DOMAIN DECOMPOSITION IN CARPET: REGRIDDING AND DETERMINATION OF THE COMMUNICATION SCHEDULE

ERIK SCHNETTER[(1,2)]

ABSTRACT. This text describes the algorithms that Carpet [**?**, **?**] uses for regridding, domain decomposition, and setting up the communication schedule. It is intended for readers interested more details than a casual user would need. This text explains the concepts that Carpet uses internally to describe the grid hierarchy and to ensure its consistency.

1. INTRODUCTION

Setting up a grid hierarch ("regridding") is in Carpet handled by three different entities: *Carpet* itself decides the extent of the domain, the type of outer boundary conditions, and distributes the domain onto processors; a *regridding thorn* is responsible for deciding the shape of the grid hierarchy, and *CarpetLib* handles the details and actually manages the data. (Technically speaking, it is the regridding thorn which has to do the domain decomposition; however, it can simply call a convenient helper routine in Carpet for this task.)

This separation leaves the decision on the shape of the grid hiearchy to a thorn which can be replaced or augmented as necessary. All handling of data happens in CarpetLib, which is thus the only entity which needs to be optimised for speed. Finally, Carpet retains the overview over the regridding process.

In the following we assume that there is a single patch (block) which contains a mesh refinement hierarchy. If there are multiple patches, then each patch is conceptually handled independently. Certain conditions may have to be satisfied if a refined mesh touches an inter-patch boundary, but these are not discussed here.

We assume that the domain has D spatial dimensions, usually $D = 3$.

2. DOMAIN DESCRIPTION

We assume that boundary location and boundary discretisation are set up via *CoordBase*. This is necessary since other methods do not allow specifying sufficient details to handle e.g. refined regions intersecting mesh refinement boundaries. Below we give an overview over the information that needs to be specified to describe a boundary. See the CoordBase thorn guide for details.

Date: March 9, 2008.

Email: mailto:schnetter@cct.lsu.edu.
Web: http://www.cct.lsu.edu/.
(1) Center for Computation & Technology, Louisiana State University, Baton Rouge, LA, USA.
(2) Department of Physics and Astronomy, Louisiana State University, Baton Rouge LA, USA.

The extent of the overall simulation domain is described in terms or real-valued coordinates. The domain is cuboidal, i.e., it can be described in terms of two vectors x^i_{min} and x^i_{max}.

The type of boundary conditions for each of the $2D$ faces is described by three quantities:

- the total number of boundary points,
- how many of these points are outside of the domain boundary,
- whether the boundary is staggered with respect to the domain boundary, or whether one of the boundary points is located on the domain boundary.

■ [TODO: Show the relevant figures from the CoordBase thorn guide.]

The extent of the domain $L^i := x^i_{\text{max}} - x^i_{\text{min}}$ must be commensurate with the coarse grid spacing h^i. This means that L^i must be a multiple of h^i, modulo the fact that the boundary may be staggered.

An *outer boundary condition* can either be a *physical* boundary condition, such as e.g. a Direchlet condition, or a *symmetry* boundary condition, such as e.g. a reflection symmetry. This distinction is irrelevant for Carpet. Both kinds of outer boundary conditions are applied by thorns which are scheduled in the appropriate way.

The main distinction between an *outer boundary point* and an *interior point* from Carpet's point of view is that an outer boundary point is not evolved in time. Instead, the value of boundary points must be completely determined by the value of interior points. (This is clearly the case for Direchlet or symmetry boundary conditions.) The reason for this distinction is that Carpet cannot fill outer boundary points on refined grids via interpolation, because this requires off-centred interpolation stencils which are not implemented at the moment. Therefore, Carpet does not fill any outer boundary points through interpolation.

■ [TODO: We should introduce mesh refinement and parallelisation before this paragraph.] If the refined region abuts the outer boundary, then outer boundary points can also be refinement boundary points. Carpet does not fill these outer boundary points through synchronisation (prolongation) either. This requires that the outer boundary condition is applied after every synchronisation. This is usally automatically the case when using the Cactus boundary condition selection mechanism, i.e., when the group `ApplyBCs` is scheduled. Synchronising outer boundary points would be possible, but this is not done so that refinement boundaries and inter-processor boundaries are treated in the same way.

3. Regridding

A regridding thorn, such as e.g. `CarpetRegrid` or `CarpetRegrid2`, sets up the *grid hierarchy*. The grid hierarchy consists of several *refinement levels*, and each refinement level consists of several *refined regions*. A refined region is a cuboid set of grid points which is assigned to a particular processor. (The “refined regions” which are specified e.g. in a parameter file are broken up during domain is decomposition into a set of regions, so that each region is assigned to exactly one processor.) There can be zero or multiple regions per processor, and differnet processors may own different numbers of regions. Each face of a region is either an outer or an internal boundary. Refined regions are described by the data structure `region_t`, which is declared in `CarpetLib/src/region.hh`.

The refined regions on one level may not overlap. (If they do not touch each other or the outer boundary, then they usually even have to keep a certain minimum distance, as described below.)

The "semantics" (the result obtained in a simulation) of a grid hierarchy is independent of the number of refined regions, or of the processors which are assigned to them. The only relevant quantity is the set of refined grid points, i.e., the conjunction of all refined regions. When running on more processors, then regions will be split up so that there are more and smaller regions.

The assignment of regions to processors is handled during regridding because it affects performance. When there are only few processors available, then it may be more efficient to use fewer regions. Combining regions may require some fill-in. ■ [TODO: Show figure.] No current regridding thorn in Carpet offers this at the moment; currently, the set of refined points is independent of the number of processors.

Each face of a refined region is either an interior or an outer boundary face. Carpet adds no ghost or buffer zones to outer boundary faces, and marks this face as outer boundary when calling thorns. An outer boundary face must extend up to the outermost outer boundary point, since otherwise thorns will apply the outer boundary conditions incorrectly. The regridding thorn has to ensure this property. Faces which are not outer boundary faces must be sufficiently far away from outer boundaries, so that any ghost or buffer zones which are added to that face do not intersect the outer boundary. ■ [TODO: Show figure.] The regridding thorn also has to ensure this property.

Each face which is not an outer boundary face is either an inter-processor or a refinement boundary face. Inter-processor faces have no buffer zones, and the ghost zones can be filled in completely from other refined regions on the same level. Refinement boundary faces do have buffer zones, and at least some of the ghost zones must be filled via prolongation from the next coarser grid. It is possible to have faces which are partly an inter-processor boundary and partly a refinement boundary. These are counted as and handled as refinement boundaries, although some of the ghost zones may still be filled via synchronisation. (Any grid points which can be filled via synchronisation are always also filled via synchronisation.) This is explained in more detail below.

As described below, ghost zones are added at the outside of regions by Carpet. Buffer zones need to be taken into account by the regridding thorn, most likely by extending the regined regions appropriately. (In terms of previous terminology: buffer zones are always "inner" buffer zones; "outer" buffer zones do not exist any more. However, if they are added by the regridding thorn, they do not need to be taken into account when specifying the refinement hierarchy in a parameter file – this depends on the regridding thorn. ■ [TODO: CarpetRegrid2 does this, CarpetRegrid does this not.]) Carpet assumes buffer zones at all faces which are marked as refinement boundaries.

It is the responsibility of the regridding thorn to mark outer boundaries and refinement boundaries appropriately. If the outer boundary mark is chosen wrongly, then the simulation is inconsistent, since the outer boundary condition may be applied at a the wrong location, or the outer boundary may be filled by interpolation which is less accurate. Depending on the number of boundary points and stencil sizes, this may or may not be detected later by self-consistency checks.

4. Zoning

This section defines the algorithm which Carpet uses to define which grid points are defined by what action. This algorithm is codified in `dh.cc`. Since the code in `dh.cc` is tested, it should be assumed to be correct where it differs with this description. This algorithm is applied to every refinement level.

Grid arrays are handled in the same way as grid functions, except that there are no refined regions. This may seem like overkill at first, but in fact it greatly simplifies the implementation since grid arrays have (almost) only a subset of the capabilities of grid functions.

4.1. Domain.

Let dom be the simulation domain. Let $domact$ be the set of grid points which are evolved in time, and which is required to be cuboidal and not empty.[1] (An empty region is also counted as cuboidal.)

Each face has a layer of outer boundary points. Let $domob$ be the set of these layers. We require that all values in $domob$ can be calculated from the values in $domact$. Let $domext := domact \cup domob$. It follows that $domext$ is cuboidal and not empty.

■ [TODO: Show figure.]

The following properties hold for dom:

- $domact$ is cuboidal and not empty
- $domob$ is a possibly empty layer around $domact$
- $domob$ has a width of `nboundaryzones` as obtained from `GetBoundarySpecification`
- $domact \cap domob = \emptyset$
- $domext = domact \cup domob$ is cuboidal and not empty
- $domext$ has the size `cctk_gsh`

For completeness, one can define $dombnd = dombuf = \emptyset$, and $domint = domown = domact$. Then the same relations hold for dom as for reg below.

4.2. Refined regions.

Let reg be a refined region. Let $regint$ be its interior, which is required to be cuboidal and not empty. $regint$ includes buffer zones and outer boundary points. We require that $regint \subseteq domext$.

There may be other refined regions reg' on the same level. We require that $\forall_{reg' \neq reg} : regint \cap regint' = \emptyset$.

Each face of $regint$ which is not an outer boundary face has a layer of `cctk_nghostzones` ghost zones added to it. (These are what Cactus calls "ghost zones", but note that not all ghost zones are filled via synchronisation.) Let $regghost$ be the set of these layers. We require that $regghost \subseteq domact$. Let $regext := regint \cup regghost$ be the interior with the ghost zones added. It follows that $regext$ is cuboidal and not empty. We require that $regext \subseteq domext$.

■ [TODO: Show figure.]

On each face of $regext$, the outermost layer of `nboundaryzones` points are not communicated. Let $regcomm$ be the set of communicated points. The $regcomm \subseteq regext$, we require it to be not empty, and we require that $recomm \subseteq domact$.

Let $regob := regext - regcomm$ be the set of outer boundary points. It follows that $regob \subseteq domob$.

[1]While grid functions cannot be empty, *domact* for grid arrays can be empty.

The owned region $regown$ is the interior region, extended up to the boundary, i.e., $regint$ with a layer of `nboundaryzones` points added to it at all outer boundary faces. Thus $regown$ is cuboidal, and we require it to be not empty. It is also $regown \subseteq domact$, and $regown \subseteq regext$. It is also $\forall_{reg' \neq reg} : regown \cap regown = \emptyset$.

Let $regbnd := regcomm - regown$. Then $regbnd \subseteq domact$. ■ [TODO: Does this follow, or is this a requirement?] These points are filled via synchronisation ■ [TODO: Is this so? Is this always via inter-processor communiocation, or also via prolongation?], and they cannot be too close to the outer boundary.

Let $regbuf$ be the set of grid points which have a distance less than `buffer_width` from the boundary of the conjunction of all active grid points. Then $regbuf \subseteq regown$. Let $regact := regown - regbuf$. Then also $regact \subseteq regown$. In general, $regact$ is not cuboidal. $regact$ can be empty. ■ [TODO: This may be controversial.]

■ [TODO: Show figure.]

Let $regsync := regbnd \cap allown$ be the boundary points which can be filled via synchronisation. Note that $regbnd \cap regown = \emptyset$ by construction, which means that a region cannot synchronise from itself. Note that outer boundary points are synchronised for backward compatibility.

Let $regref := (regbnd - regsync) \cup regbuf$ be the set of all points which need to be filled via prolongation. Note that $regref \cap regsync = \emptyset$, which means that no point is both synchronised and prolongated. Note also that $regref \cap regob = \emptyset$, which means that no outer boundary point is prolongated.

■ [TODO: Show figure.]

The following properties hold for reg:

- $regint \subseteq domext$ is cuboidal and not empty
- $regghost$ is a layer on the outside of $regint$ on those faces which are not marked "outer boundary"
- $regext = regint \cup regghost \subseteq domext$ is cuboidal and not empty
- $recgomm \subseteq domact$
- $regob \subseteq domob$
- $regown \subseteq domact$ is cuboidal and not empty
- $\forall_{reg' \neq reg} : regown \cup regown' = \emptyset$
- $regbnd$ is a layer around $regown$ on those faces which are not marked "outer boundary"
- $regbuf$ is a layer around $regact$ on those faces which are marked "refinement boundary"
- $regsync$ are those boundary points which can be filled via synchronisation
- $regref$ are those boundary or buffer points which are filled via prolongation

$regint$ is not important for Carpet's working, but only for Cactus. Since Cactus has no notion of outer boundaries, we introduce $regint$, which corresponds to $regown$, but includes outer boundary points.

5. Algorithmic details

It is straightforward to extend or shrink a cuboidal region C by a layer of grid points with a given width. It is also straightforward to extend an arbitrary region R, which is internally represented as a set of non-overlapping cuboidal regions C^i. One sets $\operatorname{ext}[R] := \bigcup \operatorname{ext}[C^i]$. The individual extended cuboidal regions $\operatorname{ext}[C^i]$

will overlap in general, but this overlap is handled correctly by the operator $\bigcup$. However, there is no straightforward way to shrink an arbitrary region R. One possibility is to extend the complement of R instead.

The layer of buffer zones $allbuf$ can be calculated in the following way. Let $allown := \bigcup regown$ as above be the set of all refined grid points. Then $domact - allown$ is the set of all not-refined grid points in the domain. In order to handle the outer boundary of the domain correctly, we define $domlarge$ to be $domact$, sufficiently enlarged in each direction, i.e., enlarged at least by the number of buffer zones. Let $notown := domlarge - allown$ be the complement of $allown$. Let $notact$ be $notown$, enlarged by the number of buffer zones. Then $allbuf := allown - notact$.

APPENDIX A. TERMINOLOGY

■ [TODO: To be filled in]

Block:: See map.
Buffer zone:: ???
Component:: ???
Ghost zone:: ???
Grid hierarchy:: ???
Inter-processor boundary:: ???
Map:: ???
Outer boundary:: ???
Patch:: See map.
Physical boundary:: ???
Refinement boundary:: ???
Refinement level:: ???
Region:: ???
Regridding:: ???
Symmetry boundary:: ???