Octopus: Meshes and Grids

Martin Lüders

Octopus Course 2023, MPSD Hamburg
Real-space methods
Real-space methods

Finite difference methods:
- Finite simulation volume (simulation box)
- Discretize space within this volume (mesh)
- Represent functions on these mesh points
- Represent derivatives through finite differences
- Boundary conditions
Simulation box

The simulation box:

- Some finite volume in which we solve the equations
- main function: is a point inside or outside?
- can have different shapes
- can be a combination of boxes (recursively)
Simulation box

Examples:
- Parallelepiped
- Sphere
- Cylinder
- Union and Intersection
- Minimum box (union of spheres)
Simulation box

Implementation:

- In folder src/boxes/
- Abstract class box_t
- Instance created by factory box_factory.F90
- ...

Octopus: Meshes and Grids

Martin Lüders
The mesh

a)

b)

c)

d)
Real-space mesh

The mesh:

- usually uniform (curvilinear meshes or double grids are possible)
- contained in 'simulation box'
- can be distributed over processes (domain decomposition)
- access via linear indices (local and global index)
- we need some 'extra points':
  - for boundary conditions
  - halo points (ghost points)
Boundary points

- Points in the simulation box
- Stencil for derivatives
- Derivatives close to the boundary: need extra points!
- Boundary points

- inner point
- boundary point
Parallel domain decomposition

- Distribute points over processors
- Derivatives close to the boundary: require communication (slow!!)
- Introduce halo (ghost points) on each processor
- Total number of points on a processor: local + boundary + ghost points
- Note: boundary and ghost points are only needed if you need to take a derivative!

- local point
  - boundary point
  - ghost point
1D mapping

- Octopus is designed to work in various spatial dimensions.
- Memory layout cannot depend on spatial dimensions.
- Map to 1-dimensional arrays.
- Cubic mapping or Hilbert curve mapping.
- Separate inner (local) from boundary (ghost) points.
- Usually hidden in low level routines.
Real-space mesh

Memory layout:

- **mesh sizes:**
  - \( \text{np} \) number of local ‘inner’ points
  - \( \text{np\_part} \) number of local ‘inner’ points + ’ghost’ points + boundary points
  - \( \text{np\_global} \) number of global ‘inner’ points
  - \( \text{np\_part\_global} \) number of global ‘inner’ points + boundary points

- **ordering:**
  - inner points first \([1:\text{np}]\)
  - ghost and boundary points: \([\text{np}+1:\text{np\_part}]\)

- mesh points: \( \text{mesh}\_x(1:\text{mesh}\_\text{np\_part}, 1:\text{space}\_\text{dim}) \)
Real-space mesh

In the typical case of zero boundary conditions $v(np+1:np\_part)$ is 0. The two parts are split according to the partitions. The result of this split are local vectors $v\_local$ on each process which consist of three parts:

- $v\_local(1:np\_local)$ local points.
- $v\_local(np\_local+1:np\_local+np\_ghost)$ ghost points.
- $v\_local(np\_local+np\_ghost+1:np\_local+np\_ghost+np\_bndry)$ boundary points.
Real-space mesh: in practice

- Low level routines take care of boundary/ghost points.
- Only need to set the values for inner points
- BUT: If you need derivatives of a mesh function, it must be allocated as contiguous memory 1:np_part
The grid
Real-space grid

The grid describes a number of things:

- the mesh (the actual points in space)
- the derivatives
- the stencil
- symmetries and the symmetrizer

```fortran
type, extends(mesh_t) :: grid_t
  ! Components are public by default
  type(derivatives_t) :: der
  type(stencil_t) :: stencil
  type(symmetries_t) :: symm
  type(symmetrizer_t) :: symmetrizer
end type grid_t
```
Stencils

The stencil describes:

- The points for the derivatives
- Specific stencils: routines to generate coefficients for derivatives
Derivatives

The derivatives object contains:
- gradient and laplacian operators (nl_operator_t)
- boundary conditions

The class provides:
- routines to apply the derivatives to mesh functions and batches
  - apply boundary conditions
  - perform ghost updates