Large-scale Computational Science Applications using the SCIRun Problem Solving Environment

Christopher R. Johnson, Steven G. Parker, and David Weinstein
Scientific Computing and Imaging Institute
School of Computing
University of Utah
Salt Lake City, Utah 84112
Email: crj@cs.utah.edu, sparker@cs.utah.edu and dmw@cs.utah.edu
URL: www.sci.utah.edu

Abstract

In this paper we describe applications of the SCIRun system to large-scale computational science problems. SCIRun is a scientific problem solving environment that allows the interactive construction and steering of large-scale scientific computations. A scientific application is constructed by connecting computational elements (modules) to form a program (network). This program may contain several computational elements as well as several visualization elements, all of which work together in orchestrating a solution to a scientific problem. Geometric inputs and computational parameters may be changed interactively, and the results of these changes provide immediate feedback to the investigator.

Introduction

Five years ago at Supercomputer ‘95, I had the privilege to give the keynote address, celebrating the 10 year anniversary of the Supercomputer Conference in Mannheim. At the conference, I introduced a new scientific problem solving environment called SCIRun to solve large-scale problems in computational medicine [1]. Since that time, many papers on SCIRun have been published [2, 3, 4, 5] and other researchers have started to use SCIRun [6, 7, 8]. These new applications range from large-scale atmospheric diffusion simulations, to problems in optical tomography, reservoir simulation and visualization, and new visualization techniques for vector fields using haptic (force) feedback. Furthermore, SCIRun is being used as the basis for new problem solving environments for the DOE ASCI project [9] and an NIH NCRR Center [10].

In the Supercomputer '95 paper, we provided the following Summary and Future Work section:

> We have presented an overview of SCIRun applied to large scale problems in computational medicine. SCIRun is a computational steering model that allows users to "close the loop" and utilize visualization to guide (steer) the design and computation phases of a simulation. We are continuing to develop, optimize, and add flexibility to current features of our steering model. We are also adapting the computational steering model to a number of applications, including solving problems in inverse ECG and EEG, providing interactive surgery assistance for problems in neurosurgery, and addressing problems in computational fluid dynamics and environmental science.
>
> Currently the system runs on Silicon Graphics computers, using shared memory hardware to communicate between processors in a multi-processor environment. Work is under way to port the system to new architectures. In particular, support for parallelism on workstation clusters will be integrated into the system. The dataflow network can map directly to a network of processors or machines (one or more modules per processor). Minimizing the network traffic and communication overhead for a particular dataflow network and machine configuration is an interesting problem that this extension will attempt to address.

This seems like an ideal place to begin our Supercomputer '2000 contribution. In this paper we will given an updated overview of the current SCIRun problem solving environment and discuss new large-scale applications that are being solving using SCIRun.

Overview of the SCIRun Problem Solving Environment

SCIRun is a parallel scientific problem solving environment that allows 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 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 dataflow programming paradigm has proven useful in many applications. We have extended the use of the dataflow programming model into the computational pieces of the simulation. To make the dataflow programming paradigm applicable to large scientific problems, we have identified ways to avoid the excessive memory use inherent in standard dataflow implementations, and we have implemented fine-grained dataflow in order to further promote computational efficiency.

Interactive Computing and Steering

A primary feature of SCIRun enables the user to interactively control scientific simulations while the computation is in progress. This 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 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 scientific investigation process relies heavily on answers to a range of "What if?" questions. Computational steering 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).

Steering Optimizations

To accommodate the large datasets required by high resolution computational models, we have optimized and streamlined the dataflow implementation. These optimizations are made necessary by the limitations many scientists have experienced with currently available dataflow visualization systems [11].

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 1. 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* [12] 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).

Progressive Refinement

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. In the defibrillator design example shown above, the user moves an electrode and sees some feedback almost immediately. The solution continues to converge to the final

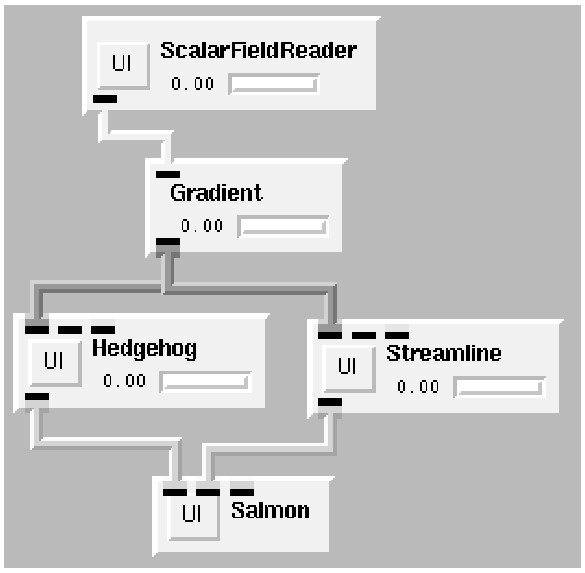


Figure 1: A closeup view of a dataflow network. A vector field, produced by the Gradient module, is consumed by both the Streamline and Hedgehog modules. In SCIRun, the data are shared between the modules so that the data do not need to be duplicated.

solution. In this way, an engineer or scientist can watch a solution converge and, based on the results observed, may either decide to make changes and start over or allow the simulation to continue to full convergence.

Exploiting Interaction Coherence

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 [13]. 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 process for the torso defibrillator modeling problem. Typically, mesh generation for the entire torso model would take tens of minutes to hours. Since we know that the user only wants to move the defibrillator electrodes, we generate the mesh without the electrodes beforehand. Then, when the user selects an electrode placement, nodes for the defibrillator electrodes are placed into the mesh in only a few seconds.

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 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:

1. **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

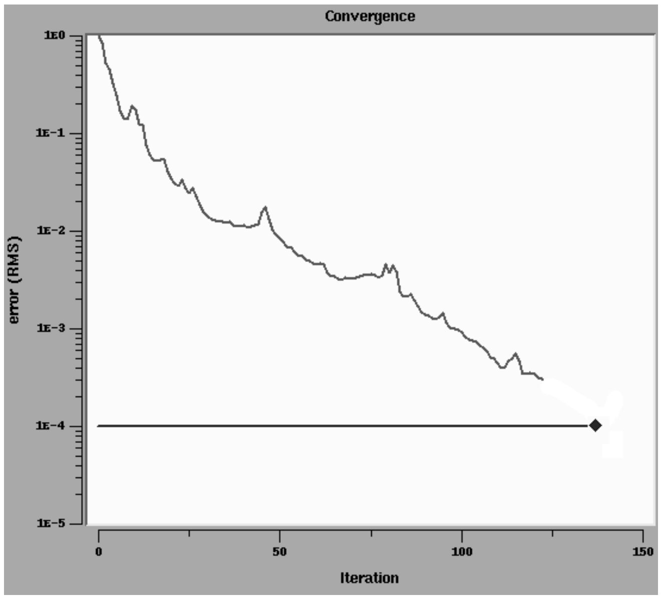


Figure 2: A portion user interface for the SolveMatrix module. The user can change the target residual by moving the small diamond on the graph. This is an example of a direct lightweight parameter change.

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. The interface for SolveMatrix is shown in Figure 2.

2. **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.

3. **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, then the user can make changes in those operators that are integrated on the next trip through the loop. An example of this is shown in Figure 3, where the user has created an adaptive finite element solution method. The results of one solution are used to estimate the solution error, and then the mesh is refined in areas of high error. Another so-

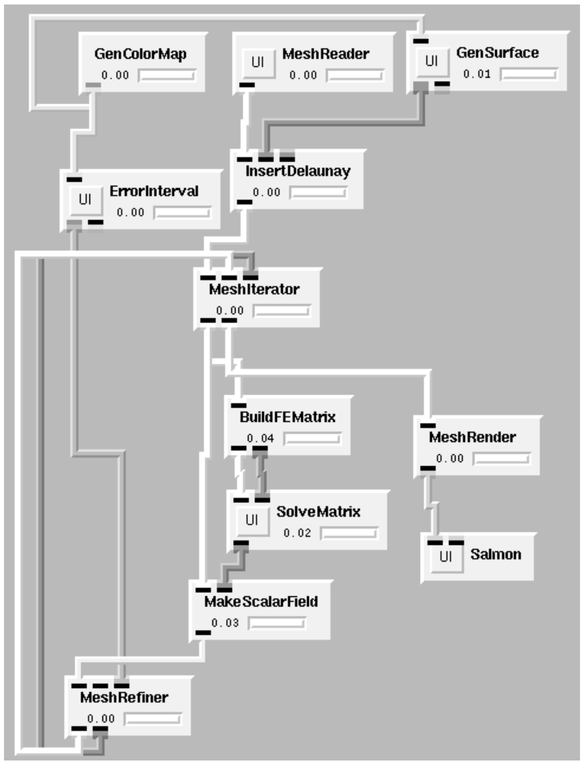


Figure 3: Demonstration of steering through a feedback loop in the dataflow network. Data flows beginning with an initial Mesh (generated in InsertDelaunay), performs a computation (BuildFEMatrix and SolveMatrix), refines the mesh according to an error estimate (MeshRefiner), and continues back to MeshIterator. The MeshIterator module will continue iterating the loop until the MeshRefiner module declares that no more adaptation is necessary. The final mesh is rendered using the MeshRender module.

lution is computed, and the process is repeated until a target error level has been reached. The user can change the mesh adaptation criteria shown on the left, but the new values are not used until the next iteration of the loop.

SCIRun as a Scientific Library

SCIRun uses a visual programming interface to allow the scientist to construct simulations through powerful computational components. While the visual programming environment is the central focus of SCIRun, it requires a powerful set of computational tools. In the first stage of SCIRun, we have concentrated on integrating the computational components that we have used to solve our own computational problems. Such tools Delaunay 3D tetrahedral mesh generators and mesh adaptation routines, direct and iterative linear and nonlinear equations solvers and finite element space discretisation routines, see [3]. We have recently expanded focus and are now in the process of integrating popular libraries and tools, see [3] into the SCIRun environment.

The SCIRun Development Environment

Perhaps the most powerful facet of SCIRun is the ability to use it in the development phases of a simulation. SCIRun augments the development environment by providing convenient access to a powerful set of computational components. However, these components could never be comprehensive, so SCIRun also provides an environment whereby new modules can be developed efficiently. If a module triggers a segmentation violation, bus error or failed assertion, SCIRun stops the module at the point of error, thus allowing the developer to attach a debugger to the program at the point of failure. This avoids the frustrating experience of trying to reproduce these errors in the debugger. In addition, SCIRun provides simple instrumentation of module performance (CPU times printed out interactively), feedback execution states (waiting for data, percent completed, etc.), and visualization of memory usage. SCIRun employs dynamic shared libraries to allow the user to recompile only a specific module without the expense of a complete re-link. Another SCIRun window contains an interactive prompt which gives the user access to a Tcl shell that can be used to interactively query and change parameters in the simulation.

Requirements of the Application

SCIRun is not magic – it is simply a powerful, expressive environment for constructing steerable applications, either from existing applications or starting from the ground-up. The application programmer must assume the responsibility of breaking up an application into suitable components. In practice, this modularization is already present inside most codes, since “modular programming” has been preached by software engineers as a sensible programming style for years.

More importantly, it is the responsibility of the application programmer to ensure that parameter changes make sense with regard to the underlying physics of the problem. In a CFD simulation, for example, it is not physically possible for a boundary to move within a single timestep without a dramatic impact on the flow. The application programmer may be better off allowing the user to apply forces to a boundary that would move the boundary in a physically coherent manner. Alternatively, the user could be warned that moving a boundary in a non-physical manner would cause gross errors in the transient solution.

Computational Applications

Atmospheric Diffusion Simulation

The first computational application we consider is taken from a model of atmospheric dispersion from a power station plume - a concentrated source of NO_x emissions, [14]. The photo-chemical reaction of this NO_x with polluted air leads to the generation of ozone at large distances downwind from the source. An accurate description of the distribution of pollutant concentrations is needed over large spatial regions in order to compare with field measurement calculations. The present trend is to use models incorporating an ever larger number of reactions and chemical species in the atmospheric chemistry model. The complex chemical kinetics in the atmospheric model gives rise to abrupt and sudden changes in both space and time in the concentration of the chemical species in both space and time. These changes must be matched by changes in the spatial mesh and the timesteps if high resolution is required, [15]. The difference in time-scale between the reaction of these species leads to stiff systems of equations which require implicit numerical solvers and special linear equations solvers [16]. The requirements of such a problem are that it is necessary to combine:

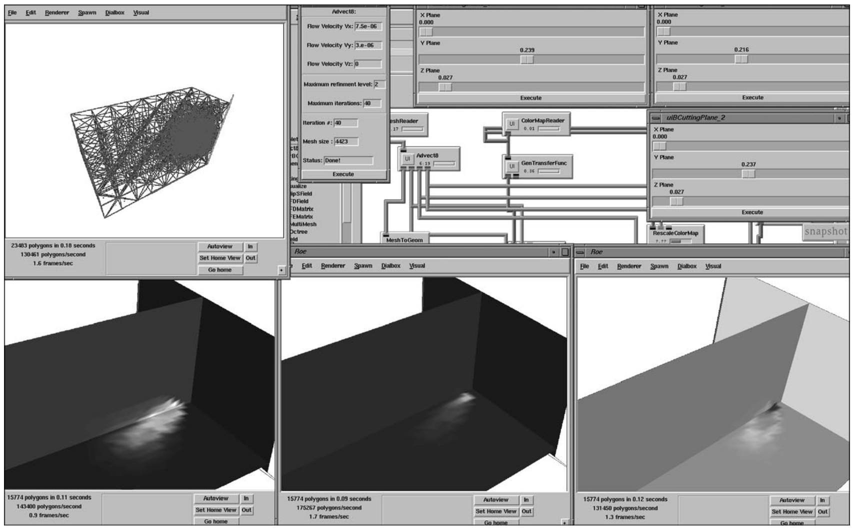
- Unstructured tetrahedral mesh generation and adaptation.
- Physically realistic spatial discretisation methods.
- Stiff ode integrators tailored to the application.
- Fast interactive visualization for multi-species flows
- Computational steering facilities for transient problems.

These requirements were met by combining the SCIRun software with the spatial discretisation, mesh adaptation and time integration codes CSPRINT and TETRAD [16, 17] and by wrapping these codes in SCIRun modules, with converters to map to the SCIRun data structures [6].

SCIRun composes computational and visualization algorithms with these data elements using a dataflow style “boxes and wires” approach. An example of the atmospheric diffusion dataflow network using Tetrad is shown in Figure 4.

Atmospheric Diffusion Simulation Results

Each run was been carried out over a period of 48 hours so that the diurnal variations could be observed. We present here only a selection of the results that illustrate the main features relating to the adaptivity and to the use of SCIRun. The main area of mesh refinement is along the plume edges close to the chimney, indicating that there is a high level of structure in the plume. Using the adaptive mesh, we can clearly see


Advect8:
Flow Velocity Vx: 7.5e-06
Flow Velocity Vy: 3.e-06
Flow Velocity Vz: 0
Maximum refinment level: 2
Maximum iterations: 40
Iteration #: 40
Mesh size : 4423
Status: Done!

X Plane 0.000
Y Plane 0.239
Z Plane 0.027

X Plane 0.000
Y Plane 0.216
Z Plane 0.027

uiBCuttingPlane_2
X Plane 0.000
Y Plane 0.237
Z Plane 0.027

Figure 4: A dataflow network, showing modules (boxes), connections (wires) and the i/o ports on the modules that the wires connect). The plume and mesh shown are in the early stages of development.

the plume edges and can easily identify areas of high concentrations. The effects of the plume on ozone concentrations also provides some interesting results. Close to the plume the concentration of O_3 is much lower than that in the background. Due to the high NOx concentrations the inorganic chemistry is dominant in this region and the ozone is consumed by the second reaction. As the plume travels downwind and the NOx levels decrease, the plume gradually picks up emissions of VOC's, as shown in Figure 5. The OVC chemistry leads to the production of NO_2 which pushes the above reaction in the reverse route. The levels of ozone can therefore rise above the background levels at quite large distances downwind from the source of NOx.

Computational Inverse EEG Problem

The inverse EEG problem can be described as the mathematical mapping of EEG scalp recordings back onto the cortical surface or within the cortex to approximate fundamental current sources. This inverse problem lies at the foundation of surgical planning and prognosis for neurological conditions ranging from epilepsy to schizophrenia [18] and to brain tumors. The goal of cortical mapping is to integrate patient anatomical information and measured voltage potential recordings from the surface of the patient's scalp in order to non-invasively determine the electrical activity on and within the patient's cortical surface [19].

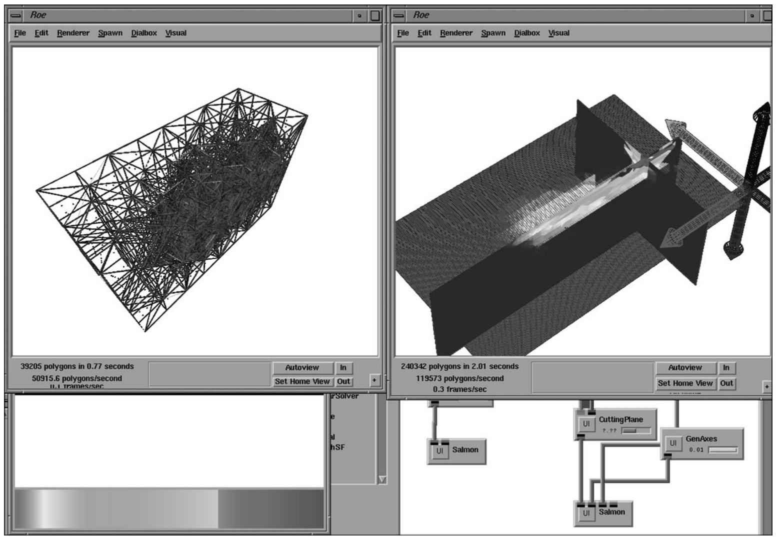


Figure 5: This picture shows one component of the plume in greater detail in three perpendicular cross-sections.

There has been much research into computationally modeling the electrical activity of the brain, but only a few successful systems have been implemented. These systems range in geometric model complexity from grossly simplified, spherical representations to patient-specific finite element models. However, none of the systems to date have been designed for large-scale architectures and none of these systems offer a flexible mechanism for comparing modeling and solution techniques.

In this application we present an inverse EEG solution and implementation within the SCIRun problem solving environment. Leveraging the infrastructure of SCIRun, we have designed a general tool that enables users to experiment with various modeling and simulation techniques and to examine the results with many types of visualization probes and methods. The user can control all aspects of the problem, scaling the model and simulation complexity to match available computational resources and experimenting with alternative solutions to gain intuition about the problem.

A schematic overview of our solution process for the inverse EEG problem is shown in Figure 6. A segmented MR volume provides anatomical data, required for accurate conductivity and boundary condition information in our model. Functional data, the known EEG potentials at the scalp boundary, are read in from a raw file and stored with the digitized point locations sampled with a Polhemus tracker. This data forms the basis for a finite element inverse problem, whereby either the electric sources within the brain that induced the recorded EEG potentials, or the corresponding potentials on the cortical surface, can be computationally recovered.

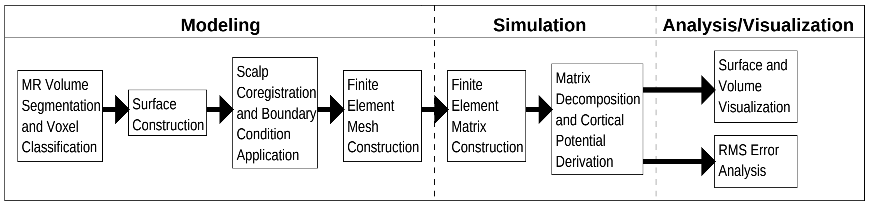

Modeling
Simulation
Analysis/Visualization
MR Volume Segmentation and Voxel Classification
Surface Construction
Scalp Coregistration and Boundary Condition Application
Finite Element Mesh Construction
Finite Element Matrix Construction
Matrix Decomposition and Cortical Potential Derivation
Surface and Volume Visualization
RMS Error Analysis

Figure 6: A schematic representation of the full inverse EEG pipeline.

As a plug-in system, SCIRun allows rapid prototyping and analysis of new implementations. The user can evaluate different methods by implementing them in different modules and "hot-swapping" them within a common dataflow network to compare results.

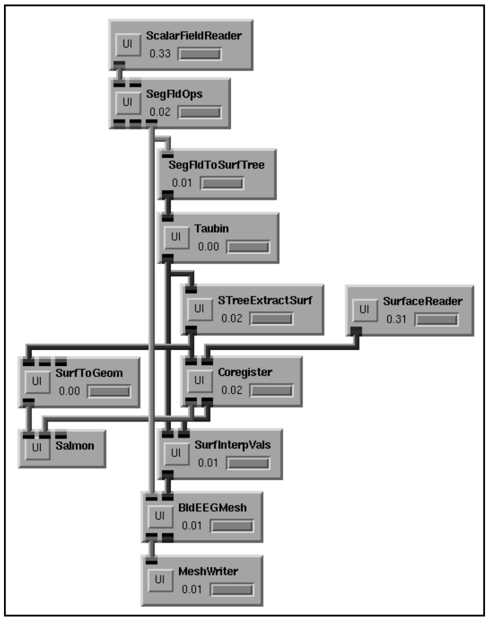


Figure 7: The SCIRun inverse EEG modeling pipeline.

The modeling components of the inverse EEG pipeline collectively enable the user to construct a full finite element mesh with appropriate boundary conditions and conductivity tensors from segmented MRI images, raw EEG potentials and digitized positional information [20]. The modeling pipeline is shown in Figure 7.

EEG Simulation Results

We have applied the inverse EEG pipeline described above to examine the neural activation process known as the P300. The P300 component of the event-related potential (ERP) follows 300-400 ms after the presentation of a stimulus [21].

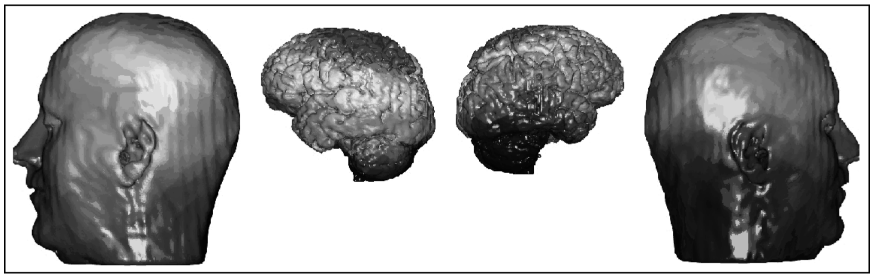


Figure 8: Results from a cortical mapping simulation. For visualization purposes, the nodal values have been interpolated over the original scalp and cortical surfaces.

Total execution time for the pipeline that generated the solution in Figure 8 was less than fifteen minutes. Most importantly, only one minute of time required manual intervention (as the user manually chose the fiducials for coregistration); the rest of the time the algorithms functioned completely automatically.

Reservoir Simulation and Visualization

Finally, we conclude with some examples of using SCIRun for large-scale geoscience simulation and visualization. We recently worked with members of the Energy Geosciences Institute (EGI) at the University of Utah and a major U.S. oil company to incorporate oil reservoir simulation and visualization within SCIRun. We incorporated a fully-implicit, three dimensional, two-phase (oil/gas) simulator into the SCIRun environment. The simulator was developed in C^{++}and highly modularized. The user can control the simulation before it starts, during the run and to initiate a new simulation based on the final or partial results. Real-time feedback is provided to the user during simulation via a graph of the convergence of the solution and via the presentation of intermediate solutions. The user can change the target error for the simulation in real-time, examine the visualization of the intermediate results or manipulate the input to the simulation.

Additionaly, the user can change well parameters, such as location of the wells in 3D by selecting a well and moving it to a different location. The user can also change the strength of the flow of a well and determine whether the well is a producer or injector. These changes can be applied while the simulation is in progress which in turn will cause to simulation to stop and restart automatically based on the new parameters. Below we show two sample visualizations of large-scale seismic data visualization.

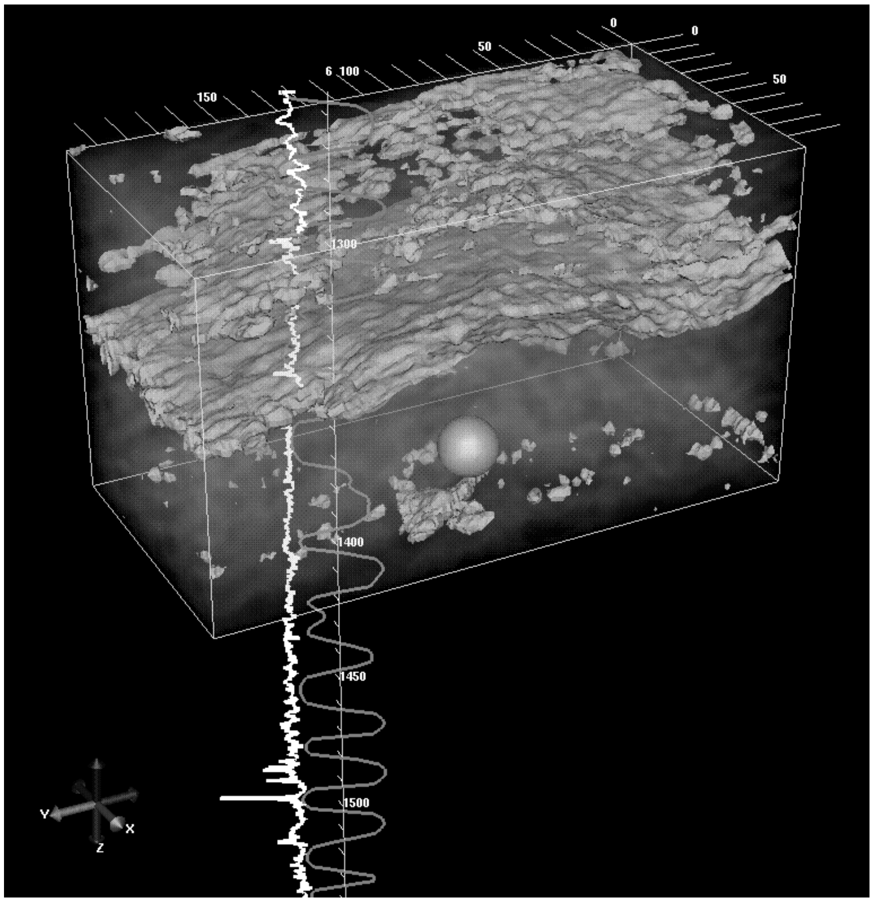


Figure 9: Seismic data for targeting strategic drilling sites using isosurface extraction.

Future Work

In a flexible, extensible environment such as SCIRun, there are numerous possibilities for expansion. Many new features driven by the needs of scientific applications such as those we have described above. Other needs are the result of basic research in component architectures for scientific computing.

One of the largest infrastructure changes will be support for execution in a networked/distributed environment [4]. For simplicity, we focused on shared memory multiprocessors for the initial implementation of SCIRun. The second implementation included support for distribting modules to achieve task based parallelism over a network. We also are now pushing it to work effectively on large-scale distributed memory

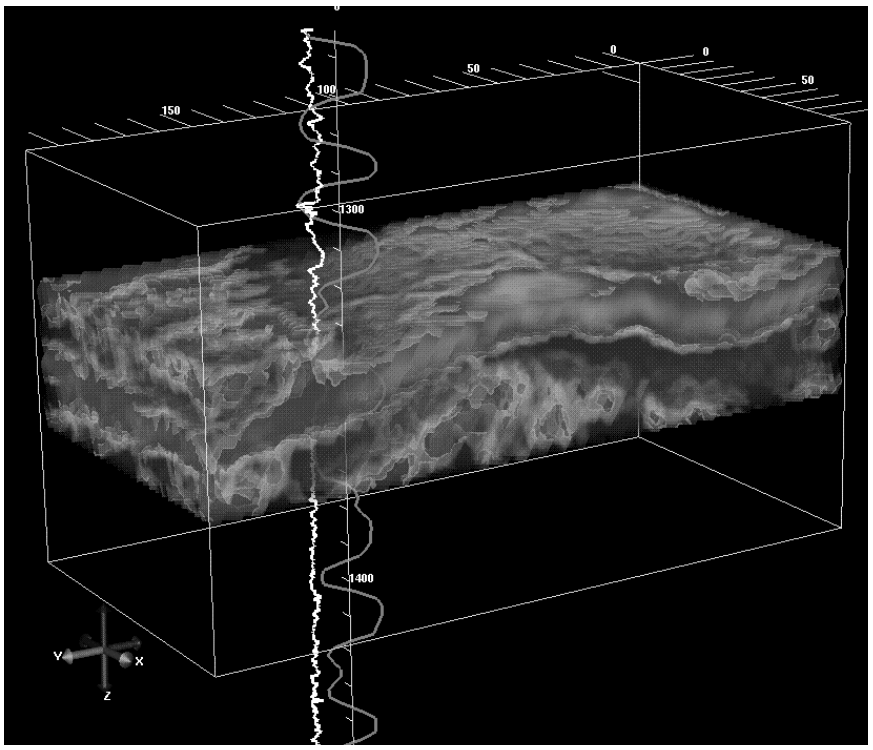


Figure 10: Volume rendering of seismic data.

supercomputers and clusters. In this case, a single module may use dozens or hundreds of processors to perform the desired task.

Along with these modifications will come "detachable user interfaces." Currently SCIRun applications must be executed within the SCIRun user interface, but for a long-running simulation, it would be beneficial to start the program and then come back periodically to check on the progress of the program. The user can steer the simulation and then return at a later time to see the effects of the changes. This modification would likely be performed in conjunction with a modified user interface that reduces the number of popup windows scattered about the user's screen.

The Common Component Architecture Forum [22] is a current and ongoing collaboration between the Center for Scientific Computing and Imaging (SCI) at the University of Utah and the Department of Energy National Laboratories (and other university research groups). With representatives from these facilities, the CCA Working Group was formed "to develop a specification for a component architecture for high-performance computing." This goal has long been a central theme of the SCIRun problem solving environment. The proposed project will form a symbiotic relationship between the DOE labs, Utah, and other university participants, in which SCIRun will provide applications, tools, and experience with high-performance component architectures to the CCA community, and the DOE labs and other university groups part of the CCA effort will help provide SCIRun with a more flexible and widely accepted component model.

Acknowledgments

This work was supported in part by awards from the NIH NCRR, DOE ASCI and the NSF. We would like to thank Yarden Livnat for the seismic images and Ray Levey of EGI for the seismic data. The authors would like to thank our collaborators: Martin Berzins, Geoff Potts, Don Tucker, Simon Arridge, and Martin Schweiger who have developed applications within SCIRun.

References

[1] C.R. Johnson and S.G. Parker. Applications in computational medicine using SCIRun: A computational steering programming environment. In H.W. Meuer, editor, *Supercomputer ‘95*, pages 2–19. Saur-Verlag, 1995.

[2] S.G. Parker and C.R. Johnson. SCIRun: A scientific programming environment for computational steering. In *Supercomputing ‘95*. IEEE Press, 1995.

[3] S.G. Parker, D.M. Weinstein, and C.R. Johnson. The SCIRun computational steering software system. In E. Arge, A.M. Bruaset, and H.P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 1–44. Birkhauser Press, 1997.

[4] M. Miller, C. Hansen, and C.R. Johnson. Simulation steering with SCIRun in a distributed environment. In B. Kgstrm, J. Dongarra, E. Elmroth, and J. Wasniewski, editors, *Applied Parallel Computing, 4th International Workshop, PARA’98*, volume 1541 of *Lecture Notes in Computer Science*, pages 366–376. Springer-Verlag, Berlin, 1998.

[5] S. G. Parker. *The SCIRun Problem Solving Environment and Computational Steering Software System*. PhD thesis, University of Utah, 1999.

[6] C.R. Johnson, M. Berzins, L. Zhukov, and R. Coffey. SCIRun: Applications to atmospheric diffusion using unstructured meshes. In M.J. Baines, editor, *Numerical Methods for Fluid Dynamics VI*, pages 111–122. Oxford University Press, 1998.

[7] M. Schweiger, L. Zhukov, S. Arridge, and Johnson C. Optical tomography using the SCIRun simulation and visualization package: Preliminary results for three-dimensional geometries and parallel processing. *Optical Express: International Electronic Journal of Optics*, 4(8):263–269, 1999.

[8] L. Durbeck, N. Macias, D. Weinstein, C. Johnson, and J. Hollerbach. SCIRun haptic display for scientific visualization. In *Third Phantom User's Group Workshop, PUG98*, Cambridge, MA, 1998. MIT.

[9] Center for the Simulation of Accidental Fires and Explosions (C-SAFE): http://www.csafe.utah.edu/.

[10] NIH NCRR Center for Bioelectric Field Modeling, Simulation, and Visualization: http://www.sci.utah.edu/ncrr.

[11] B. Ribarsky and et al. Object-oriented, dataflow visualization systems—A paradigm shift? In *Proceedings of Visualization '92*, pages 384–388. IEEE Press, 1992.

[12] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, 1991.

[13] C.R. Johnson and S.G. Parker. A computational steering model applied to problems in medicine. In *Supercomputing '94*, pages 540–549. IEEE Press, 1994.

[14] G. Hart, A. Tomlin, J. Smith, and M. Berzins. Multi-scale atmospheric dispersion modelling by the use of adaptive grid techniques. In *Environmental Monitoring and Assessment*, 1997.

[15] A. Tomlin, M. Berzins, J. Ware, J. Smith, and M. Pilling. On the use of adaptive gridding methods for modelling chemical transport from multi-scale sources. *Atmospheric Env.*, 31(18):2945–2959.

[16] I. Ahmad and M. Berzins. An algorithm for odes from atmospheric dispersion problems. *Appl. Num. Math.*, 25:137–149, 1997.

[17] W. Speares and M. Berzins. A 3d unstructured mesh adaptation algorithm for time dependent shock dominated problems. *Int. Jour Num. Meths. in Fluids*, 25:81–104, 1997.

[18] S.F. Faux, R.W. McCarley, P.G. Nestor, M.E. Shenton, S.D. Pollak, V. Penhume, E. Mondrow, B. Marcy, A. Peterson, T. Horvath, and K.L. Davis. P300 topographic assymetries are present in unmedicated schizophrenics. *Electroencephalography and clinical Neurophysiology*, 88:32–41, 1993.

[19] B. Lutkenhoner, E. Menninghaus, O. Steinstrater, C. Wienbruch, M. Gissler, and T. Elbert. Neuromagnetic source analysis using magnetic resonance images for

the construction of source and volume conductor model. *Brain Topography*, 7(4):291–299, 1995.

[20] D.M. Weinstein and C.R. Johnson. Effects of geometric uncertainty on the inverse EEG problem. In R.L. Barbour, M.J. Carvlin, and M.A. Fiddy, editors, *Computational, Experimental, and Numerical Methods for Solving Ill-Posed Inverse Imaging Problems: Medical and Nonmedical Applications*, volume 3171, pages 138–145. SPIE, 1997.

[21] S. Sutton, M. Braren, J. Zubin, and E. R. John. Evoked potential correlates of stimulus uncertainty. *Science*, 150:1187–8, 1965.

[22] Common Component Architecture Forum: http://www.acl.lanl.gov/cca.