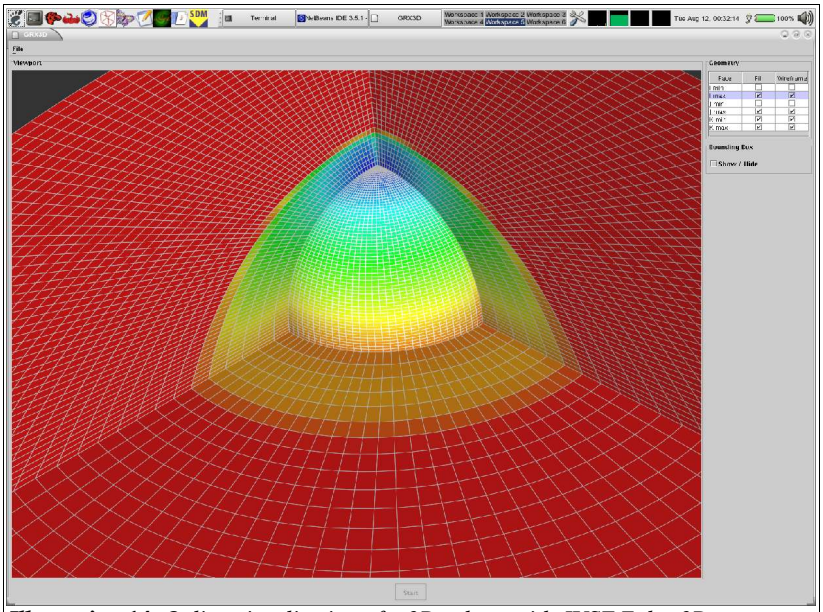


Illustration 14: *Online visualization of a 3D sphere with JUST Euler 3D*

Acknowledgments

This work was partly funded by Arbeitsgruppe Innovative Projekte (AGIP), Ministry of Science, Hanover, Germany under Efre contract

The authors are grateful to Profs. Mark Cross and Mayur Patel, University of Greenwich, London, U.K. for many stimulating discussions.

We are grateful to Mr. Jean Muylaert from ESA, ESTEC for a lot of information and discussion.

We are particularly grateful to Sun Microsystems, Benchmark Center, Germany for providing exclusive access to a 28 CPU Sun Enterprise 6000 server and a 64 CPU Sun Enterprise 10000 server.

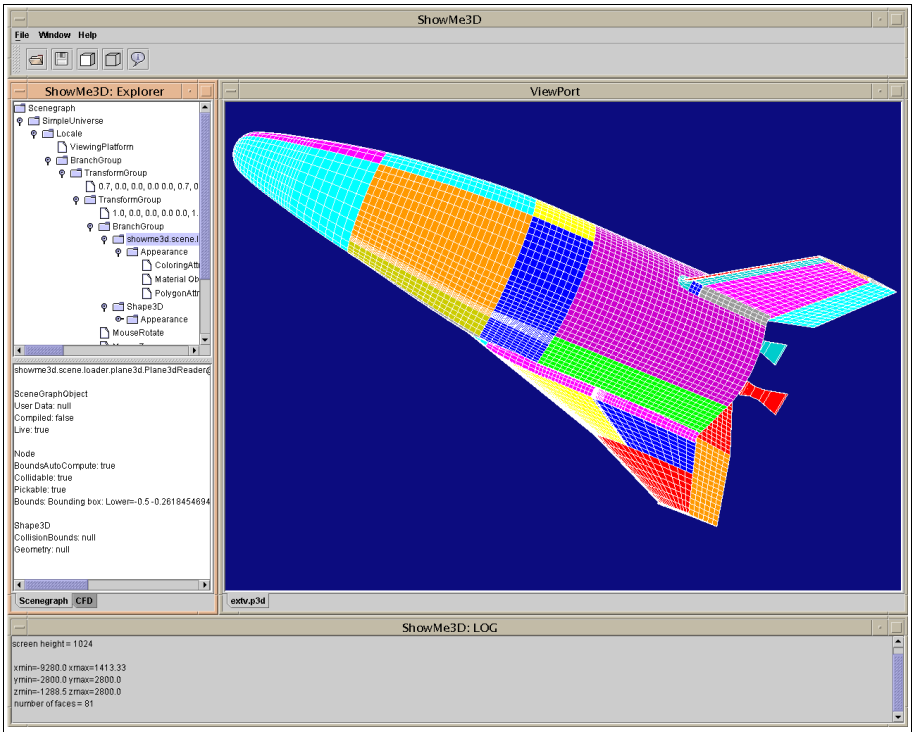


Illustration 17: *Surface visualization of the European eXperimental Test Vehicle (EXTV) with the JUSTGrid Virtual Visualization Toolkit (VVTK) formally known as ShowMe 3D*

References

[1] Tristram, Claire, *Supercomputing Resurrected*, MIT Technology Review, February 2003, pp. 54.

[2] PlayStation 3 chip nears completion, ZD Net UK News, http://news.zdnet.co.uk/story/0, ,t269-s2120395,00.html , August 2002.

[3] *The Need for Software*, Scientific Computing World, August-September 2000, Issue 54, pp.16.

[4] *Science and Technology Shaping the Twenty-First Century*, Executive Office of the President, Office of Science and technology Policy, 1997.

[5] Foster, Ian, *The Grid: Computing without Bounds*, Scientific American, April 2003, pp. 60-67 and Foster, Ian (ed.): *The Grid: Blueprint for a new Computing Infrastructure*, Morgan Kaufmann Publishers, 1999.

[6] Häuser, J., Ludewig, T., Gollnick, T., Williams, R.D.: *An innovative Software for HPCC.,* ECCOMAS 2001, Computational Fluid Dynamics Conference, Swansea, September 2001, UK

[7] Häuser, J., Ludewig, T., Williams, R.D., Winkelmann R., Gollnick T., Brunett S., Muylaert J.: *A Test Suite for High-Performance Parallel Java,* Advances in Engineering Software, 31 (2000), 687-696, Elsevier.

[8] Ginsberg, M., Häuser, J., Moreira, J.E., Morgan, R., Parsons, J.C., Wielenga, T.J. .: *Future Directions and Challenges for Java Implementations of Numeric-Intensive Industrial Applications*, 31 (2000), 743-751, Elsevier.

[9] Häuser, J., Ludewig, T., Gollnick, T., Winkelmann, R., Williams, R., D., Muylaert, J., Spel, M., *A Pure Java Parallel Flow Solver*, 37th AIAA Aerospace Sciences Meeting and Exhibit, AIAA 99-0549 Reno, NV, USA, 11-14 January 1999.

[10] Moreira, J.E., S. P. Midkiff, M. Gupta, From Flop to Megaflop: *Java for Technical Computing*, IBM Research Report RC 21166.

[11] Moreira, J.E., S. P. Midkiff, M. Gupta, *A Comparison of Java, C/C++, and Fortran for Numerical Computing*, IBM Research Report RC 21255.

[12] Häuser J., Williams R.D., *Strategies for Parallelizing a Navier-Stokes Code on the Intel Touchstone Machines*, Int. Journal for Numerical Methods in Fluids 15, pp. 51-58., John Wiley & Sons, June 1992.

[13] Winkelmann, R., Häuser J., Williams R.D, *Strategies for Parallel and Numerical Scalability of CFD* Codes, Comp. Meth. Appl. Mech. Engng., NH-Elsevier, 174, 433-456,1999.