Rocstar Quickstart Guide

Illinois Rocstar LLC

December 16, 2015

www.illinoisrocstar.com

License

Contents

1 Introduction

This quickstart guide provides a brief overview of applications architecture and instructions for how to build and run the *Rocstar* multiphysics simulation application. *Rocstar* is a massively parallel simulation software platform originally designed and implemented by the University of Illinois Center for Simulation of Advanced Rockets (CSAR) to simulate the internal ballistics of solid rocket motors from first principals. Since its original design and implementation, *Rocstar* and its development have been taken up by Illinois Rocstar. *Rocstar* has been applied to many multiphysics problems wherein multiple physical domains interact across moving, reacting interfaces.

An overview of the *Rocstar* Simulation Application architecture is shown in Figure 1. The *Rocstar* simulation application includes a number of components for simulating fluid-structure-combustion interactions and coupling support. A brief overview of *Rocstar* components follows.

Fluid Domains

Rocstar includes two fluid applications designed to simulate compressible multiphase flows.

- *Rocflo* is a block-structured, explicit finite-volume (FV) flow solver with support for LES and RANS turbulence and Lagrangian and Eulerian multiphase flows. *Rocflo* solves the compressible Navier-Stokes equations on moving block-structured grids.
- *Rocflu* is an unstructured, explicit FV flow solver with support for Lagrangian and Eulerian multiphase flows. *Rocflu* solves the compressible Navier-Stokes equations and supports moving geometries.

Structure Domains

Rocfrac is an unstructured computational structural mechanics (CSM) and transient thermal solver with both explicit and implicit options. *Rocfrac* supports several material models including linear elastic, Arruda-Boyce, and Neohookian.

Combustion

Rocburn is a zero and one-dimensional combustion solver designed to implement burn-rate and material heating models. *Rocburn* is capable of simluating material heating, ignition, and regression rates for burning or otherwise reacting surfaces.

Services

Rocstar includes a number of modules for simulation support and orchestration.

- *Rocman* is the main driver and orchestrator for the *Rocstar* simulation application. *Rocman* implements and manages the interactions between the components and control flow of the *Rocstar* simulation application.
- *Rocprop* is a surface propagation service module providing Lagrangian surface tracking and propagation for *Rocstar*. *Rocprop* implements marker-particle, and the more advanced face-offsetting methods for surface propagation.
- *Rocmop* is a mesh optimization service module providing volume mesh smoothing for unstructured meshes. *Rocmop* makes use of a customized and parallelized version of the *Mesquite* mesh optimization framework developed by Sandia National Lab.
- *Rocon* is a propagation constraint service module providing the capability to constrain mesh propagation to geometries specified by triangle surface meshes.
- *Roccom* is the underlying framework and glue which allows *Rocstar* components to export and import data and methods across software component boundaries. *Roccom* also includes internal modules for disk I/O (*Rocin*,*Rocout*), and mesh-aware communication operations (*Rocsurf*, *Rocmap*).
- *Rocface* is a data mapping service module providing transfer services for solution data across disparately discretized surfaces. *Rocface* uses a highly advanced least-squares-data-transfer algorithm based on a common refinement of participating surfaces.

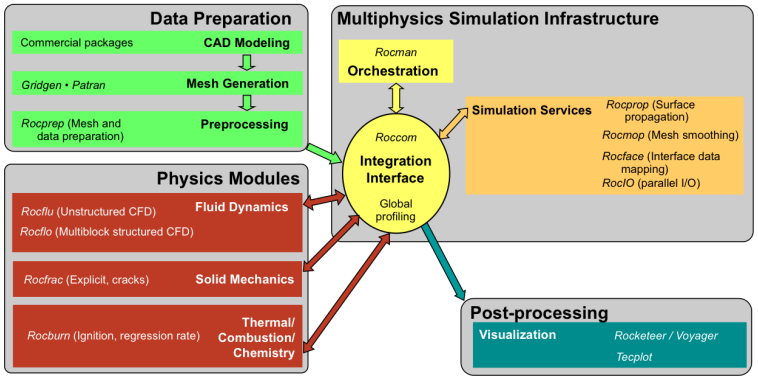

Figure 1: *Rocstar* Simulation Application Architecture

2 Installation

2.1 Prerequisites and Third-Party Libraries

Rocstar has a few dependencies on third-party libraries (TPL). These TPL and other dependencies are summarized below.

- Fortran90 and C++ - *Rocstar* requires F90 and C++ compilers in order to build. *Rocstar* is known to work under most commonly available compilers, but is most extensively tested using those from Intel, and GCC.
- MPI - *Rocstar* uses MPI1 constructs and is known to work under MPICH-derived and OpenMPI implementations.
- BLAS/LAPACK - *Rocstar* requires BLAS and LAPACK. There are no known *Rocstar* defects reported for any BLAS and LAPACK implementations.
- HDF4 - *Rocstar* requires the *HDF4* libraries and development package in order to build. One of HDF4 or CGNS must be available. *HDF4* can be obtained from The HDF Group.* Note that *HDF5* will not work, *Rocstar* requires *HDF4*.
- CGNS - *Rocstar* can use CGNS libraries in place of the HDF4 libraries mentiond above. One of these I/O libraries must be available on the host system.
- Metis-4 - *Rocstar* requires the Metis graph partitioner version 4. Version 5 and above will not work due to non-backwards-compatible changes to the Metis API. Metis-4 can be obtained from Karypis Lab†. Note that Metis version 4 does not include an install target in its build system. After building Metis-4, the library and includes will need to be placed in an install location into lib and include respectively.

*http://www.hdfgroup.org/products/hdf4/
†http://glaros.dtc.umn.edu/gkhome/views/metis

- IRAD - *Rocstar* uses some basic IO and testing functionalities provided in the Illinois Rocstar Application Development toolkit (*IRAD*). This library is provided with *Rocstar*.
- CMake - *Rocstar* uses the CMake build system. CMake is available from Kitware[‡].

2.2 Obtaining *Rocstar*

The easiest way to get *Rocstar* is through its GitHUB project:

```
> git clone https://github.com/IllinoisRocstar/Rocstar
> cd Rocstar
> git submodule update
```

The above commands will grab the latest versions of *Rocstar* and *IRAD*. *IRAD* is a required TPL and is built automatically by the *Rocstar* build process.

2.3 Preliminaries

The following environment variables need to be set before building *Rocstar*.

```
CMAKE_PREFIX_PATH=/full/path/to/metis:/absolute/path/to/tpl
```

Note that CMAKE_PREFIX_PATH should be a colon-separated list of paths to the TPL installation locations if they are in places other than typical system paths (e.g. /usr, /usr/local). CMake will automatically look in ${CMAKE_PREFIX_PATH}/{include,lib,lib64} for needed libraries and include files.

In addition, for the purposes of this guide, the following variables will be used to indicate the the distribution and installation directories.

```
ROCSTAR_SOURCE=/path/to/distribution/directory
ROCSTAR_BUILD=/path/to/installation/directory
ROCSTAR_EXAMPLES=${ROCSTAR_SOURCE}/Examples/NDAs
```

Note that ROCSTAR_SOURCE **should not** be the same path as ROCSTAR_BUILD. If it does not exist, create the build directory and cd into it with the following commands.

```
> mkdir ${ROCSTAR_BUILD}
> cd ${ROCSTAR_BUILD}
```

By default, *Rocstar* will build the *IRAD* library at compile time if it is located in a folder named IRAD in the top level of the source tree. If the user has a pre-existing install of *IRAD* that they would prefer to use instead, the path to it can be specified at compile time by including the full path to *IRAD* in the CMAKE_PREFIX_PATH variable. .

[‡] http://www.kitware.com/cmake

2.4 Build *Rocstar*

Invoke CMake to configure *Rocstar* with the following command.

```
> cmake ${ROCSTAR_SOURCE}
```

Any errors in this step (e.g. failing to find TPL or compilers) must be resolved before continuing. If there are no errors then *Rocstar* can now be built with the following command.

```
> make
```

Building *Rocstar* can take a while (e.g. approximately 20 minutes on a modern workstation). Once the build is complete with no errors, then *Rocstar* will be ready for testing. Before testing, the following environment should be set.

```
PATH=${ROCSTAR_BUILD}/bin:${PATH}
LD_LIBRARY_PATH=${ROCSTAR_BUILD}/lib:${LD_LIBRARY_PATH}
```

Note that if parallel runs through a batch system or on a cluster are planned, then these environment modifications should be available in the default shell so that the runtime system will be able to find the required *Rocstar* executables and libraries.

3 Test *Rocstar*

These are just a couple of very quick tests to make sure *Rocstar* has been properly built and installed. Create a directory in which to test *Rocstar* and set an environment variable to indicate where it is. This path is arbitrary.

```
ROCSTAR_TESTING=/path/to/rocstar/testing

> mkdir ${ROCSTAR_TESTING}
> cd ${ROCSTAR_TESTING}
```

3.1 Quick Test

This quick test will just make sure that *Rocstar* can find and load all of its modules and that all paths are set properly.

```
> cp ${ROCSTAR_SOURCE}/Control/RocstarTest.txt ./RocstarControl.txt
> rocstar
```

This should print some splash information to the screen, and return without errors. If *Rocstar* returns errors, please check your installation procedure and environment against the above instructions. Note that **warnings** are OK and unavoidable at this stage. We only check to see if the built modules can be located and dynamically loaded. As long as there are no **errors**, then this test has succeeded.

3.2 Automated Testing

Rocstar comes pre-packaged with a set of automated tests implemented with *IRAD* and *CTest* (included with *CMake*). These tests can be invoked from the command after the compilation is complete.

```
> make test
```

The test results are reported to the screen.

3.3 Rocket Test

This example will test a subset of *Rocstar* preprocessor codes and a quick runtime example of a burning rocket. To preprocess the data, enter the following command.

```
> rocprep -A -b -u 1 1 -o 1 1 -n 4 -d ${ROCSTAR_EXAMPLES}/ACM -t ./ACM_4
```

This should return with no errors and should have created the `ACM_4` directory. If not, then something has gone wrong with the install. If so, cd into the ACM_4 directory to continue the test.

```
> cd ACM_4
```

Here you will find a `RocstarControl.txt` file. Edit the control file with your favorite editor and change the `FluidModule` line from `Rocflo` to `Rocflu`.

Next, edit the `Rocflu/Modin/ACM.inp` file and change the `ORDER` parameter in the `NUMERICS` section from **2** to **1**. This tells *Rocflu* to run in 1st order mode instead of 2nd order mode.

From the top level `ACM_4` directory, start *Rocstar* in parallel on 4 processors with the following command.

```
> mpirun -np 4 rocstar
```

This should run the Attitude Control Motor (ACM) using the *Rocflu* fluid module, which tests a large number of *Rocsar* functionalities. If running through a batch system, or if the runtime system has trouble finding the rocstar executable in parallel, then it can be linked to the current directory with the following command.

```
> ln -s ${ROCSTAR_BUILD}/bin/rocstar .
```

and then run with

```
> mpirun -np 4 ./rocstar
```

If running through a batch system, or on a system that does not allow direct invocation of `mpirun`, then alter the above commands with a parallel job spawning mechanism supported by your platform.