INL/EXT-14-31222

Pervasive Restart In MOOSE-based Applications

Derek Gaston
Cody Permann
David Andrs
John Peterson
Andrew Slaughter
Jason Miller

January 2014

INL/EXT-14-31222

Pervasive Restart In MOOSE-based Applications

Derek Gaston
Cody Permann
David Andrs
John Peterson
Andrew Slaughter
Jason Miller

January 2014

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

1. Introduction

Multiphysics applications are inherently complicated. Solving for multiple, interacting physical phenomena involves the solution of multiple equations, and each equation has its own data dependencies. Feeding the correct data to these equations at exactly the right time requires extensive effort in software design.

In an ideal world, multiphysics applications always run to completion and produce correct answers. Unfortunately, in reality, there can be many reasons why a simulation might fail: power outage, system failure, exceeding a runtime allotment on a supercomputer, failure of the solver to converge, etc. A failure after many hours spent computing can be a significant setback for a project. Therefore, the ability to "continue" a solve from the point of failure, rather than starting again from scratch, is an essential component of any high-quality simulation tool. This process of "continuation" is commonly termed "restart" in the computational community.

While the concept of restarting an application sounds ideal, the aforementioned complexities and data dependencies present in multiphysics applications make its implementation decidedly non-trivial. A running multiphysics calculation accumulates an enormous amount of "state": current time, solution history, material properties, status of mechanical contact, etc. This "state" data comes in many different forms, including scalar, tensor, vector, and arbitrary, application-specific data types. To be able to restart an application, you must be able to both store and retrieve this data, effectively recreating the state of the application before the failure.

When utilizing the Multiphysics Object Oriented Simulation Environment (MOOSE) framework developed at Idaho National Laboratory, this state data is stored both internally within the framework itself (such as solution vectors and the current time) and within the applications that use the framework. In order to implement restart in MOOSE-based applications, the total state of the system (both within the framework and without) must be stored and retrieved. To this end, the MOOSE team has implemented a "pervasive" restart capability which allows any object within MOOSE (or within a MOOSE-based application) to be declared as "state" data, and handles the storage and retrieval of said data.

2. Design

One approach to restarting applications is to directly dump the memory image of the application to disk, and read that image back upon restart. This approach is problematic for many reasons, but perhaps the biggest is the excessive amount of disk-space it requires. The Fission supercomputer at INL contains over 300 nodes, and each node has 64GB of system memory, for a total of nearly 20TB. If a simulation were using the entire machine, it would need to dump 20TB of data to disk *every time step* to effectively implement the restart capability. Not only would this take an enormous amount of disk space, it would also require an inordinate amount of time, possibly even crashing the entire system in the process! An alternative to this approach is to write out only the data that is truly necessary. This involves identifying specific data values within the framework and application that are changing over time and must be recovered. The MOOSE restart system is based on this concept.

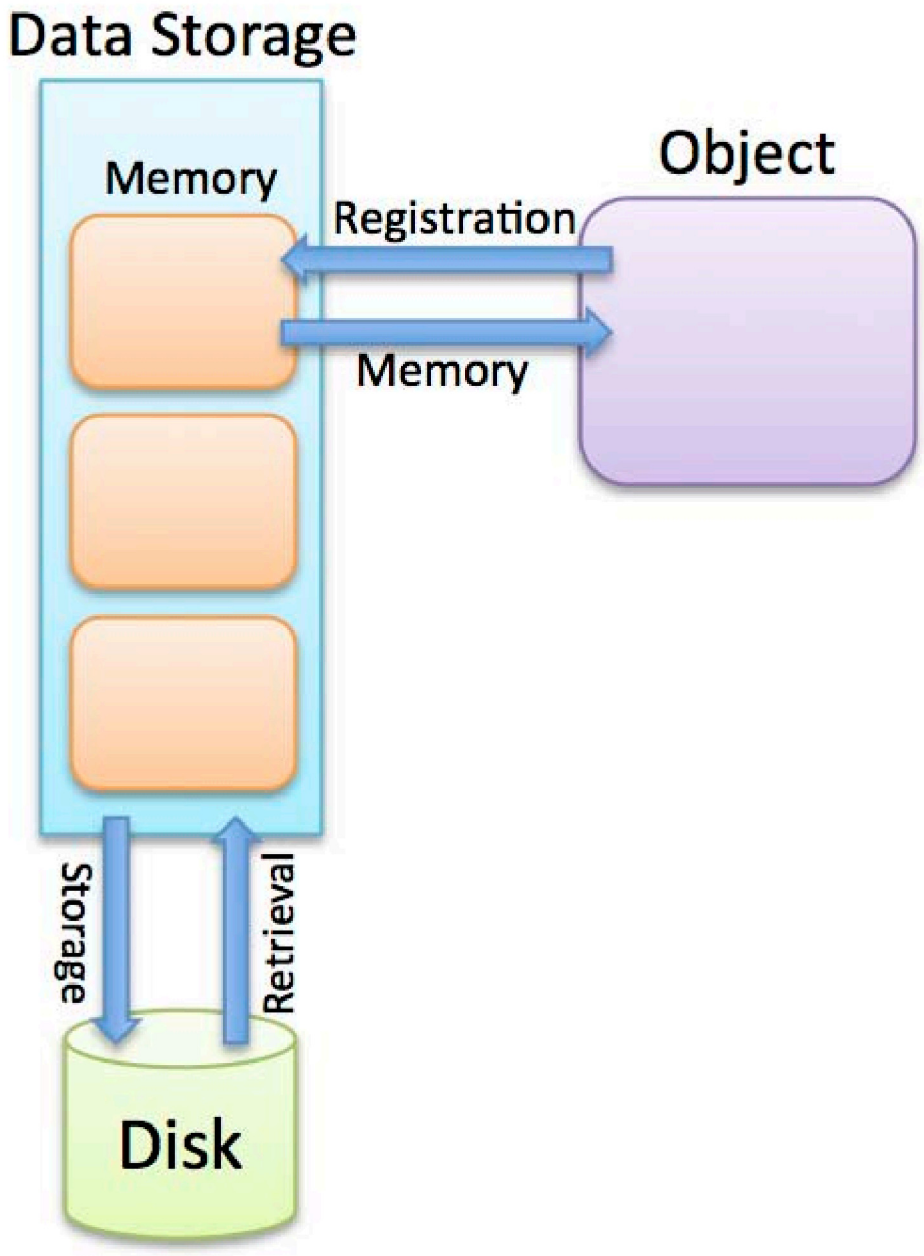
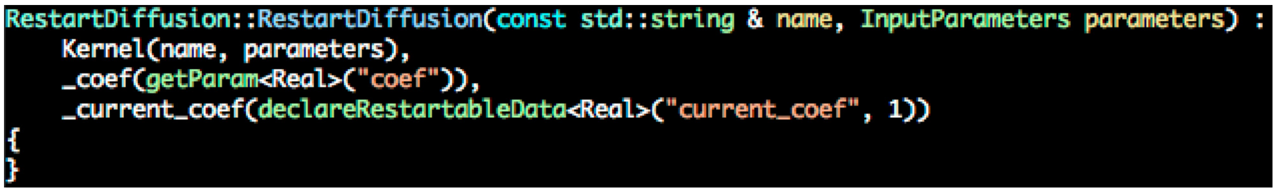


Figure 1. *New restart system components*

Figure 1 schematically shows the main components of the new restart system. The "Object" in Figure 1 can be any object within MOOSE or within a MOOSE-based application. During initialization of the object, a function is called that registers data with a separate "Data Storage" object in MOOSE. This is how the data to be stored and retrieved is identified. During the registration function call, the memory necessary to hold that piece of data is allocated and a reference to that memory is returned to the Object. From this point on, the Object can use that memory as though it were a piece of data within the Object itself, while in reality the data is actually stored within the Data Storage object. This provides a seamless experience for MOOSE application developers.



```
RestartDiffusion::RestartDiffusion(const std::string & name, InputParameters parameters) :
    Kernel(name, parameters),
    _coef(getParam<Real>("coef")),
    _current_coef(declareRestartableData<Real>("current_coef", 1))
{
}
```

Figure 2. *Code example using the new system*

Figure 2 shows a code example that utilizes this new system. The object's name is RestartDiffusion, and it is registering a scalar (Real) piece of data named "current_coef" by calling the declareRestartableData() function. That function returns a reference that is stored by the object in a member variable called _current_coef. All of the code within the RestartDiffusion class can utilize _current_coef as though it were a normal member variable of the class, accessing and setting the value as needed. The value itself is actually stored inside the "Data Storage" object in MOOSE, but operating on it is seamless.

In order to store the data to disk, the Data Storage object is asked at regular intervals to write out everything it knows about. This normally occurs every timestep, but can be controlled by the user. The storage of this data to disk is implemented using C++ template functions with template specializations. The data is written out in a machine-readable binary format to minimize disk space requirements. Developers can also create template specializations for their own data types. The Data Storage object will then utilize these specializations while writing out data. Writers and readers for the most common data structures are provided in MOOSE, only on rare occasions should it be necessary for a developer to create a custom writer.

In the event that a simulation fails, a user of a MOOSE-based code can rerun the application with the command-line switch "--recover", and MOOSE will automatically read the files written by the Data Storage object from disk, restoring all of the values to the objects which require them. Application developers need only register their state data with the system to take advantage of this capability.

3. Impact

The new restart system within MOOSE is already having a large impact on NEAMS applications. The BISON team is not only using it to recover solves that failed because of hardware and solver convergence issues, but also to accelerate comparisons to experimental results. For instance, they can perform a base irradiation of fuel out to a certain time, saving the restart files generated, then continue that calculation in a number of different environments to simulate the various post-irradiation tests. The RELAP-7 and RAVEN teams are also heavily utilizing the new restart system. RAVEN, in particular, is using restart to "branch" simulations down multiple execution paths as discrete events occur. This can accelerate probabilistic risk assessment (PRA) work.

Finally, the MARMOT team is already very reliant upon the restart capability. MARMOT simulations can contain hundreds of millions of degrees of freedom and take multiple days to execute on a cluster. As the length of a simulation grows, so does the chance of failure. This new restart system gives them the assurance that they can complete long-running complex calculations.

4. Future

The restart system is still in its infancy. Even though the design has already proven to work well for multiple application teams, modifications and enhancements will need to be made. For instance, some effort is needed to ensure that the parallel I/O routines used by the system are as efficient as they can be. Nevertheless, the groundwork has been laid and a new tool has been added to MOOSE-based development teams' belts, allowing them to work more efficiently and better cope with the imperfect world of massively parallel, multiphysics simulation.