The SCIRun Parallel Scientific Computing Problem Solving Environment

Christopher R. Johnson
Department of Computer Science
University of Utah
Salt Lake City, UT 84112
crj@cs.utah.edu

Steven G. Parker
Department of Computer Science
University of Utah
Salt Lake City, UT 84112
sparker@cs.utah.edu

Extended Abstract

Introduction SCIRun is a parallel scientific programming environment that allows the interactive construction, debugging and steering of large-scale scientific computations. SCIRun can be envisioned as a "computational workbench," in which a scientist can design and modify simulations interactively via a dataflow 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.

Steering The primary purpose of SCIRun is to enable 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 accomlishing 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. libMultitask is 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).

Managing Resources in SCIRun In a sophisticated simulation, each of the individual components (modeling, mesh generation, nonlinear/linear solvers, visualization, etc.) typically consumes a large amount of memory and CPU resources. When all of these pieces are connected into a single program, the potential computational load is enormous. In order to use the resources effectively, SCIRun adopts a role similar to an operating system in managing these resources. SCIRun manages prioritization of threads, mapping of threads to processors, inter-thread communication, thread stacks, memory allocation policies, and memory exception signals (such as segmentation violations). In the future, an even more comprehensive management of system resources will be implemented.

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.