The Cactus Framework (Cactus) is an open-source environment for numerically solving Cauchy problems on a cluster. It provides a foundation for computational toolkits, such as the Einstein Toolkit. Using Cactus, application modules called thorns can be designed and tested on workstations and then easily run on clusters. Within a thorn, scheduling information is provided by the programmer for each scheduled subroutine.

The current Cactus scheduling system relies on the manual synchronization of ghost zones for grid functions (i.e. distributed matrices). Unfortunately, deciding which subroutines should synchronize which grid functions is a non-trivial problem requiring an in-depth understanding of how Cactus schedules synchronization. Incorrect synchronization can result in over-synchronization (a performance problem) or under-synchronization (an error). This issue also creates a barrier for new users and collaborators.

In addition to synchronization issues, boundary conditions are handled separately from synchronization in Cactus. This is unfortunate, because both synchronization and boundary conditions are mechanisms for filling in the outer cells of distributed grids and are naturally applied at the same time. In the case of the Einstein Toolkit, the Boundary thorn handles physical boundary conditions, while the SymBase thorn manages symmetry boundary conditions. Regardless of the type of boundary, however, boundary updates are scheduled manually. Many thorns also introduce additional boundary condition subroutines via internal methods. Scheduling of the subroutines provided by the Boundary thorn – and, by extension, any subroutines registered with SymBase – is also left to the individual thorn writer. This paradigm leads to similar issues as with synchronization. The implementation of ghost zone synchronization and application of boundary conditions in the Einstein Toolkit both place an unnecessary burden on thorn writers.

As part of NSF SI2 grant OAC-1550551, we are developing a new scheduling and synchronization method for Cactus to improve the programmabillity and efficiency of the Einstein Toolkit. As part of this project (referred to hereafter as PreSync), we are also integrating the application of boundary conditions into synchronization. Synchronization and application of boundary conditions are now handled automatically by Cactus. The new approach requires each subroutine to have “read” and “write” declarations for individual grid functions. The read/write declarations for grid functions identify the region of validity, which can be “interior” or “everywhere”. “Write” declarations change the grid function’s region of validity to the specified region, and “read” declarations require the grid function to be valid for the specified region. Before scheduled functions run, Cactus checks these declarations and, if a grid function is only valid on the interior but is needed everywhere, performs synchronization and applies boundary conditions as needed.

This method removes the difficulty of deciding where synchronization should take place and removes unneeded synchronizations. Programmers need only declare what grid functions a given subroutine uses and on which parts of the grid they are read or written. Cactus includes all the infrastructure to handle boundaries; thorns which provide boundary conditions simply register their boundary conditions with Cactus. While the

Boundary thorn in the Einstein Toolkit has been refitted to work with the new boundary method, many thorns with internal boundary condition application are still being revised to use PreSync.

PreSync does not prevent the old synchronization mechanism from functioning, and manual synchronization can still be used. However, the old boundary condition subroutines will not properly update the region of validity, possibly generating calculation errors or causing over-synchronization. For synchronization, manually updating the region of validity within the subroutine is sufficient, but all boundary routine code should be refactored to use the new system to benefit from all the changes PreSync provides.

While the subroutines of most thorns have simple read/write declarations, some subroutines have more elaborate behavior which needs more advanced treatment. As an example, the hydrodynamics evolution thorn GRHydro has parameters which can be changed at run time deciding whether magnetic fields should be evolved. The read/write declarations for a subroutine are normally given at compile time, which means that the declarations themselves cannot take this choice into account. To resolve this discrepancy, logical if statements are used in the scheduling, with the same subroutine being scheduled with different read/write declarations depending on the state of the magnetic field parameter.

The I/O thorns exhibit an even more extreme version of this behavior – any number of grid functions could be chosen at run time to write to files. Most grid functions are valid “everywhere” when the simulation reaches the output stage. However, some thorns have output grid functions which are only written and never read. Since they are never read, synchronization and boundary condition application never triggers. To handle this case, the I/O thorns need to call special subroutines to check the region of validity and trigger synchronization if necessary. PreSync provides functions for this purpose, and the I/O thorns in the Einstein Toolkit have been updated to use these features.

The greatest burden for the transition to PreSync is adapting old thorns to use the new system. To assist with this endeavor, we are experimenting with various mechanisms to provide feedback and error checking for read/write declarations. These features will also benefit current thorns writers during and after the transition into the new method. Currently, Cactus gives all subroutines read and write access to all grid functions. We employ automatic code generation to create macros which, instead, limit the grid functions accessible to each subroutine to match the declared read/write specifications. Therefore, any grid functions which are not given in the read/write declarations will result in a compilation error. Additionally, read-only grid functions are declared as constants, so writing the grid function will also cause a compilation error. This method can fail when a subroutine calls another subroutine outside the normal Cactus scheduler, as the compiler does not know if the called subroutine reads or writes the grid function within it. In addition,

attempts to access grid functions which have not properly been declared can be detected at run time, as Cactus will set the pointers to these grid functions to null when they should not be accessed.

To provide a check for correct region access (i.e. “interior” vs. “everywhere”), diagnostic tools are in production to determine which grid functions are written by each subroutine at run time. The primary tool uses checksums to determine if a grid function has been written, and whether it was written in the “interior” or “everywhere”. However, the current method does not have a perfect detection rate, and additional changes are needed to move the diagnostics to release-quality.

The PreSync project makes programming Cactus easier. Scheduling a subroutine only requires knowledge of that one subroutine, removing the need for thorn writers to understand the details of how a routine fits into the schedule as a whole. Also, deciding the read/write declarations for a subroutine is usually a straightforward process, while the proper placing of sync statements is frequently a non-trivial problem. For backward compatibility, PreSync also allows for the old synchronization method to be used simultaneously, and parameters exist to switch these two methods on and off at runtime.

This new system is a first step in improving Cactus’ scheduling methods to use more advanced and scalable techniques. Many old thorns must be upgraded to use PreSync. Once this is accomplished, we will perform additional tests to validate its successful implementation and also compare performance of Cactus with and without PreSync. We also aim to create better diagnostic tools to help assess whether the given read/write declarations are correct for a subroutine. A longer term goal is to use read/write declarations to order the scheduled routines (where possible).