Grid Computing Environments Community Practice (CP) Document

Project Title: Component-Based Problem Solving Environments for Large-Scale Scientific Computing

CP Document Contact: (include email): Chris Johnson (crj@cs.utah.edu), Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT, 84112, US

Project Participants: (include emails): Steve Parker (sparker@cs.utah.edu), David Weinstein (dmw@cs.utah.edu), Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, UT

Project URL(s): www.sci.utah.edu

1. Overview

SCIRun is a problem solving environment that allows the interactive construction, debugging, and steering of large-scale scientific computations. Over the past few years, we have developed two additional problem solving environments that extend SCIRun's capabilities: BioPSE and Uintah. The mission of the BioPSE project is to release state-of-the-art software, datasets, and documentation for researchers investigating bioelectric field problems. Uintah is designed to specifically address the problems of interdisciplinary, massively-parallel scientific computation on terascale computing platforms. These three systems, SCIRun, BioPSE, and Uintah, together target a broad range of vastly different problem domains and target users.

SCIRun provides a component model, based on dataflow programming, that allows various computational components and visualization components to be connected together. SCIRun facilitates the interactive construction, debugging and steering of large-scale, typically parallel, scientific computations. SCIRun can be envisioned as a ``computational workbench," in which a scientist can design and modify simulations

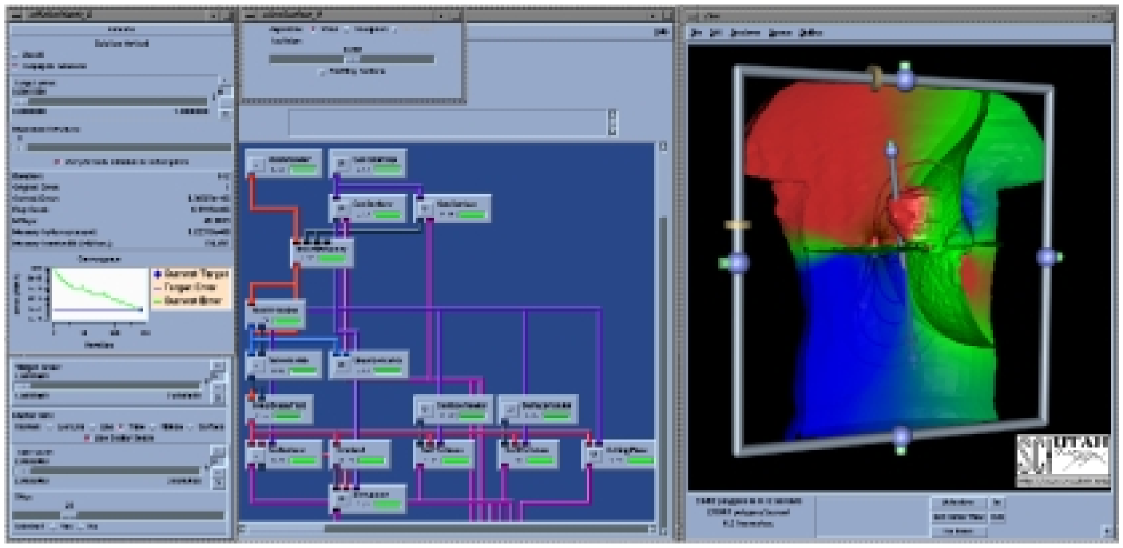


Figure 1. The SCIRun PSE showing the module network (middle), the visualization window (right). Researchers can select UI (user interaction) buttons on many of the modules that allow control and feedback of parameters within a particular module (left).

interactively via a component-based visual programming model. SCIRun enables scientists to modify geometric models and interactively change numerical parameters and boundary conditions, as well as to modify the level of mesh adaptation needed for an accurate numerical solution. As opposed to the typical ``off-line" simulation mode - in which the scientist manually sets input parameters, computes results, visualizes the results via a separate visualization package, then starts again at the beginning - SCIRun ``closes the loop" and allows interactive steering of the design, computation, and visualization phases of a simulation.

The BioPSE project seeks to release state-of-the-art software, datasets, and documentation for researchers investigating bioelectric field problems. The immediate applications of this software include forward and inverse bioelectric volume conductor problems, such as localizing active regions of the brain and imaging cardiac activation patterns. We are developing modeling, simulation, and visualization tools that will be of use to researchers engaged in disciplines ranging from cardiac defibrilation device design to cognitive mapping. Our NCRR Center is just completing its second year of funding, and we have just completed our second major software release of the BioPSE system.

Uintah is a Problem-Solving Environment that is being developed to support the University of Utah’s Center for Simulation of Accidental Fires and Explosions (C-SAFE). C-SAFE is funded under the Department of Energy’s Accelerated Strategic Computing Initiative (ASCI) program. The primary goal of C-SAFE focuses specifically on providing state-of-the-art, science-based tools for the numerical simulation of accidental fires and explosions, especially within the context of handling and storage of highly flammable materials. The objective of C-SAFE is to provide a system comprising a problem-solving environment (the Uintah PSE) in which fundamental chemistry and engineering physics are fully coupled with non-linear solvers, optimization, computational steering, visualization and experimental data verification. Uintah extends the capabilities of SCIRun with the addition of the component model being developed by the Department of Energy’s Common Component Architecture (CCA) Forum. While SCIRun and BioPSE currently target shared-memory parallel systems, Uintah uses the CCA component model in a distributed-memory environment for the simulation aspects of the project. Uintah components perform simulations running on hundreds to thousands of processors, computing tens to hundreds of gigabytes of data per simulation.

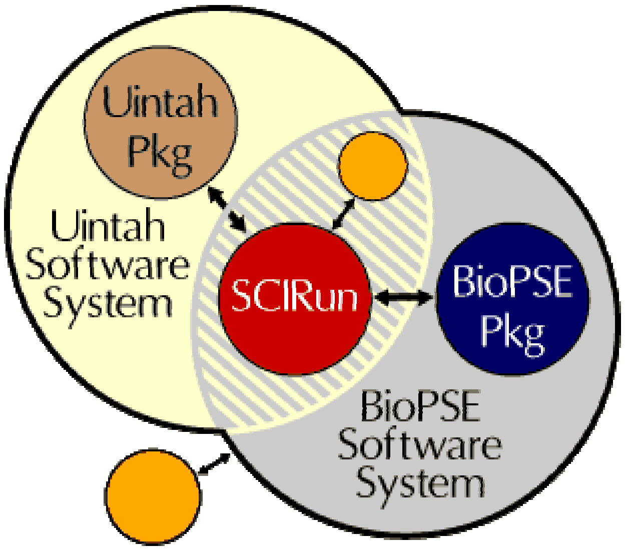


Figure 2. Relationship between the SCIRun, BioPSE, and Uintah software systems.

Each of these software systems are in active use by a wide range of users. SCIRun and BioPSE are publicly available at www.sci.utah.edu.

2. SCIRun, BioPSE, and Uintah Architectures

SCIRun

The SCIRun architecture [1-3] is the basis for a suite of scientific PSEs that allow the interactive construction, debugging, and steering of large-scale scientific computations. The primary purpose of the SCIRun architecture is to enable the user to interactively control scientific simulations while the computation is in progress. This control allows the user, for example, to vary boundary conditions, model geometries, and/or various computational parameters during simulation. Currently, many debugging systems provide this capability in a low-level form. SCIRun, on the other hand, provides high-level control over parameters in an efficient and intuitive way through graphical user interfaces and scientific visualization. These methods permit the scientist or engineer to ``close the loop" and use the visualization to steer phases of the computation.

The ability to steer a large-scale simulation provides many advantages to the scientific programmer. As changes in parameters become more instantaneous, the cause-effect relationships within the simulation become more evident, allowing the scientist to develop more intuition about the effect of problem parameters, to detect program bugs, to develop insight into the operation of an algorithm, or to deepen an understanding of the physics of the problem(s) being studied.

The SCIRun architecture is based on a model of components, each of of which perform some computational or visualization algorithm. The SCIRun component model is currently a dataflow-based programming model, with a visual programming interface. Data is also represented by a set of components, called the field classes. These components provide the ability to adapt data structures from an existing simulation to SCIRun's visualization components and many of the computational components. Uintah, described below, seeks to expand on the success of the component-based approach, by providing a component model based on the DOE Common Component Architecture, a model which is more familiar to programmers who have used the Object Management Group's (OMG) CORBA, or Microsoft's COM.

Initially, we designed SCIRun to solve specific problems in Computational Medicine, but we have made extensive efforts to make the SCIRun architecture applicable in other computational science and engineering problem domains [4]. In addressing specific problems, we found that there were a wide range of disparate demands placed on a steerable problem solving environment such as the SCIRun system. Steering (or interactive) control allows the user to vary boundary conditions, model geometries, or various computational parameters during simulation. Currently, many debugging systems provide this capability in a very raw, low-level form. SCIRun is designed to provide high-level control over parameters in an efficient and intuitive way, through graphical user interfaces and scientific visualization. These methods permit the scientist or engineer to ``close the loop" and use the visualization to steer phases of the computation.

The ability to steer a large-scale simulation provides many advantages to the scientific programmer. As changes in parameters become more instantaneous, the cause-effect relationships within the simulation become more evident, allowing the scientist to develop more intuition about theeffect of problem parameters, to detect program bugs, to develop insight into the operation of an algorithm, or to deepen an understanding of the physics of the problem(s) being studied. The scientific investigation process relies heavily on answers to a range of ``What if?" questions. Interaction allows these questions to be answered more efficiently and therefore to guide the investigation as it occurs.

Parallelism in SCIRun

SCIRun utilizes two methods of parallelism. The first, task parallelism, is implemented automatically by simultaneously executing multiple modules according to the dataflow graph. Since task parallelism is very limited in the typical scientific application, the second method of parallelism is to explicitly parallelize various modules in a data-parallel (SPMD) fashion. A set of worker threads will be mapped to various processors and will cooperate in accomplishing the function of the module. The worker threads may use the synchronization primitives provided by the Multitask library to communicate with one another.

The shared memory assumption allows for a simple, clean implementation of steerable parameters with low synchronization overhead in the normally running cases. As an example of parallelism, a simple data-parallel conjugate gradient matrix solver in SCIRun achieves a 90% parallel efficiency on 16 MIPS R10K (195 Mhz) processors, solving a 200,000 row sparse matrix with 3.2 million non-zeros in 32 seconds.

Components of SCIRun

In order to implement a steerable application, we have broken down SCIRun into a layered set of libraries. SCIRun uses an object oriented design; however, it should be stressed that we have paid careful attention to avoid over-using the object oriented paradigm to a point that efficiency suffers. In implementing the SCIRun kernel and modules, we leverage off of a powerful toolbox of C++ classes that have been tuned for scientific computing and operation in a multi-threaded environment.

SCIRun derives much of its flexibility from its internal use of threads. Threads allow multiple concurrent execution paths in a single program. SCIRun uses threads to facilitate parallel execution, to allow user interaction while computation is in progress, and to allow the system to change variables without interrupting a simulation. However, standards for implementing threads are only starting to appear, and the standards that are appearing are, thus far, cumbersome. We have constructed a layer that provides a simple, clean C++ interface to threads and provides abstraction from the actual standard used to implement them (currently pthreads and SGI sproc).

Data Structure Management

Many implementations of the dataflow paradigm use the port/connection structure to make copies of the data. Consider the example in Figure 3. If the vector field is copied to both the Hedgehog and Streamline modules, then twice as much memory is consumed as necessary. In addition, a significant amount of CPU time is required to copy large, complex data structures. To avoid these overheads, we employ a simple reference counting scheme with *smart pointers* in C++. This scheme helps reduce complexity by allowing different modules to share common data structures with copy-on-write semantics. When the Gradient module creates the VectorField, it sets a reference count in the vector field to zero. As Gradient hands a copy of the vector field data structure to each of the downstream modules, the receiving module increments the reference count. When each module is finished with the data structure, it decrements the reference count. When the reference count reaches zero, the object is destroyed. These reference counts are maintained automatically by C++ classes (the smart pointers) to reduce programmer error. Copying the object is necessary only when a module needs to change a data structure and the reference count is greater than one (i.e., another module is also using it).

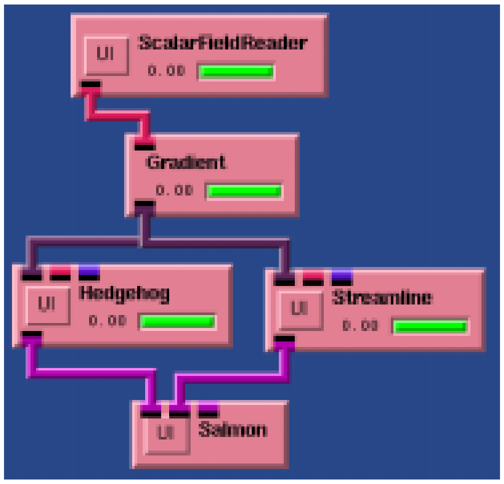


Figure 4. A closeup view of a dataflow network. A vector field, produced by the Gradient module, is consumed by both the Streamline and Hedgehog modules. In SCIRun, the data are shared between the modules so that the data do not need to be duplicated.

Progressive Refinement and Exploiting Coherence

Due to memory and speed limitations of current computing technologies, it will not always be possible to complete these large-scale computations at an interactive rate. To maintain some degree of dynamic interactivity, the system displays intermediate results as soon as they are available. Such results include partially converged iterative matrix solutions, partially adapted finite element grids, and incomplete streamlines or isosurfaces.

A common interactive change consists of moving and orienting portions of the geometry. Because of the nature of this interaction, surface movement is apt to be restricted to a small region of the domain. Using information available from both how the geometry has moved and its position prior to the move, the system can anticipate results and ``jump start'' many of the iterative methods. For example, iterative matrix solvers can be jump-started by interpolating the solution from the old geometry onto the new mesh. When changes to the model geometry are small, the resulting initial guess is close to the desired solution so the solver converges rapidly. This concept is similar to exploiting temporal coherence in a time-dependent system by using the previous time-step as the initial guess to the next time step. An even more compelling example is seen in the mesh generation. Typically, mesh generation for large-scale geometric models can take tens of minutes to hours to compute. Since it is sometimes the case the the user only wants to try a slight modification of the geometry or a new location for sources can modify and update the mesh only in places where it is needed in short amount of time.

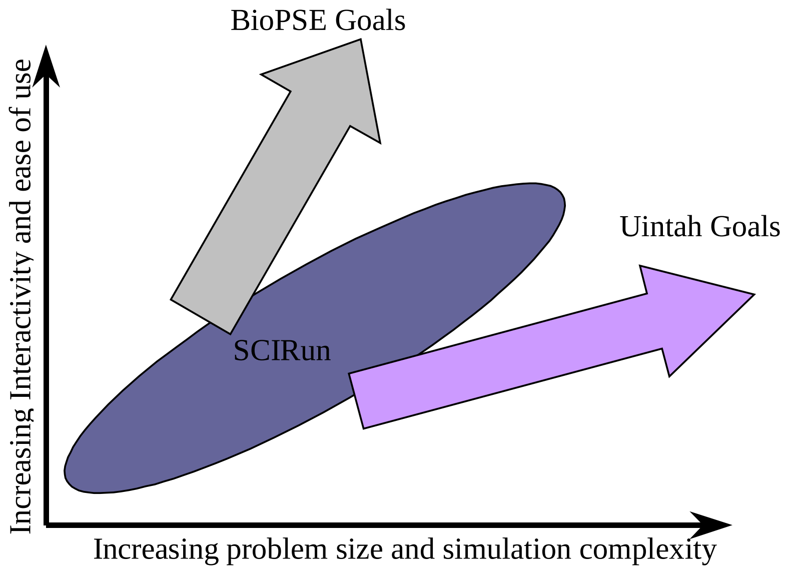
For most boundary value and initial value problems, the final answers will be the same for the incremental and brute-force approaches (subject to numerical tolerances). However, for nonlinear problems where there may be multiple solutions or for unsteady (parabolic) problems, results may be completely different. In these instances, the interaction coherence should not be exploited or results will not be scientifically repeatable.

Through coupling each of these techniques, we are able to introduce some degree of interactivity into a process that formerly took hours, days or even weeks. Although some of these techniques (such as displaying intermediate results) add to the computation time of the process, we attempt to compensate by providing optimizations (such as exploiting interaction coherence) that are not available with the old ``data file'' paradigm.

Steering and Interaction in a Dataflow System

The dataflow mechanism and the modules have been designed to support steering of large-scale scientific simulations. SCIRun uses three different methods to implement steering in this dataflow-oriented system:

- Direct lightweight parameter changes: A variable is connected to a user interface widget, and that variable is changed directly (by another thread) in the module. The iterative matrix solver module allows the user to change the target error even while the module is executing. This parameter change does not pass a new token through the dataflow network but simply changes the internal state of the SolveMatrix module, effectively changing the definition of the operator rather than triggering a new dataflow event.
- Cancellation: When parameters are changed, the module can choose to cancel the current operation. For example, if boundary conditions are changed, it may make sense to cancel the computation to focus on the new solution. This makes the most sense when solving elliptic problems, since the solution does not depend on any previous solution.
- Feedback loops in the dataflow program: For a time varying problem, the program usually goes through a time stepping loop with several major operations inside. The boundary conditions are integrated in one or more of these operations. If this loop is implemented in the dataflow system,


BioPSE Goals
Uintah Goals
SCIRun
Increasing Interactivity and ease of use
Increasing problem size and simulation complexity

then the user can make changes in those operators that are integrated on the next trip through the loop.

BioPSE

The BioPSE system is built on top of the SCIRun core architecture. With each release, we have improved the underlying SCIRun core and added support for a growing number of bioelectric field research problems. The BioPSE package (that part which is additional to SCIRun) consists of new modules and data-structures that are specifically customized for bioelectric field problems. These pieces include bioelectric field stiffness matrix construction, boundary condition assignment, and dipole source localization. While the bioelectric field modules are significant, they only represent a small percentage of the BioPSE project effort. The major undertaking of the BioPSE project has been to clean up and document the underlying SCIRun architecture. Through this process, we have redesigned and extended much of the SCIRun core. Some of these enhancements are described below.

Bridging is one of the newer design pillars of the SCIRun architecture. Philosophically, we have always believed that SCIRun would be most powerful and most useful if it supported flexible mechanisms for the importing and exporting of data and services. For this first release, we have introduced three levels of bridging. These bridges vary in how tightly they integrate the third party functionality. The tightest level of integration is what we call ``assimilation bridging'' --- we simply rewrite the existing data or functionality within the SCIRun system. This approach is the most time-intensive, and is demonstrated in our Focusing Inversion research project [5], where we have converted matlab code into C++ and embedded it within SCIRun. The second level is ``library bridging'' --- we make calls into third party libraries or wrap up third party data structures, and use their functionality within SCIRun modules. An example of library bridging has been to use netsolve for solving linear systems from SCIRun. We linked in the netsolve client libraries and added support for invoking various netsolve preconditioners and solvers to SCIRun [6]. The third level of bridging is ``I/O bridging'' --- we exchange data through files, sockets, or databases, and invoke external functionality through system calls. We have used I/O bridging with matlab, where we communicate via sockets and with a third-party database package to exchange data between SCIRun and matlab. We have also used it extensively in our development of filters for importing/exporting data between SCIRun and other systems.

A major piece of software infrastructure work was the improvement of the interface to the Fields classes. The Fields data structure is used to represent any geometric domain and a set of data values defined within that domain. The design challenge for Fields was to create a completely general and flexible data structure, without sacrificing system performance. Since Fields are the predominant primitive in our dataflow architecture, it was crucial that they be carefully designed to meet the various needs of our users. The results of this effort have been quite gratifying; the new Fields have greatly reduced the complexity of many of the existing modules, while simultaneously enhancing their functionality.

A common drawback to general interfaces is that generality typically comes at the price of performance. This is because in order to support a general interface, the code can not know in advance what type of Fields will be passed as input, and therefore it is impossible to optimize. In order to avoid this drawback, we designed an easy to use mechanism known as ``dynamic compilation and loading'' (DCL). The way this mechanism works is that as soon as a module finds out which specific type of Fields have been passed to it as input, it begins compiling optimized code for that specific case; when the code is compiled, this system loads that library and executes it. All of this happens transparently to the user, and has been designed to be simple to use for developers. In summary, the advantage to this technique is that we only generate specific optimized code for the types of fields in which we are interested.

In addition to the Field interface, we also recently redesigned our visualization pipeline. A typical visualization algorithm, such as Streamline advection, works by computing sample positions, evaluating the value of the Field at those positions, and creating a geometric representation for those values and positions. This three step process is common to many visualization methods: isosurfacing, streamlining, volume rendering, tensor field rendering, surface potential mapping, cutting plane rendering, etc. Typically, a particular visualization algorithm will implement all three of these steps itself. Such an approach results in substantial coding inefficiencies. For example, the same geometric representations may be of interest to multiple visualization techniques (e.g. rendering pseudo-colored surfaces is common to surface potential mapping, cutting plane rendering, and often isosurfacing). In the spirit of modular programming and reusable components, we have pipelined the majority of our visualization methods, with interchangeable modules available for each of the three stages.

In addition to the ``bridging'' and ``dynamic compilation and loading'' features described above, we have also added enhancements to the Network Editor, tools for importing and exporting data, and infrastructure for a single-SCIRun installation that can be shared within a facility by a collection of users.

The Network Editor was enhanced to support a full range of common File menu options: Save, Save As..., Insert, and Clear. These options have the standard behaviors and lend familiarity to the SCIRun user interface. We also added Network Editor features that will be most useful for debugging new Networks. These features allow the user to track individual pipes and ports on the network and to query the process ID of a specific module.

As mentioned above, new users are frequently interested in how to get their datasets loaded into SCIRun. We have begun implementation of a general import/export mechanism that will allow a user to add a simple conversion function in order to be able to filter their data into and out of SCIRun. The conversion function can be dynamically compiled and linked into SCIRun, much in the way that a ``plug-in'' is loaded in other software.

The final major enhancement that the NCRR project has contributed to the SCIRun core has been to test and release a central, shared version of SCIRun. That is, a centrally-located, installed version of SCIRun that many users can work from at the same time. The challenge for this project was to provide individual users with their own separate space for developing new datatypes and modules, while allowing them to link against the central SCIRun installation. We implemented this by extending the Package facility to support disjoint compilation. In our own lab we now have students collaborating through the central SCIRun installation, and we anticipate this will be useful functionality at many other labs as well.

BioPSE and the underlying SCIRun core have matured significantly through the development efforts of the NCRR Center. We presently have dozens of users, and anticipate that number will rise rapidly as we continue to add bioelectric field and infrastructure functionality. Enhancing the SCIRun core will continue to be a priority of the NCRR project for the foreseeable future.

For bioelectric field simulations, we will be adding transient visualization methods, and a growing array of inverse techniques. And we will continue adding surface and volume mesh generation tools into the system.

In terms of general enhancements to the SCIRun architecture, we will first be designing and implementing state and event management tools. These will serve as a stepping stone for our subsequent development of collaborative computing, scripting, and checkpointing capabilities. We will also be implementing multi-resolution modeling, simulation, and visualization techniques that we believe will be of general use to researchers working with large-scale data.

Uintah

C-SAFE was create by the Department of Energy's (DOE) Accelerated Strategic Computing Initiative's (ASCI) Academic Strategic Alliance Program (ASAP). The purpose of ASAP is to “engage the best minds in the U.S. academic community to help accelerate the emergence of new unclassified simulation science and methodology and associated supporting technology for high-performance computer modeling and Simulation [10].” C-SAFE, located on the University of Utah campus, “is focused on providing state-of-the-art, science-based tools for the numerical simulation of accidental fires and explosions.” C-SAFE is staffed by twenty key University of Utah faculty, faculty from BYU and WPI, experimental scientists from Thiokol, eighteen post doctoral/professionals, and 14 graduate students. These personnel are divided into six teams: Fire Spread, Container Dynamics, High Energy Transformations, Applied Math, Computer Science, and Validation. It is the combined strengths of these assorted disciplines that are allowing C-SAFE to create a software system that will accurately model the physical processes in our simulations while taking advantage of massively parallel computers to solve the problems in a reasonable amount of time.

Within C-SAFE, it is the Computer Science team's task to architect and implement a software system that will provide the ability to run large simulations on massively parallel computers. To fulfill this mission, we are in the process of designing and implementing the Uintah PSE. Uintah is based upon the SCIRun PSE architecture, utilizing the visual dataflow programming paradigm, but has helped extend the SCIRun architecture in two important manners: Uintah has added the Common Component Architecture's (CCA) interchangeable component programming model to SCIRun, and has architected support for running under a mixed shared memory/message passing model [7]. These additions to SCIRun have paved the way in making Uintah a scalable, high-performance system that is capable of solving large scale, very complex scientific problems.

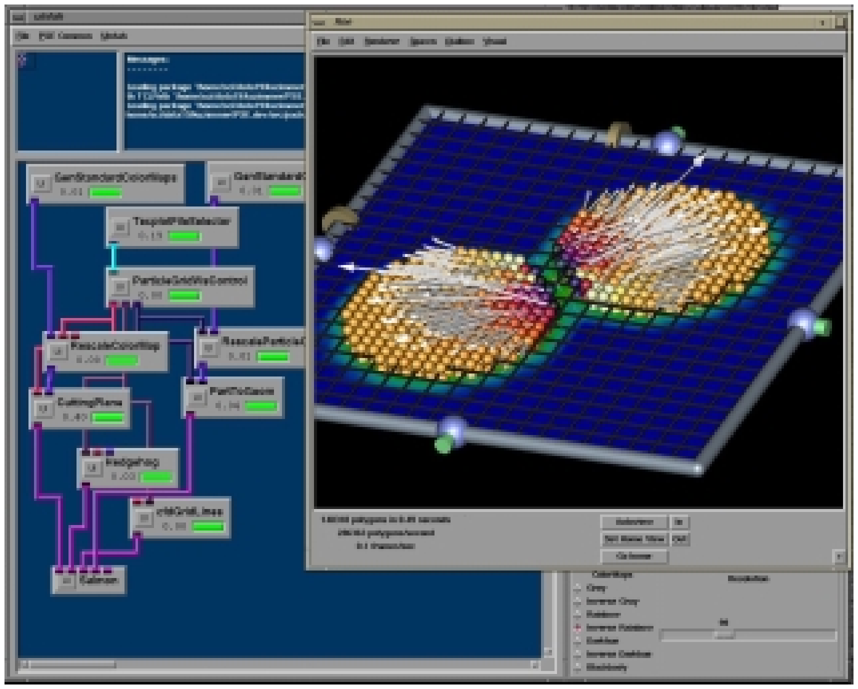


Figure 5. Two disks colliding simulated using a material point computational method within the Uintah PSE.

C-SAFE has set a number of ambitious software goals that are intended to push the envelope of large scale scientific computing. In order to accomplish the extremely complex goal of accurately simulating a large scale file engulfing complex materials, the simulation must take into account processes ranging from the atomic level to tens of meters, and time steps from seconds down to femtoseconds. This process is an active collaboration between chemists, mechanical engineers, chemical engineers, and computer scientists; each playing an important and necessary role in creating a unified simulation system. The C-SAFE system, named Uintah, is comprised of a number of software pieces: Blazer (a Scientific Data Management System), Uintah PSE (discussed in detail in this paper), and independent applications that produce the necessary data for the Uintah PSE to accurately perform the simulations.

An important criteria for creating a software system that will effectively utilize large numbers of CPUs working in concert is the development of a parallelization strategy that encompasses both shared memory and MPI style message passing, allowing PSE based simulations to run on a number of different computer architectures.

Another important criterion taken into account in the design of the Uintah PSE is the ability to remotely create and monitor simulations on large, geographically remote, super computers. Grid services, such as remote communication and authentication are essential for these features. While these features are still under development, we are currently using the Globus toolkit to develop these services.

A third criterion driving the design of the Uintah PSE is the ability to easily allow users to investigate the effect of different routines on a simulation and to integrate other software sub-systems into the PSE. This ability is provided by adhering to the Common Component Architecture (CCA) Model being developed by a consortium of National Lab, Industry, and Academic scientists. When complete, this technology will raise the bar from code reuse, to working component reuse.

The current trend in super computing is to use a large number of full-function processors, grouped in shared memory nodes of 2 to 128 processors. These nodes communicate with each other using high speed data interconnects. To take full advantage of this type of architecture, processes running on a single node must take advantage of the benefits of shared memory to reduce communication overhead and problem complexity.

3. Implementation

The combination of SCIRun, Uintah, and BioPSE consist of over 500,000 lines of C++ code. In addition, the user interface is written in Tcl/Tk, and various components use C and Fortran code. XML is used throughout, for describing components to the SCIRun scheduler, and for representing data input and output in Uintah. In addition, we rely on a whole host of commodity software, such as PETSc from Argonne National Laboratory, LAPACK, BLAS, OpenGL from SGI, pthreads from Posix, Apache's Xerces XML toolkit, GNU autoconf., GNU make, and Tuning and Analysis Utilitites (TAU) from the University of Oregon.

The core of Uintah is centered around the DOE Common Component Architecture (CCA) component model. The CCA specification is still evolving but Uintah uses CCA components to implement different aspects of the simulation. Example components are Computational Fluid Dynamics mdels, Solid Modeling models, Checmical Reaction models, Parallel schedulers, and load balancers.

As previously mentioned, SCIRun is currently publicly available as open-source software. We intend to make Uintah available in the same manner at some point in the future. As much as possible these systems are built on public standards, leverage existing third party software.

4. Supported Grid Services

SCIRun and its derivatives are currently moving from a standalone program to a more distributed programming model. This transition involves adapting grid services to perform many functions that programmers take for granted on a single shared-memory parallel machine.

SCIRun, as with most problem solving environments, has a great potential to benefit from most of the services offered by the grid. To date, we have utilized many grid services in different aspects of the implementation. SCIRun has used Netsolve [6,7,9] to provide a mechanism for distributing linear algebra components. Uintah utilizes Globus I/O (formerly Nexus), a part of the Globus toolkit [9], to provide inter-component communication for distributed-memory components. Uintah provides Blazer, a mechanism for indexing and managing large datasets via the world-wide web.

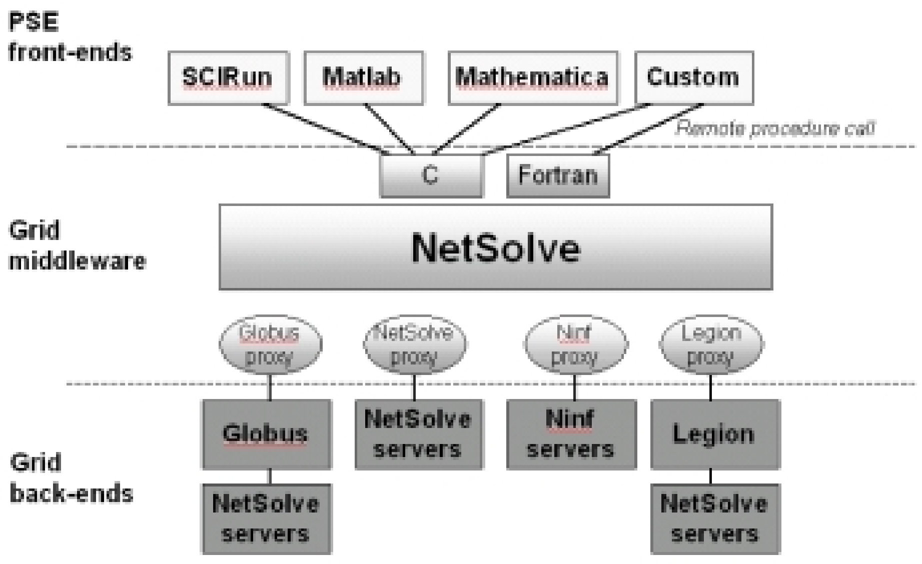


Figure 6. Possible relationships between application problem solving environments, such as SCIRun, with Grid middleware, such as NetSolve, with Grid back-ends.

Other ongoing projects in SCIRun, Uintah, and BioPSE are considering use of grid services for other applications. Uintah would benefit from the security and scheduling capabilities to implement component staging and scheduling. BioPSE is examining possibilities to utilize microscopes, magnetic resonance machines, and other large-scale instruments via the web.

As Problem Solving Environments become more comprehensive, more distributed, and undoubtedly more complex, grid services become an essential part of the system. Standardization and wide-spread adoption of these services continue to be the key ingredients for successful application of grid services in a PSE.

However, we note that many high level applications are not readily able to utilize grid services because of the significant difficulty in integrating grid services with complex applications. There seems to be a "gap" between low-level grid services and large-scale user applications and front ends. It seems to us that there needs to be an effort in creating software tools that allow application programs to more easily take advantage of grid services. As we have recently created a "module maker" that helps enable users to import their own codes within SCIRun and BioPSE, we could envision a similar software tool for helping users import and integrate grid software tools within application codes.

5. Project Status and Future Plans

SCIRun, BioPSE, and Uintah are all long-term development projects that support a wide variety of research. Many of the ongoing research projects will require further integration of grid computing services.

Some of the key apsects include:

- Completion of the CCA-based component model: This will include support for component staging, management of available resources, and parallel-to-parallel component communication. In addition, we are working to separate the SCIRun graphical user interface, and will use it to provide visual programming capabilities for a number of different

- Implementation of simulation components for clusters: Clusters are becoming a popular mechanism for providing inexpensive computing cycles. In addition to the efforts underway in Uintah to provide mechanisms for large distributed-memory supercomputers, BioPSE intends to provide components that will utilize small to medium clusters of inexpensive personal computers.

- Remote Visualization: Often, the computing resources used to perform a simulation are not located physically near the scientists. In order to view the data, the scientist typically chooses to transport the data to his/her desktop workstation or local visualization workstation. However, datasets often overwhelm local resources. We are attempting to bridge this distance gap and are developing components for generating visualizations on remote computing resources, and using advanced methods in computer graphics to transmit those visualizations back to a remotely located desktop workstation in an interactive manner.

- Cluster-based visualization: In the same manner that Beowulf clusters have revolutionized the simulation community by providing low-cost mechanisms for obtaining computing cycles, we expect that the same revolution to apply to visualization capabilities using commodity 3D video cards. However, the software required to harness the power of multiple video cards is not mature. We have several ongoing projects that are developing components for cluster-based parallel computer graphics.

We look forward to working with the grid services community to help develop software systems that can benefit computational science and engineering end users.

6. Acknowledgments

This work was supported by grants from the Department of Energy, National Institutes of Health, and the National Science Foundation.

7. References

[1] S.G. Parker, D.M. Weinstein, and C.R. Johnson. “The SCIRun computational steering software system,” **Modern Software Tools in Scientific Computing**, pp. 1-40, 1997. edited by E. Arge, A. M. Bruaset and H. P. Langtangen. Birkhauser Press.

[2] S.G. Parker, M. Miller, C.D. Hansen, and C.R. Johnson. “An Integrated Problem Solving Environment: The SCIRun Computational Steering System”. *31st Hawaii International Conference on System Sciences (HICSS-31)*, volume VII, pp. 147-156, Jan. 1998.

[3] C. Johnson and S. Parker. “The SCIRun Parallel Scientific Computing Problem Solving Environment”, extended abstract, *Ninth SIAM Conference on Parallel Processing for Scientific Computing* , 2 pages, 1999.

[4] C. Johnson, S. Parker, and D. Weinstein “Large-Scale Computational Science Applications Using the SCIRun Problem Solving Environment”, in *Supercomputer 2000*.

[5] O. Portniaguine, D. Weinstein and C. Johnson. “Focusing Inversion of Electroencephalography and Magnetoencephalography Data,” To appear at the *3rd International Symposium On Noninvasive Functional Source Imaging*, Sept. 6-9, 2001, Innsbruck, Austria and published in a special issue of the *Journal of Biomedizinische Technik*, 2001.

[6] M. Miller, C. Moulding, J. Dongarra, C.R. Johnson “Grid-enabling Problem Solving Environments: A Case Study of SCIRun and Netsolve,” (to appear in the *Proceedings of HPC 2001*).

[7] J. McCorquodale, D. de St. Germain, S. Parker, and C.R. Johnson. “The Uintah Parallelism Infrastructure: A Performance Evaluation on the SGI Origin 2000”, *High Performance Computing 2001*, Seattle, March 2001.

[8] M. Miller and C. Hansen, and C. R. Johnson. “Simulation Steering with SCIRun in a Distributed Environment”, in Applied Parallel Computing, 4th International Workshop, PARA'98, B. Kagstrom, J. Dongarra, E. Elmroth, and J. Wasniewski, eds., Lecture Notes in Computer Science, Springer-Verlag, 1541, pp. 366-376, 1998.

[9] H. Casanova, J. Dongarra, C.R. Johnson, and M. Miller, Application-specific Toolkits, In **The Grid**, I. Foster and C. Kesselman, eds., Morgan Kauffman, pp. 159-180, 1999.

[10] DOE ASCI website: www.llnl.gov/asci-alliances