Indiana University Final Report for the Center for Simulation of Wave Interactions with Magnetohydrodynamics (SWIM) project, DOE contract ER2580-4

Randall Bramley
Department of Computer Science
School of Informatics
Indiana University
Bloomington, IN

August 2, 2012

1 Project Description

Indiana University's SWIM activities have primarily been in three areas. All are completed, but we are continuing to work on two of them because refinements are useful to both DoE laboratories and the high performance computing community.

2 Linear Solver Interface (LISI)

The Linear Solver Interface (LISI) is a Common Component Architecture (CCA) compliant interface to high performance solvers for sparse systems of linear equations. Those are requried by several SWIM codes, and the MHD codes in particular rely heavily on the availability of such solvers. A single solver method usually cannot succeed on every linear system generated even within an MHD code, so the ability to rapidly test and switch to a different package when failures occur is vital. This is easy to do and has been done for serial codes, but for parallel linear solver packages a large amount of data decomposition, initial set up of distributed data structures, and ways of establishing communications between processes for a linear solve make this kind of plug and play capability difficult. Most codes are written to use one such system, and switching to a different solver is difficult, error-prone, and time-consuming.

An former IU Ph.D. graduate, Fang (Cherry) Liu, has developed LISI in collaboration with the Ames Iowa lab to address this problem, and now it encapsulates many of the most frequently used parallel linear solver systems including Hypre (LLNL), PeTSC (ANL), Trilinos (Sandia), SuperLU (Lawrence Berkeley Lab). LISI itself is not a linear system solver, but instead a simple CCA interface to the solvers, which hides the differences between the solver implementations. This

makes experimentation and simulation easier and more robust. Changing a code like NIMROD to use Hypre instead of PeTSC requires changing roughly 200 lines in the source code; with LISI the user does not need to change any lines of code and can specify which solver to use as an input parameter.

End users primarly only care about the solvers working accurately, reliably, and scalably parallel. Getting the solvers to work correctly is the task of numerical linear algebraists and scientific computing experts. They in turn are greatly interested in the linear solve process itself, to find limitations and design better algorithms. Hierarchical parallel solvers (e.g., Hypre and Trilinos) in particular are still in development, since a failure to solve on a single level or single component causes the entire solve to fail. To address both end users and linear solver experts, LISI has delineated three user interface levels explicitly defined. It is still a single CCA interface, but that interface can be subclassed to allow finer control over the solve process. Three levels of user interface were defined together with Ames Lab and CCA participants and are

1. A level that just provides a minimalist invocation to a solver, which simply consists of a call to a solver provider with only the linear system defined. In this case the solver provider determines all of the method and parameter settings required.
2. An intermediate level where an end user can choose particular solver methods (e.g., GMRES with diagonal preconditioning) and other high-level control parameters.
3. An expert level which allows full control of all parameters and methods.

An end user can use any of those or use primarily a minimalist level with only a few parameters set. The expert level in part is intended for researchers in numerical linear algebra who are working with particular applications, where some knowledge of the underlying problem can help guide those choices.

2.1 ISP Framework

The SWIM framework is the Integrated Plasma Simulator (ISP), a set of Python codes and interfaces that a plasma physicist uses to specify a simulation that involves multiple codes acting indepentdently. Python was chosen because it is available on all DOE high-performance platforms, including some for which Java is not available. Python has also been widely adopted by end-user scientists independently of the ISP, making the learning curve as flat as possible. The framework provides an interface that allows physicists to formulate a combined simulation in a natural style, using loops and other control structures which are familiar to Fortran programmers. Underneath the ISP is an object-oriented Common Component Architecture (CCA) software component system, which end users can manipulate and modify, but which no end user has to deal with. The ISP is currently in use for production SWIM runs, and seems stable.

The ISP-related activity at IU was the development of the "sub-batch resource manager" (SBR). When multiple interacting codes are used, they all need to be launched from a single MPI batch job. If multiple batch jobs are launched, one for each code, a single code may wait for days in the queue before being started. Because SWIM requires coupling codes in a tight time loop, this makes using the batch system impractical.

The Fusion Simulation Project (FSP) codes also vary widely from serial to limited parallelism to scalably parallel programs. To avoid having several thousand processors idle when a limited-parallel code is running, the SBR has the opportunity to identify other tasks from the same run which are waiting execution. Although all HPC batch managers allow launching a script instead of a single parallel code, those scripts are static and cannot take advantage of execution opportunities that occur during a run.

This scheduling problem is familiar for multiprogram operating systems and Grid job management tools like Kepler, but it occurs at a lower level at which those tools do not operate.

The IU Ph.D. graduate Samantha Foley has had primary responsibility for research on this work, and it is now part of the SWIM toolset. The ISP has been written in a way that has allowed the SBR to be introduced without requiring intervention from the end user. Further unfunded work includes experimenting with different scheduling strategies such as guided self-scheduling.

3 Connecting Components with an MPI I/O Interface

Many existing methods for connecting independently running programs involve using file I/O. One or more codes writes out a results file which is subsequently used as input for dependent waiting codes. File I/O involves a major bottleneck – not from the actual time to write out or read in a file, but from the implicit synchronization point required to have a single process (the process with rank 0 in MPI) carry out the task. The MPI I/O interface was created to deal with that problem, allowing users to read and write a single file in parallel. Furthermore, many end users are familiar with the parallel I/O interface and all HPC users are accustomed to using MPI.

The IU Ph.D. candidate D. Kevin McGrath researched the mechanisms that will allow codes that are connected by I/O to files, to instead be seamlessly shifted to using in-system network communications that do not require actually writing to a file. End users do not have to modify their source code, but instead the underlying I/O library handles the between- programs data flow. This task is eased by Argonne National Lab's ROMIO implementation of MPI I/O. That library allows easily including a new "abstract device" in its own directory, which then handles th actual data transfer. Although intended for multi-network HPC clusters, the design of ROMIO has greatly eased implementing this idea.

In related work, ANL has used the same mechanism for intercepting I/O to use a WAN for connecting codes running across the Grid, but here the goal is for fully-parallel connections with components simultaneously running on the same HPC platform. This is similar to the MxN problem that Felipe Bertrand implemented in the CCA. The MxN problem is that of connecting a set of software components running on M processors to a second set running on N processors. However, the design of CCA required initially creating M*N connections, then dropping those not needed. Bertrand's work showed that the idea was viable, but clearly it is not scalable – consider the case when M = 20,000 and N = 64,000.

The research and devlopment at IU has eases the CCA restrictions, allowing the number of connections established to be $O(M + N)$ instead of $O(M * N)$. ROMIO is the most common implementation of parallel I/O, shared by several MPI implementations, including OpenMPI, LAM MPI, and MPICH. This allows simply dropping in the new abstract device for any of those implementations.

4 Collaborations and Activities

4.1 People

Personal supported in whole or part by SWIM funding include:

- Randall Bramley (PI)
- Fang (Cherry) Liu (former Ph.D. student now at Iowa Ames)
- Samantha Foley (former Ph.D. student now at ORNL)
- D. Kevin McGrath (Ph.D. student)

One undergraduate and three other graduate students have also worked on projects related to SWIM but were funded from other sources.

5 Papers

Papers and presentations about IU's activities in SWIM include:

- Wael R. Elwasif, Aniruddha G. Shet, David E. Bernholdt, Samantha S. Foley, Randall Bramley, Donald B. Batchelor, and Lee A. Berry. *The Design and Implementation of the SWIM Integrated Plasma Simulator*. In 18th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP 2010), 2010.
- Samantha S. Foley, Wael R. Elwasif, David E. Bernholdt, Aniruddha G. Shet, and Randall Bramley. *Extending the Concept of Component Interfaces: Experience with the Integrated Plasma Simulator*. In Component-Based High-Performance Computing (CBHPC), 2009.
- Samantha S. Foley, Wael R. Elwasif, Aniruddha G. Shet, David E. Bernholdt, and Randall Bramley. *Incorporating Concurrent Component Execution in Loosely Coupled Integrated Fusion Plasma Simulation*. In Component-Based High-Performance Computing (CBHPC), 2008.
- Samantha S. Foley, Wael R. Elwasif, David E. Bernholdt, Aniruddha G. Shet, and Randall Bramley. *Extending the Concept of Component Interfaces: Experience with the Integrated Plasma Simulator*. Component-Based High-Performance Computing (CBHPC), Portland, OR, November 2009.
- Samantha S. Foley. *Experiences Building a Resource Manager for the IPS*. Common Component Architecture (CCA) Forum Meeting, Bloomington, IN, October 2009.