PUGHInterp

Paul Walker, Thomas Radke, Erik Schnetter

Date: 2004/06/20 12:31:17

Abstract

Thorn *PUGHInterp* implements the Cactus interpolation API `CCTK_InterpGridArrays()` for the interpolation of CCTK grid arrays at arbitrary points.

1 Introduction

Thorn *PUGHInterp* provides an implementation of the Cactus interpolation API specification for the interpolation of CCTK grid arrays at arbitrary points, `CCTK_InterpGridArrays()`.

This function interpolates a list of CCTK grid arrays (in a multiprocessor run these are generally distributed over processors) on a list of interpolation points. The grid topology and coordinates are implicitly specified via a Cactus coordinate system. The interpolation points may be anywhere in the global Cactus grid. In a multiprocessor run they may vary from processor to processor; each processor will get whatever interpolated data it asks for.

The routine `CCTK_InterpGridArrays()` does not do the actual interpolation itself but rather takes care of whatever interprocessor communication may be necessary, and – for each processor's local patch of the domain-decomposed grid arrays – calls `CCTK_InterpLocalUniform()` to invoke an external local interpolation operator (as identified by an interpolation handle). It is advantageous to interpolate a list of grid arrays at once (for the same list of interpolation points) rather than calling `CCTK_InterpGridArrays()` several times with a single grid array. This way note only can *PUGHInterp*'s implementation of `CCTK_InterpGridArrays()` aggregate communications for multiple grid arrays into one (resulting in less communications overhead) but also `CCTK_InterpLocalUniform()` may compute interpolation coefficients once and reuse them for all grid arrays.

Please refer to the *Cactus UsersGuide* for a complete function description of `CCTK_InterpGridArrays()` and `CCTK_InterpLocalUniform()`.

2 *PUGHInterp*'s Implementation of `CCTK_InterpGridArrays()`

If thorn *PUGHInterp* was activated in the `ActiveThorns` list of a parameter file for a Cactus run, it will overload at startup the flesh-provided dummy function for `CCTK_InterpGridArrays()` with its own routine. This routine will then be invoked in subsequent calls to `CCTK_InterpGridArrays()`.

PUGHInterp's routine for the interpolation of grid arrays provides exactly the same semantics as `CCTK_InterpGridArrays()`which is thoroughly described in the *Function Reference* chapter of the *Cactus UsersGuide*. In the following, only user-relevant details about its implementation, such as specific error codes and the evaluation of parameter options table entries, are explained.

2.1 Implementation Notes

At first, `CCTK_InterpGridArrays()` checks its function arguments for invalid values passed by the caller. In case of an error, the routine will issue an error message and return with an error code of either `UTIL_ERROR_BAD_HANDLE` for an invalid coordinate system and/or parameter options table, or `UTIL_ERROR_BAD_INPUT` otherwise. Currently there is the restriction that only `CCTK_VARIABLE_REAL` is accepted as the CCTK data type for the interpolation points coordinates.

Then the parameter options table is parsed and evaluated for additional information about the interpolation call (see section 2.2 for details).

In the single-processor case, `CCTK_InterpGridArrays()` would now invoke the local interpolation operator (as specified by its handle) by a call to `CCTK_InterpLocalUniform()` to perform the actual interpolation. The return code from this call is then also passed back to the user.

For the multi-processor case, *PUGHInterp* does a query call to the local interpolator first to find out whether it can deal with the number of interprocessor ghostzones available. For that purpose it sets up an array of two interpolation points which denote the extremes of the physical coordinates on a processor: the lower-left and upper-right point of the processor-local grid's bounding box[1]. The query gets passed the same user-supplied function arguments as for the real interpolation call, apart from the interpolation points coordinates (which now describe a processor's physical bounding box coordinates) and the output array pointers (which are all set to NULL in order to indicate that this is a query call only). A return code of `CCTK_ERROR_INTERP_POINT_OUTSIDE` from `CCTK_InterpLocalUniform()` for this query call (meaning the local interpolator potentially requires values from grid points which are outside of the available processor-local patch of the global grid) causes `CCTK_InterpGridArrays()` to return immediately with a `CCTK_ERROR_INTERP_GHOST_SIZE_TOO_SMALL` error code on all processors.

Otherwise the `CCTK_InterpGridArrays()` routine will continue and map the user-supplied interpolation points onto the processors which own these points. In a subsequent global communication all processors receive "their" interpolation points coordinates and call `CCTK_InterpLocalUniform()` with those. The interpolation results are then sent back to the processors which originally requested the interpolation points.

Like the *PUGH* driver thorn, *PUGHInterp* uses MPI for the necessary interprocessor communication. Note that the `MPI_Alltoall()/MPI_Alltoallv()` calls for the distribution of interpolation points coordinates to their owning processors and the back transfer of the interpolation results to the requesting processors are collective communication operations. So in the multi-processor case you *must* call `CCTK_InterpGridArrays()` in parallel on *each* processor (even if a processor doesn't request any points to interpolate at), otherwise the program will run into a deadlock.

2.2 Passing Additional Information via the Parameter Table

One of the function arguments to `CCTK_InterpGridArrays()` is an integer handle which refers to a key/value options table. Such a table can be used to pass additional information (such as the interpolation order) to the interpolation routines (i.e. to both `CCTK_InterpGridArrays()` and the local interpolator as invoked via `CCTK_InterpLocalUniform()`). The table may also be modified by these routines, eg. to exchange internal information between the local and global interpolator, and/or to pass back arbitrary information to the user.

The only table option currently evaluated by *PUGHInterp*'s implementation of `CCTK_InterpGridArrays()` is:

```
CCTK_INT input_array_time_levels[N_input_arrays];
```

which lets you choose the timelevels for the individual grid arrays to interpolate (in the range $[0, N\_time\_levels\_of\_var\_i - 1]$). If no such table option is given, then the current timelevel (0) will be taken as the default.

The following table options are meant for the user to specify how the local interpolator should deal with interpolation points near grid boundaries:

```
CCTK_INT  N_boundary_points_to_omit[2 * N_dims];
CCTK_REAL boundary_off_centering_tolerance[2 * N_dims];
CCTK_REAL boundary_extrapolation_tolerance[2 * N_dims];
```

In the multi-processor case, `CCTK_InterpGridArrays()` will modify these arrays in a user-supplied options table in order to specify the handling of interpolation points near interprocessor boundaries (ghostzones) for the local interpolator; corresponding elements in the options arrays are set to zero for all ghostzone

[1] Note that because the query is done with extreme interpolation points coordinates, the interpolation call may fail even if all the user-supplied interpolation points are well within each processor's local patch. The reason for this implementation behaviour is that we safely want to catch all errors caused by a too small ghostzone size.

faces, i.e. no points should be omitted, and no off-centering and extrapolation is allowed at those boundaries. Array elements for physical grid boundaries are left unchanged by `CCTK_InterpGridArrays()`.

If any of the above three boundary handling table options is missing in the user-supplied table, `CCTK_InterpGridArrays()` will create and add it to the table with appropriate defaults. For the default values, as well as a comprehensive discussion of grid boundary handling options, please refer to documentation of the thorn(s) providing local interpolator(s) (eg. thorn *LocalInterp* in the *Cactus ThornGuide*).

At present, the table option `boundary_extrapolation_tolerance` is not implemented. Instead, if any point cannot be mapped onto a processor (i.e. the point is outside the global grid), a level-1 warning is printed to stdout by default, and the error code `CCTK_ERROR_INTERP_POINT_OUTSIDE` is returned. The warning will not be printed if the parameter table contains an entry (of any type) with the key `"suppress_warnings"`.

The local interpolation status will be stored in the user-supplied parameter table (if given) as an integer scalar value with the option key `"local_interpolator_status"` (see section 2.3 for details).

The table options

```
CCTK_POINTER per_point_status;
CCTK_INT     error_point_status;
```

are used internally by `CCTK_InterpGridArrays()` to pass information about per-point status codes between the global and the local interpolator (again see section 2.3 for details).

2.3 `CCTK_InterpGridArrays()` Return Codes

The return code from `CCTK_InterpGridArrays()`[2]is determined as follows:

- If any of the arguments are invalid (e.g. `N_dims` < 0), the return code is `UTIL_ERROR_BAD_INPUT`.
- If any errors are encountered when processing the parameter table, the return code is the appropriate `UTIL_ERROR_TABLE_*` error code.
- If the query call determines that the number of ghost zones in the grid is too small for the local interpolator, the return code is `CCTK_ERROR_INTERP_POINT_OUTSIDE`.
- Otherwise, the return code from `CCTK_InterpGridArrays()` is the minimum over all processors of the return code from the local interpolation on that processor.

If the local interpolator supports per-point status returns and the user supplies an interpolator parameter table, then in addition to this global interpolation return code, `CCTK_InterpGridArrays()` also returns a "local" status code which describes the outcome of the local interpolation for all the interpolation points which originated on *this* processor:

```
CCTK_INT local_interpolator_status;
```

This gives the minimum over all the interpolation points originating on *this* processor, of the `CCTK_InterpLocalUniform()` return codes for those points. (It doesn't matter on which processor(s) the points were actually interpolated – `CCTK_InterpGridArrays()` takes care of gathering all the status information back to the originating processors.)

3 Comments

For more information on how to invoke interpolation operators please refer to the flesh documentation.

[2]In C the return code is the `CCTK_InterpGridArrays()` function result; in Fortran it's returned through the first (`status`) argument.