ANL/MCS-TM-356

SHARP pre-release v1.0 – Current status and documentation

Mathematics and Computer Science Division

About Argonne National Laboratory
Argonne is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357. The Laboratory's main facility is outside Chicago, at 9700 South Cass Avenue, Argonne, Illinois 60439. For information about Argonne and its pioneering science and technology programs, see www.anl.gov.

DOCUMENT AVAILABILITY

Online Access: U.S. Department of Energy (DOE) reports produced after 1991 and a growing number of pre-1991 documents are available free via DOE's SciTech Connect (http://www.osti.gov/scitech/)

Reports not in digital format may be purchased by the public from the National Technical Information Service (NTIS):
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312
www.ntis.gov
Phone: (800) 553-NTIS (6847) or (703) 605-6000
Fax: (703) 605-6900
Email: **orders@ntis.gov**

Reports not in digital format are available to DOE and DOE contractors from the Office of Scientific and Technical Information (OSTI):
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831-0062
www.osti.gov
Phone: (865) 576-8401
Fax: (865) 576-5728
Email: **reports@osti.gov**

Disclaimer
This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor UChicago Argonne, LLC, nor any of their employees or officers, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of document authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof, Argonne National Laboratory, or UChicago Argonne, LLC.

ANL/MCS-TM-356

SHARP pre-release v1.0 – Current status and documentation

Vijay S. Mahadevan, Ronald Rahaman
Mathematics and Computer Science Division,
Argonne National Laboratory

September 30, 2015

EXECUTIVE SUMMARY

The NEAMS Reactor Product Line effort aims to develop an integrated multiphysics simulation capability for the design and analysis of future generations of nuclear power plants. The Reactor Product Line code suite's multi-resolution hierarchy is being designed to ultimately span the full range of length and time scales present in relevant reactor design and safety analyses, as well as scale from desktop to petaflop computing platforms. In this report, building on a several previous report issued in September 2014, we describe our continued efforts to integrate thermal/hydraulics, neutronics, and structural mechanics modeling codes to perform coupled analysis of a representative fast sodium-cooled reactor core in preparation for a unified release of the toolkit. The work reported in the current document covers the software engineering aspects of managing the entire stack of components in the SHARP toolkit and the continuous integration efforts ongoing to prepare a release candidate for interested reactor analysis users.

Over the past several years, the Reactor Product Line effort has developed high-fidelity single-physics codes for neutron transport modeling, in the PROTEUS code, and Computational Fluids Dynamics thermal/fluid modeling in the Nek5000 code. Both these codes have been exercised on over 100,000 processors of the IBM Blue Gene/P. The Diablo code has been used to perform structural mechanics and thermo-mechanical modeling. MOAB, the Reactor Geometry Generator (RGG), and MeshKit have been developed to generate and manipulate mesh and mesh-based data, in both serial and parallel environments. With the coupled physics simulations orchestrated by CouPE, the combination of these modules enables computational scientists to simulate complex multi-physics problems in nuclear engineering. These tools together form a strong basis on which to build a multi-physics modeling capability. The goal of developing such a tool is to perform multi-physics neutronics, thermal/fluid, and structural mechanics modeling of the components inside a reactor core, the full reactor core or portions of it, and be able to achieve that with various level of fidelity. This flexibility allows users to select the appropriate level of fidelity for their computational resources and design constraints.

Here we report on the continued integration effort of PROTEUS/Nek5000 and Diablo into the NEAMS framework and the software processes that enable users to utilize the capabilities without losing scientific productivity. Due to the complexity of the individual modules and their necessary/optional dependency library chain, we focus on the configuration and build aspects for the SHARP toolkit, which includes capability to auto-download dependencies and configure/install with optimal flags in an architecture-aware fashion. Such complexity is untenable without strong software engineering processes such as source management, source control, change reviews, unit tests, integration tests and continuous test suites. Details on these processes are provided in the report as a building step for a SHARP user guide that will accompany the first release, expected by Mar 2016.

TABLE OF CONTENTS

EXECUTIVE SUMMARY i
Table of Contents ii
List of Figures iv
List of Tables iv
1 Introduction 5
2 SIGMA toolkit 6
3 The SHARP toolkit 8
4 Configuration 10
4.1 Package Dependencies 10
4.2 Source management 11
4.3 Getting started 12
4.4 Configuration Options 12
4.4.1 Enabling/disabling compile-time features or packages 13
4.4.2 Linking to existing libraries 13
4.4.3 Downloading third-party libraries 13
4.4.4 Precedence of Options 14
4.4.5 Available options 14
4.5 Configuration on the Blues Cluster 16
4.5.1 Basic Example 16
4.5.2 MPI libraries and compilers 16
4.5.3 Configuring with Diablo 17
4.5.4 Building essential third-party libraries from source 17
4.5.5 Linking to pre-installed libraries 17
4.6 Compiler Choice 18
4.7 Updated Documentation 18
4.8 Standardizing command-line options 19
4.9 Early checking of command-line options 19
5 Open and Recently-Resolved Issues 19
Parsing Options 19
Help messages 19
Detecting and Propagating Variables for Compiler Choices, Flags, Libraries, etc. 20
PROTEUS Compilation 20
Multiphysics Coupling 20
6 Regression testing 20
7 Test problems 22
8 Future plans 23
Acknowledgments 23

REFERENCES 23

LIST OF FIGURES

Figure 1. SIGMA toolkit 7
Figure 2. SHARP Architecture Scheme 9
Figure 3. SHARP Workflow 10
Figure 4. SHARP Architecture Scheme 22

LIST OF TABLES

Table 1. *List of SHARP toolkit dependencies* 10
Table 2. *Available list of configuration options* 14
Table 3. *List of continuous regression builds and tests* 21

1 Introduction

The NEAMS Reactor Product Line (RPL) aims to develop an integrated multi-physics simulation with a multi-resolution hierarchy that is designed to ultimately span the full range of length and time scales present in relevant reactor design and safety analyses, as well as scale from desktop to petaflop computing platforms. In this report, we focus on the steps taken in terms of software engineering to verify individual physics component codes, and the integrated coupled system through robust architecture-aware configuration and build system development, regression test suites, and nightly verifications. These steps are essential in preparation of a reliable first release of the Simulation-based High-efficiency Advanced Reactor Prototyping (SHARP) toolkit to interested users. The current report is intended to provide details on the current software architecture of SHARP, its components and the ongoing steps in preparation for the first release version by Mar 2016.

The SHARP modeling and simulation project is a component-based toolkit that enables nuclear engineers and computational scientists to analyze and design the next generation of predictive tools for nuclear reactor analysis. The SHARP project is unique in its capabilities by employing advanced parallel methods development in support of reactor core calculations, using state-of-art discretization and solver strategies for both the single physics and coupled multi-physics applications. The SHARP toolkit integrates neutronics, thermal/hydraulics, and structural mechanics physics modules to perform coupled reactor analysis on representative sodium-cooled fast reactor core geometries.

In order to produce a fully coupled-physics simulation capability, there are two obvious approaches that can be pursued. In one approach, existing single-physics codes/components can be assembled into an overall coupled simulation code with appropriate interfaces to communicate between the components to capture the nonlinear feedback effects. This is generally referred to as a “small-f” or “bottom-up” framework approach [1, 2]. The other approach is to use an integrated, coupled-physics modeling framework, with new code pieces for each relevant physics area developed inside that framework from scratch. This is sometimes referred to as a “large-F” or “top-down” framework approach [3, 4]. The primary advantage of the former approach is that it preserves several man-years invested in existing verified and validated individual physics modeling codes, but at the cost of some intrusive modifications to enable the software interfaces. The large-F approach avoids intrusive interfacing by providing a unified platform to enable coupling, but at the cost of re-writing all the necessary physics codes and verifying the components individually and as a whole. The overall approach being pursued in the RPL effort is to develop and demonstrate a small-f framework for performing coupled multiphysics analysis of reactor core systems. This system takes advantage of many single-physics codes also sponsored by the overall NEAMS program over past several years.

The details regarding the background on construction of the IPL coupled physics framework (SHARP) along with the software engineering details are discussed in the following sections.

2 SIGMA toolkit

The Scalable Interfaces for Geometry and Mesh based Applications (SIGMA) toolkit [5] provides interfaces and tools to access geometry data, create high quality unstructured meshes along with unified data-structures to load and manipulate parallel computational meshes for various applications to enable efficient physics solver implementations. Mesh generation is a complex problem since most problem geometries involve complicated curved surfaces that require physics imposed spatial resolution and optimized elements for good quality. These tools simplify the process of generation and handling of discrete meshes with scalable algorithms to leverage efficient usage from desktop to petascale architectures.

The SIGMA toolkit has been designed with the aim to enable and expedite modeling scientific problems by computational scientists and converge on simulation workflow without investing significant effort in learning an array of tools. It provides interfaces and tools to understand geometry models, create high quality unstructured meshes along with unified data-structures to load and manipulate parallel computational meshes for various applications to enable efficient physics solver implementations. Additionally, optimal quality mesh generation is a complex process for complicated curved problem geometries, with physics imposed spatial resolution requirements. These tools simplify the process of generation and handling of discrete meshes with scalable algorithms to leverage efficient usage from desktop to petascale architectures. The interaction of the various SIGMA components along with several key dependent applications and tools are shown in Fig. 1.

CGM: The Common Geometry Module (CGM) [6], is a library with a common API for querying and modifying the topological model for geometry represented in `ACIS` and `OpenCASCADE` solid modeling engines. CGM (`ITAPS iGeom`) also provides the common geometry infrastructure for the CUBIT mesh generation toolkit [7].

MeshKit: One of the unique capabilities provided by SIGMA toolkit is the inclusion of an array of generic mesh generation algorithms that seamlessly integrates into several different external applications. MeshKit serves as a library of advanced mesh-generation algorithms implemented natively and provides an extensible design to perform R&D on new meshing techniques.

The MeshKit library [8] contains several mesh-generation algorithms that utilize CGM and MOAB descriptions of geometry and mesh to discretize complex geometric domains. Nuclear reactor specific geometries can be easily meshed through a GUI tool from Kitware built on top of Reactor Geometry Generator written in MeshKit whose overall goal is to develop a complete package for generating reactor core models.

MOAB: The Mesh-Oriented datABase (MOAB) [9] (`ITAPS: iMesh, iMeshP`), is a library for uniformly representing both structured and unstructured meshes, along with associated field data defined on mesh entities, using an efficient array-based data model. MOAB provides query, construction, and modification of finite-element meshes, plus polygons and polyhedra. Various options are available for writing and visualizing the final

meshes produced by meshing algorithms. MOAB uses an HDF5-based file format, which can be visualized by using a ParaView plugin that is implemented by the MOAB library. The Visit visualization tool can also be configured and built with MOAB to provide a similar import capability.

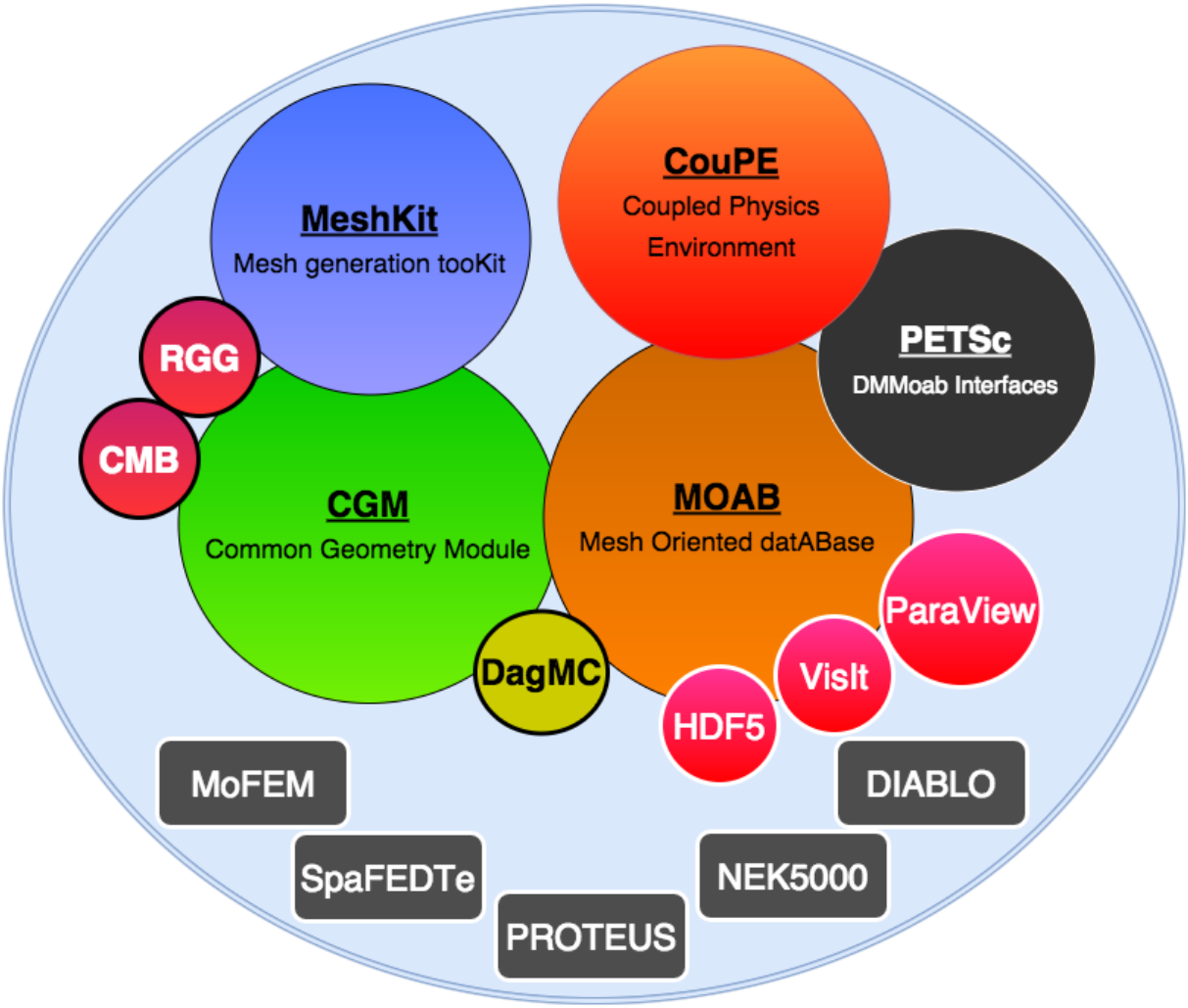

Figure 1. SIGMA toolkit

MOAB has shown demonstrated scalability to at least 512K processors [10] using its efficient parallel mesh handling algorithms, which includes arbitrary point location, and interpolation of field data during multi-mesh solution transfers.

The relational mapping between MOAB and CGM is also implemented in Lasso (`ITAPS iRel`), which is utilized for mesh generation and adaptive mesh refinements, where associations between mesh and geometry must be queried and maintained. Recently, in a move to reduce the dependency chain for SIGMA, a decision was taken to integrate the Lasso source base into MOAB. The description of this effort is provided in one of the subsections.

Solver Interoperability: High-resolution computational physics solvers require assembling the discrete operators on unstructured meshes and strong software coupling between scalable solvers in PETSc [11] and MOAB provide abstractions (**DMMoab**) to manage field data, DoFs and traversal semantics to fully enable efficient solution techniques. In coupled phenomena, aggregating the interaction between such solvers to capture the physical behavior during solution evolution of dependent models require consistent, global nonlinear solvers with advanced block-preconditioning strategies (in addition to robust multi-mesh transfer projection algorithms). These algorithms have been implemented within the Coupled Physics Environment (**CouPE**) [12] that utilizes the software abstractions exposed through DMMoab and native MOAB-based wrappers.

CouPE is implemented based on MOAB and makes use of PETSc solvers to drive the global nonlinear problem to convergence and provides a flexible infrastructure to abstract coupling needs without intrusively modifying existing physics solvers. Several real-world problems in nuclear reactor analysis [13] have been analyzed using this framework to accurately converge to coupled solutions of interest while quantifying numerical errors introduced due to both coupling and discretization using SIGMA tools.

The above SIGMA components provide the tools to seamlessly traverse the entire computational workflow from problem design to analysis, improving scientific productivity.

3 The SHARP toolkit

Typically, there are two primary approaches to assemble a coupled-physics simulation capability for analyzing reactor core systems. Of the two options, we choose to use the “bottom-up” approach for its ability to use existing physics codes and to take advantage of existing infrastructure capabilities in the MOAB and the coupling driver/solver library, CouPE which utilizes the widely used, scalable PETSc library.

Using an existing physics codes in this system (Fig. 2) requires that the system support whichever mesh type(s) the individual physics natively uses. The physics models can retain their own native representation of the mesh, which gets transferred to and from MOAB’s representation through a Mesh Adaptor (Physics Model A); or, it can use MOAB’s representation directly (Physics Model B). Language interoperability through the C/Fortran based iMesh interfaces also allows flexibility in the implementations that are tuned to individual physics requirements without overhead.

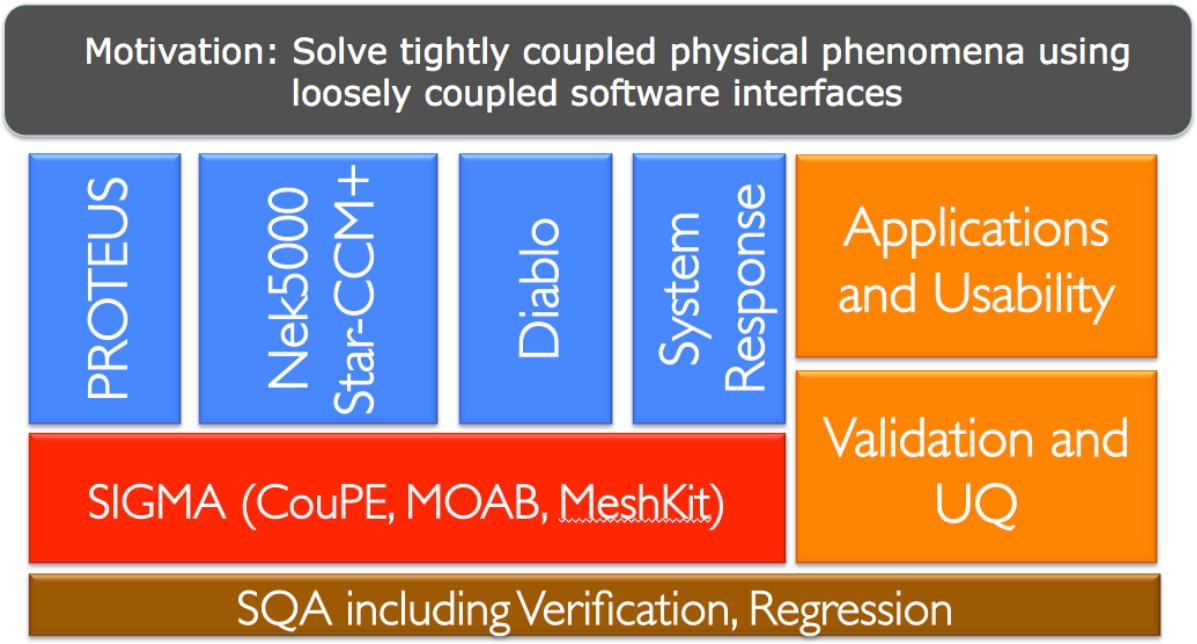

Figure 2. SHARP Architecture Scheme

In practice, this means that the coupled system may be solved on multiple meshes, each of which models part or the entire physical domain of the problem. In order to perform efficient coupled calculations, the results must be transferred from the physics/mesh on which they are generated (source), to the physics/mesh for which they provide initial or boundary conditions (target) due to nonlinearity introduced because of coupling between physics. "Multi-way" transfer is required in cases where the physics depend on each other's solution fields, for example in reactor analysis where neutronics computes heat generation based on temperature properties computed by thermal-hydraulics, which in turn depends on heat generation source term computed by neutronics.

Since relevant physics components solving a nuclear engineering problem have widely varying backgrounds in terms of code architectures, dependency requirements and specialized solver data-structures, a flexible approach to the coupling methodology was necessary to obtain accurate solutions. The SIGMA tools have been proven effective to handle unstructured mesh needs of applications in a scalable fashion and developments to handle the requirements for efficient, conservative spatial projection tools were pursued. The coupled physics driver based on CouPE library orchestrates the global nonlinear solver and effectively reduces the nonlinearities between the physics at every time-step in order to preserve the solution accuracy whilst maintaining unconditional stability requirements.

Fig. 3 illustrates the bottom-up approach used in the current NEAMS IPL effort. The MOAB library provides a representation of the meshes and optional tools configured as part of the library provide the solution transfer capability to project each physics component from the source to target meshes, with appropriate conservation prescriptions. The CouPE library is responsible for implementing multiphysics coupling methods to consistently and accurately couple the different components, in order to solve the nonlinear reactor-physics

problem. The combination of these tools provides the basis for the SHARP component-based framework in RPL.

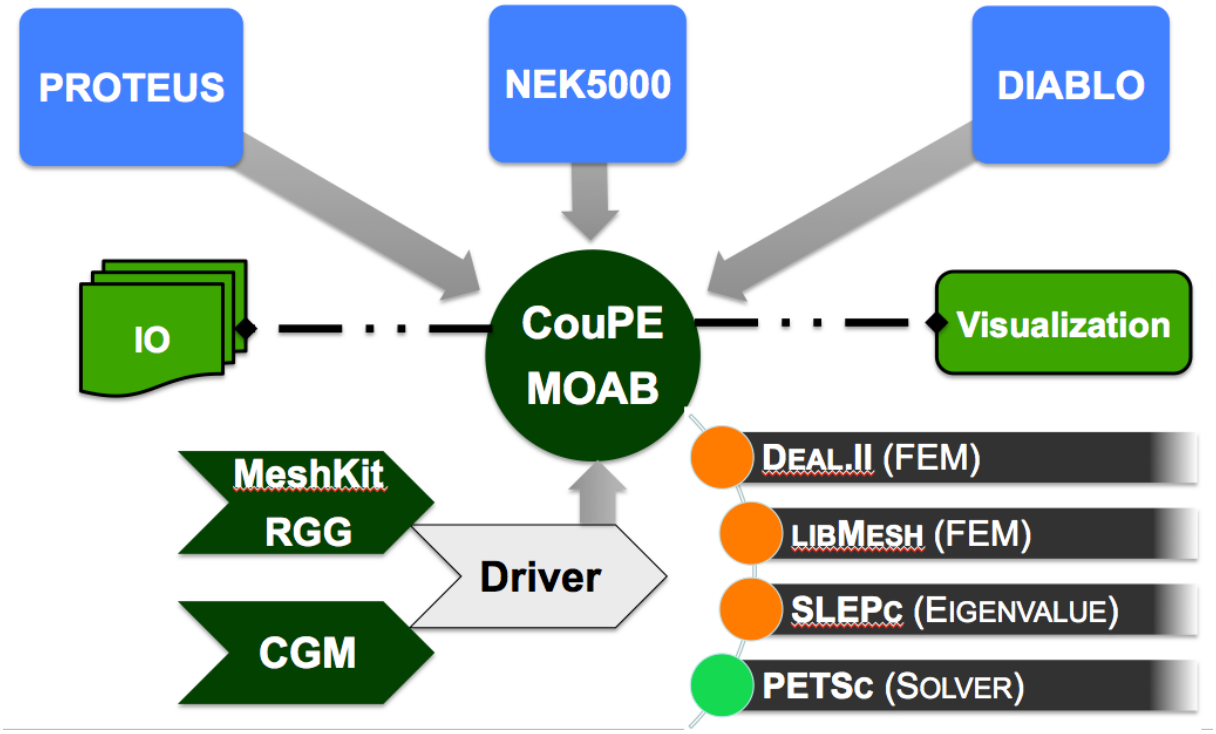

Figure 3. SHARP Workflow

4 Configuration

In this section, description of the complex configuration of the dependencies and the ease of use in managing the installation of SHARP toolkit is provided.

4.1 Package Dependencies

In the table below, "X" indicates a required package and "(X)" indicates an optional package. A number of packages are optional for single-physics modules but required by the coupled problems in SHARP.

Table 1. ***List of SHARP toolkit dependencies***

Package	Version	DIABLO	PROTEUS	Nek5000	SHARP	Notes
ARPACK		**X**				
EXODUS II	6.06	**(X)**				Only part of TRILINOS used by DIABLO
FEMSTER		**X**				
HDF5	1.8	**(X)**	**(X)**	**(X)**	**X**	Required for NetCDF and

						MOAB. Optional for single-physics
HYPRE	2.9	**X**				
ITAPS		(**X**)			**X**	Required for SHARP. Optional for single-physics
Metis	4.0	(**X**)	**X**	(**X**)	**X**	Required for MOAB and PROTEUS. Optional for single-physics with Nek5000 or DIABLO
MILI	13.1	**X**				
MOAB	4.7	(**X**)	(**X**)	(**X**)	**X**	Required for SHARP. Optional for single-physics
MPI		**X**	**X**	**X**		
MUMPS	4.1	**X**				
NASA		**X**				
NetCDF	4.3	(**X**)	(**X**)	(**X**)	**X**	Required for MOAB. Optional for single-physics
ParMetis	4	(**X**)			**X**	Required for SHARP. Optional for single-physics
PETSc	3.1		**X**			
PWSSMP		**X**				
SILO	4.1	**X**				
ZOLTAN	3.8	(**X**)	(**X**)	(**X**)	**X**	Required for MOAB. Optional for single-physics

4.2 Source management

The SHARP toolkit utilizes SVN version control system to manage the sources for different components. The repository is hosted at Argonne and is available at the following link, as long as users have been pre-approved to gain access to it.

`Link: https://svn.mcs.anl.gov/repos/SHARP`

Once source access has been provided, the user could checkout the sources by invoking the following Subversion command:

```
svn co https://svn.mcs.anl.gov/repos/SHARP/trunk SHARP
```

This command will create the SHARP directory containing all the configuration scripts, physics code module sources and a particular version of the coupled system driver along with several examples and test problems.

4.3 Getting started

To configure SHARP, the user first needs to run a top-level configuration generator script, aptly named "bootstrap". This script will verify whether the system contains the necessary and supported version of autotools before proceeding further. After verification, the autotools toolchain (aclocal, autoheader, autoconf, automake) can be used to generate the configuration script. An example output from successfully running the bootstrap script is shown below.

```
-----------------------------------------------------------------
Bootstrap for SHARP build system.
Beginning to run bootstrap in /Users/mahadevan/source/sharp/trunk.
-----------------------------------------------------------------
Scanning dependencies...
     Checking for autoconf........          [ found version 2.69   ]
     Checking for autoheader........        [ found version 2.69   ]
     Checking for aclocal........           [ found version 1.14.1 ]
     Checking for automake........          [ found version 1.14.1 ]
     Checking for libtoolize........        [ found version 2.4.4  ]
     Checking for autoreconf........        [ found version 2.69   ]
-----------------------------------------------------------------
  Found all necessary dependencies.  Proceeding with setup...
-----------------------------------------------------------------
Running the autotools...
     Running autoreconf.....                [   done   ]
-----------------------------------------------------------------
  Done bootstrapping your system.  You may now run ./configure.
  To see options use ./bootstrap --help or view the README file.
-----------------------------------------------------------------
```

Once the configure script is generated, users can invoke "./configure –help" to view and set up the build process with desired set of features. Based on the provided options, the top-level configure script will automatically determine any third-party library dependencies. To satisfy those dependencies, the user may either: (a) link to previously installed libraries on the user's system; or (b) request that the top-level script download and build the necessary libraries. Details on these options are described in the subsections.

4.4 Configuration Options

There are several families of configuration options available when running the configure script, that can be used to control whether certain features are enabled/disabled, or if certain dependencies (pre-installed) need to be used in the current build or if the user wants the package manager to auto-download and configure some of the dependencies.

4.4.1 Enabling/disabling compile-time features or packages

Compile-time features are enabled by options of the form:

```
--enable-<feature>[=yes|no]
--enable-<package>[=yes|no]
```

Values other than yes/no will return an error. --enable-<feature> is synonymous with --enable-<feature>=yes and --disable-<feature> is synonymous with --enable-<feature>=no.

4.4.2 Linking to existing libraries

The user may to link pre-installed libraries with options of the form:

```
--with-<PACKAGE>=PATH
```

where the PATH points to the installation directory typically containing the library and its headers. The PATH argument is mandatory and invalid paths return an error during configuration checks.

If a valid library or dependency has been found, then the configuration for the dependency is processed to see if the required headers are available and if a test program that utilizes the library calls can be successfully compiled and linked in order to accumulate the overall LDFLAGS and LIBS to compile SHARP successfully.

4.4.3 Downloading third-party libraries

SHARP allows the user to download and compile third-party libraries through the configure script. Third-party libraries may be downloaded and compiled with options of the form:

```
--download-<PACKAGE>[=yes|no|url]
```

Specifying --download-library[=yes] will download a tarball (.tar.gz file) of the library's source code from a default URL and utilize default, verified workflows to configure, build and install the dependency onto a dependency installation directory. Running "configure –help" provides a list of the default URLs associated with each library. Optionally, the user can also specify --download-library=URL in order to download the library from the given URL and build/install using the same process. Note that the user-provided URL should point to a valid tarball and may not be guaranteed to be compatible with other SHARP libraries and modules.

Additionally, SHARP accepts two other helper download options (`--download-essential` and `--download-all`) to build a set libraries for some typical configurations. When DIABLO is disabled (by default), the --download-essential option will download and build: MPI, BLAS/LAPACK, ScaLAPACK, PETSc, HDF5, MOAB, ParMetis, Metis, and NetCDF. When DIABLO is enabled (--enable-diablo), the --download-essential option will additionally

build: HYPRE, MUMPS, MILI, and SILO. The --download-all option will build all the above-mentioned libraries in addition to: PNetCDF, Meshkit and NiCE.

4.4.4 Precedence of Options

In several cases, --enable-<package> is available along with a --with-<package> and/or --download-<package> argument. The precedence is resolved according to the table below. The values may be either user-provided or internal default values.

4.4.5 Available options

Table 2. ***Available list of configuration options***

Options	--enable-XXX	--with-XXX	--download-XXX	Description
32bit	**X**			Force 32-bit objects
64bit	**X**			Force 64-bit objects
arpack		**X**		ARPACK
blas		**X**		BLAS
cc		**X**		C compiler
cubit		**X**		Cubit library for MeshKit
cxx		**X**		C++ compiler
debug	**X**			Build with debugging symbols
diablo	**X**	**X**		DIABLO
exodus		**X**		EXODUS
f77		**X**		Fortran77 compiler
fast-install	**X**			Optimize for fast installation
fc		**X**		Fortran compiler
fc-profiling	**X**			Use profiling for the Fortran code
fortran	**X**			Build with Fortran language support
gm		**X**		Prefix where GM is installed (GNOME)
gnu-ld		**X**		Assume C compiler uses GNU linkage

hdf5	X	X	X	HDF5
hdf5-ldflags		X		Extra LDFLAGS for HDF5 library
hypre			X	HYPRE
lapack		X		LAPACK
libtool-lock	X			Avoid locking (might break parallel builds)
meshkit	X	X		MeshKit library
metis	X	X	X	Metis library
metis-include		X		Path for Metis headers
mili		X		MILI
moab	X	X	X	MOAB
moose	X	X		MOOSE
mpi	X	X	X	MPI
mpi-include		X		Path for MPI headers
mpi-libs		X		Path for MPI libraries
mumps	X	X	X	MUMPS
nek	X	X		Nek5000
netcdf		X	X	NetCDF
nice	X	X	X	NiCE
openmp	X			Build with OpenMP support
optimized	X			Compile optimized (-O2)
parmetis	X	X	X	ParMetis
parmetis-include		X		Path for ParMetis headers
parmetis-lib		X		Path for ParMetis libraries
petsc		X	X	PETSc
pic		X		Try to use only PIC/non-PIC objects
pnetcdf		X	X	PNetCDF
shared	X			Build shared libraries
silo		X		SILO

silo-debug	X			Compile and load with debug version of SILO
slepc	X	X		SLEPc
static	X			Build static libraries
szip		X		szip for HDF5
unic	X	X		PROTEUS (formerly UNIC)
zlib		X		zlib for HDF5
zoltan	X		X	ZOLTAN
zoltan-include		X		Path for ZOLTAN headers
zoltan-lib		X		Path for ZOLTAN libraries

4.5 Configuration on the Blues Cluster

4.5.1 Basic Example

One of the simplest configurations is to use a preinstalled MPI library and download all other essential dependencies through SHARP. For example, the user may quickly get up and running on Blues with the following commands:

```
$ export MPI_DIR=/soft/mpich2/1.2.1-intel-11.1
$ ../configure --enable-diablo --download-essential --with-
  mpi=$MPI_DIR CC=$MPI_DIR/bin/mpicc CXX=$MPI_DIR/bin/mpicxx
  F77=$MPI_DIR/bin/mpif77 FC=$MPI_DIR/bin/mpif90
```

These and other possible options are explained below.

4.5.2 MPI libraries and compilers

It is highly recommended to use one of the MPI libraries that are already installed on Blues, even though SHARP provides an option to download and install MPICH. We have successfully built SHARP using the Intel compiler v11.1 with either MPICH2 v1.2 or MVAPICH2 v1.4. On Blues, these library paths are given below.

`/soft/mpich2/1.2.1-intel-11.1/`

OR

`/soft/mvapich2/1.4.1-intel-11.1.064/`

More recent versions are also successful; the most recent available on Blues is Intel v13.1. Ongoing verifications with other available MPI implementations should expand the supported set of both MPI and compiler versions.

The user may find success with other compiler versions or MPI implementations. However, when compiling external libraries from source, the user must choose an implementation with MPI I/O features that support parallel HDF5. See http://www.unidata.ucar.edu/software/netcdf/docs/getting_and_building_netcdf.html#build_parallel and https://www.hdfgroup.org/HDF5/PHDF5 for more info.

In the example above, the library path was set to an environment variable with:

```
$ export MPI_DIR=/soft/mpich2/1.2.1-intel-11.1
```

The library path specified to the configure script by the option:

```
--with-mpi=$MPI_DIR
```

and the compilers was specified by the variables:

```
CC=$MPI_DIR/bin/mpicc CXX=$MPI_DIR/bin/mpicxx
F77=$MPI_DIR/bin/mpif77 FC=$MPI_DIR/bin/mpif90
```

4.5.3 Configuring with Diablo

By default, SHARP enables only PROTEUS and Nek5000 modules. Since the Diablo module requires separate approvals, a decision was made to explicitly enable Diablo only when required. To build DIABLO during SHARP configuration, the --enable-diablo option can be utilized, which will also trigger building several Diablo specific dependencies as needed.

Diablo also requires lexical analyzer (lex) and scanner generator (flex) for its build and so it is essential to also check for these system dependencies when configuring.

4.5.4 Building essential third-party libraries from source

As described above, using the --build-essential flag will download and install MPI, BLAS/LAPACK, ScaLAPACK, PETSc, HDF5, MOAB, ParMetis, Metis, and NetCDF.

4.5.5 Linking to pre-installed libraries

Some libraries required by SHARP are already installed on Blues. These include PETSc, NetCDF, HDF5. (see http://www.lcrc.anl.gov/installed-software for a full list of installed software)

For example, the most recent versions on Blues for Intel and MVAPICH2 are:

- `/soft/netcdf/4.3.1-parallel/intel-13.1/mvapich2-1.9/`
- `/soft/hdf5/1.8.12-parallel/intel-13.1/mvapich2-1.9/`
- `/soft/petsc/3.4.4/intel-13.1/mvapich2-1.9/`

which can be used with the corresponding Intel MVAPICH 2 libs:

```
/soft/mvapich2/1.9-intel-13.1
```

Together, these libs can be used for configuring SHARP with command such as:

```
$ export MPI_DIR=/soft/mvapich2/1.9-intel-13.1
$ ../configure --enable-diablo=yes --download-essential \
--with-mpi=$MPI_DIR \
CC=$MPI_DIR/bin/mpicc CXX=$MPI_DIR/bin/mpicxx F77=$MPI_DIR/bin/mpif77
FC=$MPI_DIR/bin/mpif90 \
--with-netcdf=/soft/netcdf/4.3.1-parallel/intel-13.1/mvapich2-1.9/ \
--with-hdf5=/soft/hdf5/1.8.12-parallel/intel-13.1/mvapich2-1.9/ \
--with-petsc=/soft/petsc/3.4.4/intel-13.1/mvapich2-1.9/
```

4.6 Compiler Choice

For some time, several older compiler versions had unknown compatibility issues with MOAB, which is required for coupled problems. We recently determined that the incompatible compilers were unable to build parallel HDF5, which is a requirement of NetCDF, which is finally a requirement of MOAB. In particular, parallel HDF5 required MPI I/O features that were unsupported by older compilers.

http://www.unidata.ucar.edu/software/netcdf/docs/getting_and_building_netcdf.html#build_parallel

Though NetCDF failed to compile during SHARP's configuration, the top-level configure script did not stop when it encountered this error. In fact, the top-level script only stopped when it encountered the MOAB error. Detecting and presenting the NetCDF error will be fixed in an upcoming revision.

4.7 Updated Documentation

The configuration options are documented in at least two places: (a) in the README file; (b) in the message shown by "configure --help". It was identified that the existing documentation does not provide sufficiently recent or detailed information about configuration options and this is an important concern to address going forward. Some significant omissions included: (a) acceptable values were not described for all arguments; (b) default values were not described for all arguments.

We first elaborated the help messages presented by "configure --help." In many respects, these are more straightforward to maintain than the README. This is because the help messages for "--enable-feature" are specified by the "help string" like below.

```
AC_ARG_ENABLE(feature,help-string,[action-if-given],[action-if-not-given])
```

Hence the descriptions of the actions are provided in the same macro call as the actions themselves. Arguments of the form "--with-feature" are described similarly by AC_ARG_WITH. We updated the help messages to present the user with acceptable values and default values for arguments of the form "--enable-feature[=value]" and "--with-

feature[=value]". It is our intention that the messages in "configure --help" will ALWAYS contain the most up-to date documentation.

The README will also be updated with the same info. Maintaining the README will require more diligence; and we are considering an automated solution, such as Doxygen.

4.8 Standardizing command-line options

The command-line options for the top-level config script follow these conventions:

- Compile-time features are enabled using "--enable-feature[=yes|no]" such as "--enable-mpi[=yes|no]", "--enable-hdf5[=yes|no]", "--enable-openmp[=yes|no]", "--enable-debug[=yes|no]", etc
- Paths to external libraries are specified using "--with-feature=PATH", for example: "--with-mpi=$MPI_DIR", "--with-hdf5=$HDF_DIR", etc

There were several options in the initial configuration implementation that did not follow this format, which were corrected to greatly simplify documentation and reduce user confusion.

4.9 Early checking of command-line options

We've implemented early checks for command-line options. Options that don't follow the above-mentioned format raise an error and halt the configure script. This happens as soon as the arguments are parsed, before installation of any libs. Previously, incorrect options would be caught much later.

5 Open and Recently-Resolved Issues

Parsing Options

Changes were made to interpret command-line options more consistently and correctly. The --enable-diabo flag is now implemented correctly (r2768). The --download-essential option includes the correct third-party dependencies (r2775). The user-provided arguments for optional features and libraries (--enable-feature[=yes|no] and --with-feature=PATH) are validated and raise errors early in the configure process (r2790). The format for these arguments has also been standardized.

Help messages

The configure script's help message now displays default values for all command-line options (r2769).

Detecting and Propagating Variables for Compiler Choices, Flags, Libraries, etc.

The configure script interacts with the MPI libraries more robustly and consistently. When the user does not specify the MPI directory, it is now detected from the environment variable $MPI_DIR and never from $MPI_HOME; this was especially problematic on Blues, since the softenv did not consistently set $MPI_HOME (r2773, r2774).

In several other cases, MPI compiler-related variables were not correctly propagated throughout the configure script, which affected configuration of Metis, ParMetis, NetCDF, PNetCDF, PETSc, and PROTEUS (r2773, r2776, r2777, r2778, r2787).

Variables related to the HDF5, MOOSE, NiCE, and PETSc libraries were also propagated incorrectly (r2782). Variables related to the PROTEUS include paths and compilers were also propagated incorrectly (r2778). Variables related to the MOAB include paths were also propagated incorrectly (r2781).

Overall, these fixes have made configuration much more robust and transparent to the user.

PROTEUS Compilation

Several changes were made to the PROTEUS makefiles to make compilation much more robust (r2783, r2785)

Multiphysics Coupling

Numerous fixes to CouPE in terms of new algorithmic developments and enhancements to the convergence determination in the existing iterative schemes were committed to the repository. Implementation of acceleration schemes for the Picard iterative solver using Aitken and Minimum Polynomial extrapolation methods are also currently underway.

6 Regression testing

SHARP is composed of modules and libraries that are under active development driven through several SciDAC, ASCR and DOE funded projects. Given the rapidly evolving component changes, it is vital to run frequent regression tests, which can reveal whether the existing functionality and feature sets in SHARP have been affected by modifications in the downstream components.

Due to the need for continued automated testing, we have set up both nightly and changeset driven regression test suites using the Buildbot suite (http://buildbot.net/). Currently, there are four specific buildbot builds to stress test the configuration combinations offered by SHARP. This decision has been taken to make the most commonly used options more robust by enabling maximal set of available features and dependency combinations for all modules. Hence, all of the builds use the --download-essential option.

They also differ some time depending on whether optimizations are turned on through the use of the debug flag (--enable-debug) or the optimization flag (--enable-optimized). Since Diablo is also an optional component, the inclusion of DIABLO (--enable-diablo) and it's required dependencies with or without the use of pre-installed packages covers the regular configuration space common to many use cases.

Table 3. ***List of continuous regression builds and tests***

Build	debug	optimized	DIABLO	Preinstalled Packages	Downloaded Packages
sharpdbg-moab	yes	no	no	MPI	HDF5, Metis, MOAB, MPI, NetCDF, ParMetis, PETSc
sharpopt-moab	no	yes	no	MPI	HDF5, Metis, MOAB, MPI, NetCDF, ParMetis, PETSc
sharpdbg-mbmaster	yes	yes	no	HDF5, MOAB, MPI, NetCDF	Metis, ParMetis, PETSc
sharpopt-all	yes	yes	yes	HDF5, MOAB, MPI, NetCDF	HYPRE, Metis, ParMetis, PETSc, SCALAPACK, SILO, MUMPS, MILI

Note that two of the builds (sharpdbg-mbmaster and sharpopt-all) use a pre-installed version of MOAB (along with its corresponding dependencies, HDF5 and NetCDF), rather than the version of MOAB that is downloaded through the SHARP configuration script. The pre-installed version of MOAB is pulled from the current up-to-date master branch of the MOAB repository directly to make sure feature additions in MOAB are always compatible with the SHARP development process.

Fig. 4 shows the continuous regression tests being run in the buildbot system that is currently hosted at http://sigmaserver.mcs.anl.gov:8010. Out of the 4 builds being tested, the figure shows that "**sharpopt-all**" is failing, while the others have consistently passed the regression tests. The failure for the build with Diablo enabled is due to the architecture of the machine with GNU compilers. This is due to a known issue [14] pointed out by Russell Whitesides at LLNL with respect to GNU Fortran 4.8.x compilers, which has not been patched in the machine running the buildbot server.

Such intricacies in the dependency builds are difficult to infer through manual testing and the Buildbot test suites will help SHARP evolve into a portable toolkit that is architecture and machine configuration aware.

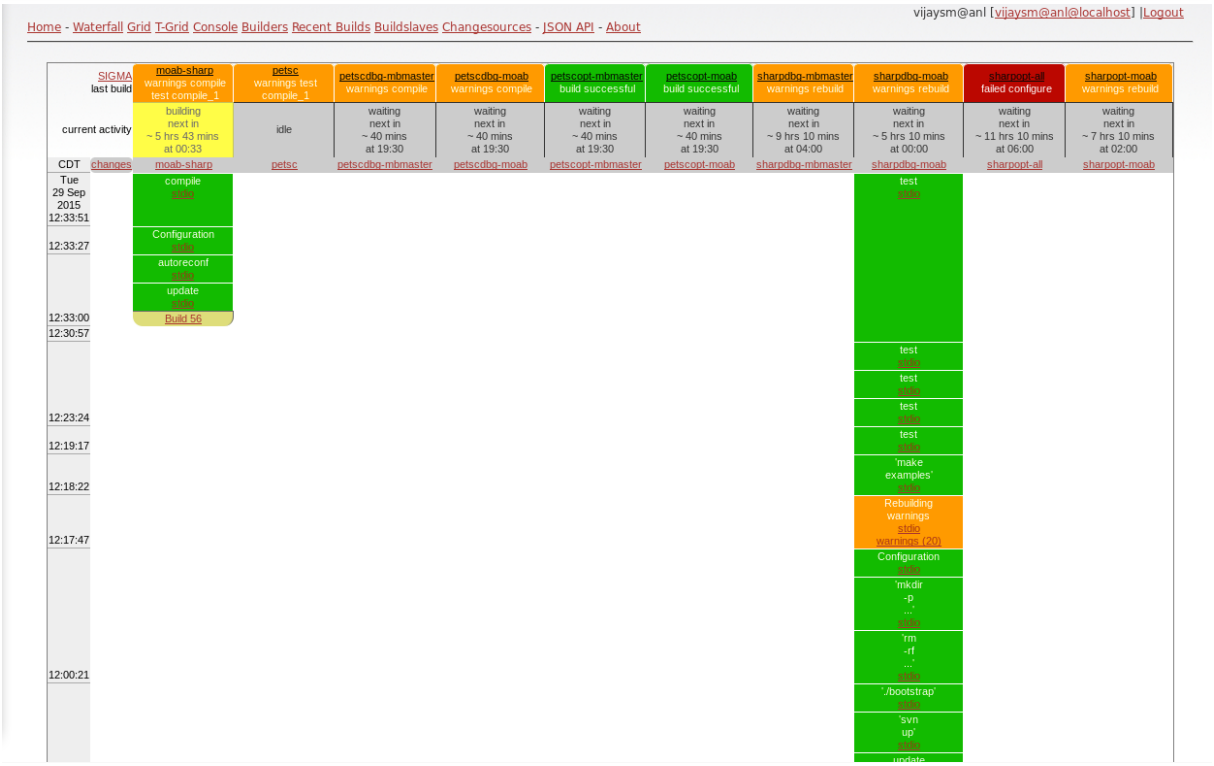

Figure 4. SHARP Architecture Scheme

7 Test problems

The SHARP code repository contains several test problem setups, each of which contains a standard template format for the mesh files, individual physics module input files and driver options. For each of the configurations shown in Table. 3, the following test problems are run through both individual physics drivers and coupled physics drivers.

The list of available test problems in the SHARP toolkit is provided below.

dbgprb: A simple tube problem with verifiable analytical solution

SAHEX: A 6 pin, single hexagonal assembly problem that serves as an integration test

SAHEX_Core: A multi-resolution, homogeneous-heterogenous problem description that contains explicitly represented pin-cell assembly in the middle, surrounded by homogenized assemblies with a duct surrounding it.

XX09: The XX09 assembly with explicit geometry description

XX09_Core: Similar to the SAHEX_Core, the XX09_Core problem represents a multi-resolution core configuration with mixed explicit and homogenized assemblies

ABTR: The full ABTR core represented as homogenized assemblies along with explicit duct representations to evaluate core deformation

Currently, the verification and generation of gold-standard results for all of these problems are being performed. When running the regression suite, these gold-standard results will be used to benchmark the output from the tests after every change in either the SHARP dependencies or the physics modules to catch inconsistencies in code changes.

8 Future plans

Considerable effort has been invested in FY15 to update the configuration and build system for the SHARP toolkit in order to make it substantially portable and to improve the overall computational workflow experience for both new and existing users. The ability to seamlessly tackle explicitly heterogeneous reactor assembly and full core geometry configurations places SHARP in a unique space where high-fidelity nuclear reactor analysis for real world scenarios can be performed. The documentation and user experience have served as the biggest hurdles in the past and recent work to change this will prove beneficial to SHARP.

The continuous integration and test suites for SHARP will help the developers identify and fix several of the pending issues on available architectures, including the LCRC and ALCF machines at Argonne in order to leverage the petascale computing capability to simulate high-fidelity problems at scale. We currently expect that a release process will be in place along with updated user guides and documentation by March 2016. We also plan to provide the first public release of SHARP tarball to interested users during the March time frame. Depending on the FY16 funding allocations, a decision will be made to provide consistent support and hosting tutorial sessions for users to start using the various tools and components in SHARP toolkit in order to analyze nuclear engineering problems of interest.

Acknowledgments

We thank the contributions from various developers of the code modules (PROTEUS – ANL/NE, NEK5000 – ANL/MCS, DIABLO – LLNL) for their continued support. We would also like to thank the SIGMA team for their efforts in supporting a robust open-source workflow from mesh generation to multi-mesh coupling. This work was supported in part by the U.S. Department of Energy Office of Nuclear Energy Nuclear Energy Advanced Modeling and Simulation (NEAMS) Program; by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research; and by the U.S. Department of Energy's Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

REFERENCES

1. A. Siegel, T. Tautges, A. Caceres, D. Kaushik, P. Fischer, G. Palmiotti, M. Smith, J. Ragusa, "Software design of SHARP", Proc of the Joint International Topical Meeting on

Mathematics and Computations and Supercomputing in Nuclear Applications (M&C+SNA), American Nuclear Society, April 2007.

2. Timothy J. Tautges, Hong-Jun Kim, Alvaro Caceres, and Rajeev Jain, “Coupled Multi-Physics simulation frameworks for reactor simulation: A Bottom-Up approach”, Proc. of the International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C), American Nuclear Society, Rio de Janiero, Brazil, May 2011.
3. Derek Gaston, Chris Newman, Glen Hansen, and Damien Lebrun-Grandi, “MOOSE: a parallel computational framework for coupled systems of nonlinear equations”, Nuclear Engineering and Design, 239, 10, 1768–1778, October 2009.
4. Keyes D. E., et al, ”Multiphysics Simulations: Challenges and Opportunities”, International Journal of High Performance Computing Applications, 27, 1, 4-83 (2012).
5. SIGMA: Scalable Interfaces for Geometry and Mesh based Applications, URL: http://sigma.mcs.anl.gov
6. Tautges, T. J. (2005). CGM: a geometry interface for mesh generation, analysis and other applications. Eng. Comput. 17, 486–490.
7. G. D. Sjaardema, T. J. Tautges, T. J. Wilson, S. J. Owen, T. D. Blacker, W. J. Bohnhoff, T. L. Edwards, J. R. Hipp, R. R. Lober, and S. A. Mitchell, CUBIT mesh generation environment volume 1: Users manual, Tech. Report SAND-94-1100, Sandia National Laboratories, 1994.
8. Rajeev Jain and Timothy J. Tautges. Generating unstructured nuclear reactor core meshes in parallel. Procedia Engineering, 82(0):351 – 363, 2014. 23rd International Meshing Roundtable (IMR23).
9. Tautges, T. J., Meyers, R., Merkley, K., Stimpson, C., and Ernst, C. (2004). MOAB: A mesh-oriented database, SAND2004-1592. Sandia National Laboratories, Albuquerque, NM.
10. Timothy J. Tautges, Jason A. Kraftcheck, Nathan Bertram, Vipin Sachdeva, and John Magerlein. Mesh interface resolution and ghost exchange in a parallel mesh representation. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, IPDPSW ’12, pages 1670–1679, Washington, DC, USA, 2012. IEEE Computer Society.
11. Satish Balay, William D. Gropp, Lois Curfman McInnes, Barry F. Smith, “Efficient management of parallelism in object oriented numerical software libraries”, Modern Software Tools in Scientific Computing, 163–202, Birkhäuser Press, 1997.
12. V. S. Mahadevan, "Coupled Physics Environment (CouPE) library – Design, implementation and release", prepared for the U.S. Department of Energy, Office of Nuclear Energy, Milestone Technical Report, ANL/MCS-TM-345, January 2014.
13. Mahadevan VS, Merzari E, Tautges T, Jain R, Obabko A, Smith M, Fischer P., High-resolution coupled physics solvers for analysing fine-scale nuclear reactor design problems, Phil. Trans. R. Soc. A, 2021, 372:20130381, June 2014.

14. Russell Whitesides, Bug 60780 - Equivalence statements in nested modules results in fast growing duplicate statements in module files, URL: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=60780

Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue, Bldg. 240
Argonne, IL 60439

www.anl.gov

Argonne National Laboratory is a U.S. Department of Energy
laboratory managed by UChicago Argonne, LLC