# PUGHSlab

Gabrielle Allen, Tom Goodale, Thomas Radke,
with many comments and suggestions from Erik Schnetter and Jonathan Thornburg

October 2001

**Abstract**

Thorn PUGHSlab implements the generic hyperslab data extraction API for CCTK arrays.

## 1 Introduction

Many I/O thorns output data from distributed CCTK array variables. If – in a multiprocessor run – output is done by only one processor, it needs to collect the data from the others. This ties the I/O thorn to the driver since it has to know about domain-decomposed data layout, interprocessor ghostzones, etc.

A clean way of separating the I/O code from the driver is to use a thorn which provides a generic interface to get/put the distributed data on/from the I/O processor for writing/reading. This interface can also provide more features such as downsampling, datatype conversions, or hyperslab selections. A hyperslab is defined in this context a subset of a global CCTK array, with its own dimension, origin, direction, and extents.

Another possible use of hyperslabs is the implementation of certain boundary conditions (e.g. reflection). By having the boundary condition code calling generic hyperslab get/put calls, it can be written without special knowledge about driver specifics.

This thorn documentation describes the complete generic hyperslab API. All routines use CCTK data types in their argument lists and as return codes exclusively. This allows actual implementations of this API to be realized as CCTK function aliases. Different hyperslab thorns can then implementing the API using the same function names, and other thorns using the API can be independent of the actual hyperslab thorns which are compiled in.

The current version of thorn `PUGHSlab` implements only parts of the CCTK hyperslab API. Please refer to section 3 for implementation details.

## 2 CCTK Hyperslab API Specification

In general, a hyperslab get/put operation is done in a three-level scheme:

1. In a first step, a hyperslab mapping is defined by calling one of the following routines:
   `Hyperslab_LocalMappingByIndex()`
   `Hyperslab_LocalMappingByPhys()`
   `Hyperslab_GlobalMappingByIndex()`
   `Hyperslab_GlobalMappingByPhys()`
   There exists two complement sets of routines: one for the definition of local hyperslabs (which apply to a processor's local patch of a distributed grid variable only), and one for global hyperslabs (which spawn the entire grid).

   A mapping can be defined by either physical coordinates or by grid index points.

   All hyperslab mapping routines return an integer handle which refers to an internally allocated data structure describing the defined hyperslab.

2. With such a mapping, hyperslabs can then be extracted/distributed by one or more calls to
   `Hyperslab_Get()`
   `Hyperslab_GetList()`
   `Hyperslab_Put()`
   `Hyperslab_PutList()`
   There are routines for getting/putting a hyperslab from/to a single grid variable or from/to a list of variables.

3. Once all hyperslabs are done, the hyperslab mapping should be freed by a call to
   `Hyperslab_FreeMapping()`.

If the `Hyperslab_Get*()/Hyperslab_Put*()` get passed a mapping for a global hyperslab, a global, synchronuous operation will be performed (i.e., they must be called in sync by every processor). All input arguments must be consistent between processors.

## 2.1 Defining a hyperslab mapping

An $M$-dimensional hyperslab subspace mapped into an $N$-dimensional space can be specified either by coordinates on the physical grid or by index points on the underlying computational grid.

```
CCTK_INT Hyperslab_GlobalMappingByIndex (
           CCTK_POINTER_TO_CONST GH,
           CCTK_INT              vindex,
           CCTK_INT              hdim,
           const CCTK_INT       *direction         /* vdim*hdim */,
           const CCTK_INT       *origin            /* vdim */,
           const CCTK_INT       *extent            /* hdim */,
           const CCTK_INT       *downsample        /* hdim */,
           CCTK_INT              table_handle,
           CCTK_FPOINTER         conversion_fn,
           CCTK_INT             *hsize             /* hdim */);

CCTK_INT Hyperslab_GlobalMappingByPhys (
           CCTK_POINTER_TO_CONST GH,
           CCTK_INT              vindex,
           CCTK_INT              hdim,
           CCTK_STRING           coord_system_name,
           const CCTK_INT       *direction         /* vdim*hdim */,
           const CCTK_REAL      *origin            /* vdim */,
           const CCTK_REAL      *extent            /* hdim */,
           const CCTK_INT       *downsample        /* hdim */,
           CCTK_INT              table_handle,
           CCTK_FPOINTER         conversion_fn,
           CCTK_INT             *hsize             /* hdim */);


CCTK_INT Hyperslab_LocalMappingByIndex (
           CCTK_POINTER_TO_CONST GH,
           CCTK_INT              vindex,
           CCTK_INT              hdim,
           const CCTK_INT       *direction         /* vdim*hdim */,
           const CCTK_INT       *origin            /* vdim */,
           const CCTK_INT       *extent            /* hdim */,
           const CCTK_INT       *downsample        /* hdim */,
           CCTK_INT              table_handle,
           CCTK_FPOINTER         conversion_fn,
           CCTK_INT             *hsize_local,      /* hdim */
```

```
           CCTK_INT             *hsize_global,     /* hdim */
           CCTK_INT             *hoffset_global    /* hdim */);

CTK_INT Hyperslab_LocalMappingByPhys (
           CCTK_POINTER_TO_CONST GH,
           CCTK_INT              vindex,
           CCTK_INT              hdim,
           CCTK_STRING           coord_system_name,
           const CCTK_INT       *direction         /* vdim*hdim */,
           const CCTK_REAL      *origin            /* vdim */,
           const CCTK_REAL      *extent            /* hdim */,
           const CCTK_INT       *downsample        /* hdim */,
           CCTK_INT              table_handle,
           CCTK_FPOINTER         conversion_fn,
           CCTK_INT             *hsize_local,      /* hdim */
           CCTK_INT             *hsize_global,     /* hdim */
           CCTK_INT             *hoffset_global    /* hdim */);
```

**Function arguments:**

- `CCTK_POINTER_TO_CONST GH`
  The reference to the CCTK grid hierarchy.

  In a C implementation, this should be a pointer of type `const cGH *`.

- `CCTK_INT vindex`
  In order to compute a hyperslab mapping, a CCTK grid variable must be given by this argument which will be used as a template in the following hyperslab get/put operation to denote the input arrays' domain decomposition (dimensionality and distribution over processors). The domain decomposition of all input CCTK variables given by the `vindex, vindices` arguments in subsequent calls to `Hyperslab_GetXXX()/Hyperslab_PutXXX()` must match the one of the template variable `vindex`.

- `CCTK_INT hdim`
  The dimension of the hyperslab to get/put ($0 <$ `hdim` $<=$ `vdim`).

- `const CCTK_INT *direction`
  `const CCTK_INT *origin`
  `const CCTK_INT *extent`
  `const CCTK_INT *downsample`
  `---------------------------`
  `const CCTK_CHAR *coord_system_name`
  `const CCTK_INT *direction`
  `const CCTK_REAL *origin`
  `const CCTK_REAL *extent`
  `const CCTK_INT *downsample`
  Arguments describing the actual mapping of the hyperslab to get/put.

  The hyperslab location is identified by its origin (lower left corner), the direction vectors starting from the origin and spanning the hyperslab in the $N$-dimensional space, and its extents (size of the hyperslab in each direction).

  There are `hdim` direction vectors (one for each hyperslab axis) with `vdim` elements. The direction vectors are given in grid index points and must be linearly independent. The `direction` argument must be passed as an array `direction[vdim_index + hdim_index*vdim]` (`vdim` is the fastest changing dimension).

  The origin and extent can be given in either physical coordinates or grid points – for the first case a coordinate system needs to be given to do the mapping onto the underlying grid. For the second case, integer extents can be given as negative values meaning that the hyperslab mapping should be defined with the maximum possible extents (ie. the size of the underlying grid).

The downsampling parameter denotes the downsampling factors for the hyperslab to be extracted/distributed. They are given in terms of grid points in each hyperslab direction. The downsampling parameter is optional – if NULL is passed here, no downsampling will be applied.

- `CCTK_INT table_handle`
  A key/value table can be passed in via its handle to provide additional information to the hyperslab get/put routines about the hyperslab mapping. For example, there could be a tolerance parameter for hyperslabs which are not rectangular to the underlying grid. For grid points which offset from the direction vectors, the tolerance would then specify a (plus/minus) offset for the directions saying which points should still be included in the hyperslab space.

  Another example could be whether to do interpolation between grid points or not.

  Passing a table handle is optional, an invalid (negative) table handle denotes no additional table information.

- `CCTK_FPOINTER conversion_fn`
  Reference to a user-defined datatype conversion function.

  Users can request a type conversion between input and output data during a hyperslab get/put operation. A hyperslab API implementation may provide a set of predefined data type conversion routines for this purpose. In addition to this feature, users can also provide their own data type conversion function and pass a reference to it in the conversion_fn argument.

  For a C implementation, such a user-supplied conversion function should be declared according to the following typedef:

```
typedef CCTK_INT (*t_hslabConversionFn) (CCTK_INT nelems,
                                         CCTK_INT src_stride,
                                         CCTK_INT dst_stride,
                                         CCTK_INT src_type,
                                         CCTK_INT dst_type,
                                         CCTK_POINTER_TO_CONST src,
                                         CCTK_POINTER dst);
```

  A data type conversion function gets passed the number of elements to convert (`nelems`), the strides between adjacent elements in the source and destination arrays (`src_stride, dst_stride`), the source and destination CCTK datatypes (`src_type, dst_type`), a pointer to the data to convert (`src`), and a pointer to the conversion target buffer (`dst`). The routine should return the number of elements converted (`nelems`) for success.

  If a user-supplied function is given (`conversion_fn` is not NULL), subsequent hyperslab get/put calls will use for data type conversions. Otherwise the hyperslab get/put calls should fall back to using an appropriate predefined data conversion function (if any exists).

- `CCTK_INT *hsize`
  `CCTK_INT *hsize_local`
  Reference to a size array with `hdim` elements to be set by the `Hyperslab_XXXMappingByXXX()` routines.

  The resulting size of the hyperslab to be extracted is set according to the hyperslab extents and downsampling parameters chosen. With this information, one can compute the overall size of the hyperslab, allocate memory for it and pass it as a user-provided hyperslab data buffer into subsequent calls to `Hyperslab_GetXXX()`.

- `CCTK_INT *hsize_global`
  Reference to a size array with `hdim` elements to be set by the `Hyperslab_LocalMappingBy*()` routine.

  This array holds the sizes of the corresponding global hyperslab. It is set according to the local hyperslab extents and downsampling parameters chosen and locates the local hyperslab in the global grid.

  A value of NULL can be passed for `hsize_global` if no information about the global hyperslab size is needed.

- `CCTK_INT *hoffset_global`
  Reference to an offset array with `hdim` elements to be set by the `Hyperslab_LocalMappingBy*()` routine.

  This array holds the offsets of the local hyperslab into the corresponding global hyperslab. It is set according to the local hyperslab extents and downsampling parameters chosen and locates the local hyperslab in the global grid.

  A value of NULL can be passed for `hoffset_global` if no information about a hyperslab offsets is needed.

**Return codes (according to the Cactus Coding Conventions):**

- 0 for success
- negative for some error condition (to be defined by an actual implementation of the `Hyperslab_*MappingBy*()` routines)

## 2.2 Extracting/distributing a hyperslab

Each set of hyperslab get/put routines has two functions: one which operates on a single hyperslab, and another which gets/puts hyperslabs for a list of variables. Depending on the actual hyperslab implementation it might be more efficient to operate on a list of grid variables using `Hyperslab_GetList()/Hyperslab_PutList()` rather than doing sequential calls to `Hyperslab_Get()/Hyperslab_Put()` with individual grid variables.

```
CCTK_INT Hyperslab_Get (CCTK_POINTER_TO_CONST GH,
                        CCTK_INT       mapping_handle,
                        CCTK_INT       proc,
                        const CCTK_INT vindex,
                        const CCTK_INT timelevel,
                        const CCTK_INT hdatatype,
                        void          *hdata);

CCTK_INT Hyperslab_GetList (CCTK_POINTER_TO_CONST GH,
                            CCTK_INT        mapping_handle,
                            CCTK_INT        num_arrays,
                            const CCTK_INT *procs      /* num_arrays */,
                            const CCTK_INT *vindices   /* num_arrays */,
                            const CCTK_INT *timelevels /* num_arrays */,
                            const CCTK_INT *hdatatypes /* num_arrays */,
                            void *const    *hdata      /* num_arrays */,
                            CCTK_INT       *retvals    /* num_arrays */);

CCTK_INT Hyperslab_Put (CCTK_POINTER_TO_CONST GH,
                        CCTK_INT              mapping_handle,
                        CCTK_INT              proc,
                        CCTK_INT              vindex,
                        CCTK_INT              timelevel,
                        CCTK_INT              hdatatype,
                        CCTK_POINTER_TO_CONST hdata);

CCTK_INT Hyperslab_PutList (CCTK_POINTER_TO_CONST       GH,
                            CCTK_INT                    mapping_handle,
                            CCTK_INT                    num_arrays,
                            const CCTK_INT             *procs      /* num_arrays */,
                            const CCTK_INT             *vindices   /* num_arrays */,
                            const CCTK_INT             *timelevels /* num_arrays */,
                            const CCTK_INT             *hdatatypes /* num_arrays */,
                            const CCTK_POINTER_TO_CONST hdata      /* num_arrays */,
                            CCTK_INT                   *retvals    /* num_arrays */);
```

**Function arguments:**

- `CCTK_POINTER_TO_CONST GH`
  The reference to the CCTK grid hierarchy.

  In a C implementation, this should be a pointer of type `const cGH *`.

- `CCTK_INT mapping_handle`
  The handle for the hyperslab mapping as returned by a previous call to one of the hyperslab mapping routines.

- `CCTK_INT num_arrays`
  The total number of input arrays to get/put a hyperslab from/to.

  This must be a positive integer and match the number of array elements in the arguments following.

- `CCTK_INT proc`
  `const CCTK_INT *procs`
  The (list of) processor(s) which will receive/provide the hyperslab data.

  For `Hyperslab_GetXXX()`, there may be either exactly one processor providing the hyperslab data (in this case, its processor ID must be given in `proc`), or all all processors will get the extracted hyperslab data (an invalid (i.e., negative) processor ID should be given as `proc`, or `procs` is passed as a NULL pointer). For `Hyperslab_PutXXX()`, there may only be one processor providing the hyperslab data to be distributed to all others.

- `CCTK_INT vindex`
  `const CCTK_INT *vindices`
  The (list of) CCTK variable(s) to get/put a hyperslab from/to.

  The grid variables are given by their CCTK indices; their domain decomposition must match the template variable as given in a previous hyperslab mapping routine call.

- `CCTK_INT timelevel`
  `const CCTK_INT *timelevels`
  The (list of) timelevel(s) for the grid variable(s) to get/put a hyperslab from/to.

  Each element in the `timelevels` array matches its entry in the `vindices` array argument. If `timelevels` is passed as a NULL pointer then all timelevels for the list operation will default to 0 (denoting the current timelevel).

- `CCTK_INT hdatatype`
  `const CCTK_INT *hdatatypes`
  The (list of) CCTK data type(s) of the hyperslab data.

  The hyperslab data to be extracted/distributed may be given in a data type which is different to its corresponding grid variable. For this case, the requested hyperslab data type must be specified explicitely. The hyperslab routines will then do the neccessary data type conversions either by using a user-supplied data type conversion function (as specified in the `conversion_fn` argument to the hyperslab mapping routines), or by choosing a built-in predefined data type conversion function. convert the input array datatypes to some output array datatype.

  A negative value for `hdatatype` or type or passing a NULL pointer for the `hdatatypes` argument indicates that both the grid variable and its corresponding hyperslab have the same CCTK data type so that no type conversion is necessary.

- `(const) void *hdata`
  `(const) void *const *hdata`
  The (list of) user-supplied buffer(s) to store the extracted hyperslabs for each input variable (for a get operation) or to read the hyperslab data from (for a put operation).

  This argument is only evaluated on processors which are part of the hyperslab mapping.

- `CCTK_INT *retvals`
  User-provided array to store the status of each individual get/put operation in a `Hyperslab_XXXList()` call.

  Each element in the `retvals` array will contain the status of the corresponding hyperslab operation on grid variable *i*. If `retvals` is passed as a NULL pointer then no status codes for individual hyperslab operations will be passed back to the caller.

**Return Codes for these routines (according to the Cactus Code conventions:**

- 0 for success
- negative for some error condition (to be defined by an actual implementation of the `Hyperslab_GetXXX()/Hyperslab_Put` routines)

# 3 Implementation Details

The current version of thorn `PUGHSlab` implements only parts of the CCTK hyperslab API as described in section 2:

1. the definition of local/global hyperslabs based on grid indicices
   Currently, the only additional hyperslab mapping information which can be passed through a key/value table is a `CCKT_INT` option with the key `with_ghostzones`. If the value of this key is non-zero `PUGHSlab` will not strip outer boundary ghostzones for periodic boundary conditions as implemented by `PUGH`.

   `PUGHSlab` provides a set of predefined built-in functions for the following classes of data type conversions:

   - any `CCTK_INT` data type into any other `CCTK_INT`
   - any `CCTK_REAL` data type into any other `CCTK_REAL`
   - any `CCTK_COMPLEX` data type into any other `CCTK_COMPLEX`

2. local/global hyperslab extractions
   Global hyperslab get requests will strip off all processor-boundary ghostzones from the returned hyperslab data.

   Local hyperslabs will always include processor-boundary ghostzones. The `hsize_local, hoffset_local` information returned by the hyperslab mapping routines should be used to locate the locate hyperslab within the global grid (e.g. during a recombination of several local hyperslabs into a single global one).