A New Petsc Interface for Adaptive Meshing: Combining Usability and Computational Efficiency using P4est

Max Heldman Boston University
Johann Rudi Argonne National Laboratory
Emil Constatinescu Argonne National Laboratory

P4est: forest-of-octrees based mesh adaptivity

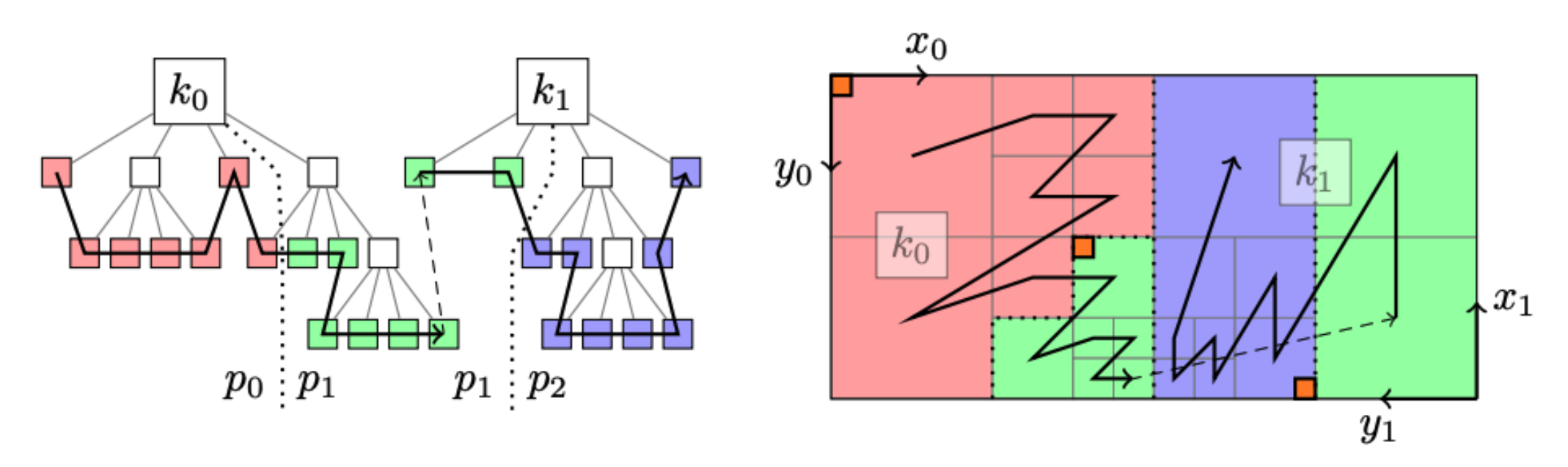


Image credit: [1]

Forest-of-octrees data structure

An *octree* is a tree-based representation of a mesh, geometrically interpreted as a hierarchy of cubes (or smooth images of cubes) in $\mathbb{R}^3$ subdivided into identical octants. Leaves of the octree are called elements or cells. A *forest-of-octrees* is a collection of octrees plus a specification of their relative orientations in space.

Advantages of octree-based refinement

- Tree structure allows for extremely lightweight storage.
- Efficient algorithms for, e.g., parallel load balancing and 2:1 balance enforcement

Challenges of octree-based refinement

Nonconforming refinement leads to *hanging nodes and faces*:

- Simple to handle for continuous tensor product finite elements
- Less natural for cell-centered finite volume methods; may require several different stencils

PETSc and the DM

A *PETSc DM* is an interface between a (parallel) discretization of a computational domain and (parallel) algebraic objects like vectors (`Vec` objects) and matrices (`Mat` objects). Common DM-compatible operations include

- Creating serial and parallel vectors and matrices (`DMCreateLocalVector`, `DMCreateGlobalVector`, `DMCreateMatrix`)..
- ..which come with mappings between process local and global orderings of indices, to be used for transferring ghost data (`DMLocalToGlobal`, `DMGlobalToLocal`)
- Viewing the discretized domain (the mesh) and mesh functions in a variety of different formats (`DMView`, `VecView`)

The algebraic objects obtained through discretization using the DM can then be inserted into PETSc solvers for linear (KSP: Krylov subspace iterative methods) and nonlinear (SNES: Scalable Nonlinear Equations Solvers) algebraic equations, or into time-stepping routines (TS).

Numerical PDE solvers with a new DM, DMBF: PETSc + P4est

Basic DMBF operations

DMBF provides a direct interface to the forest-of-octrees adaptive meshing library p4est [1] from PETSc, allowing access to the PETSc DM tools described above for meshes generated by `p4est` along with core functions of `p4est`:

- mesh refinement, coarsening, partitioning
- transferal of ghost data between processes
- iterations over mesh cells and faces

A Short Tutorial

Examples of `p4est` features interfaced by DMBF include mesh iterators:

```
DMBFIterateOverCells(DM,DM_BF_Cell_Fn,void*);
DMBFIterateOverFaces(DM,DM_BF_Cell_Fn,void*);
```

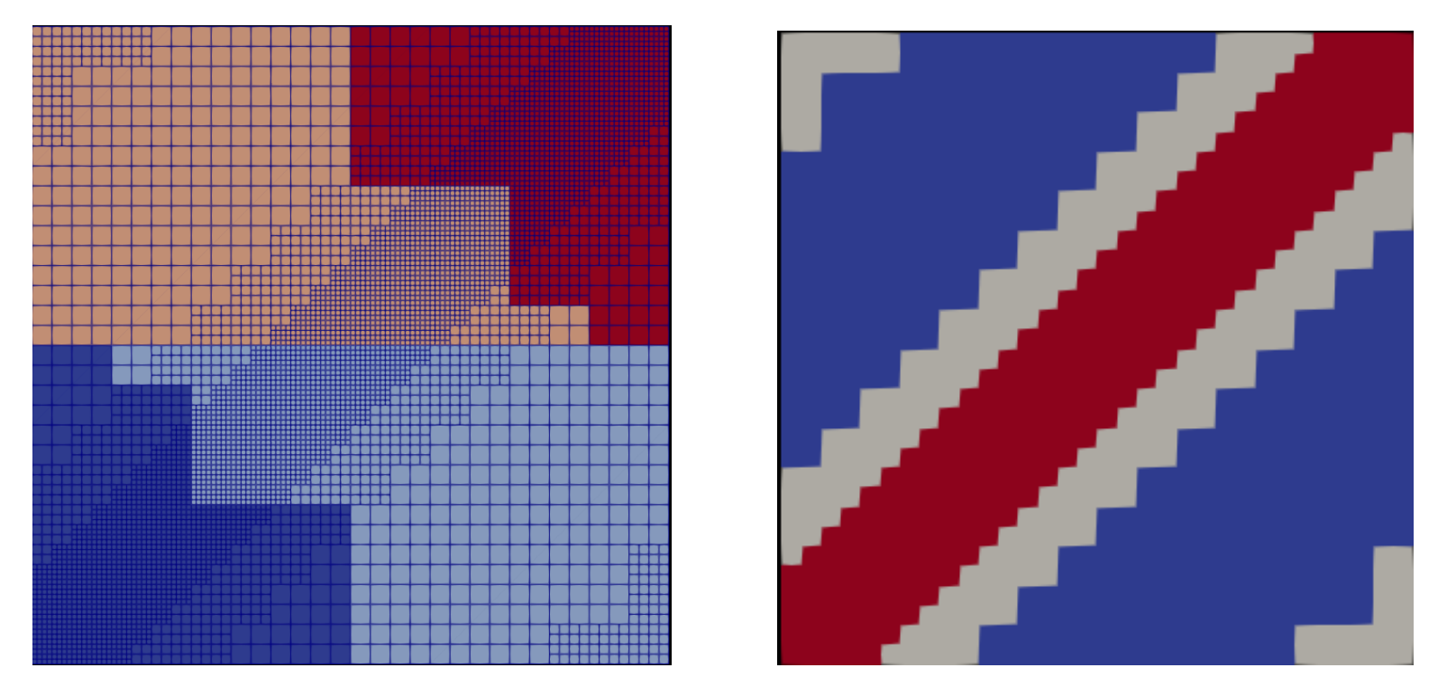
These interface to `p4est_iterate`, executing a user callback on each cell or face of the mesh. These iterators are usually a fundamental feature of numerical algorithms using `p4est`. The user callbacks `DM_BF_Cell_Fn` and `DM_BF_Cell_Fn` have `(DM_BF_Cell*,void*)` and `(DM_BF_Face*,void*)` as arguments, respectively. The `DM_BF_Face*` carries the `DM_BF_Cell*`(s) on each side of the face, and the `DM_BF_Cell*` carries information about the cell including coordinates of vertices, sidelengths, and additional data inserted by the user.

A typical workflow for applying a function over each face of the mesh:

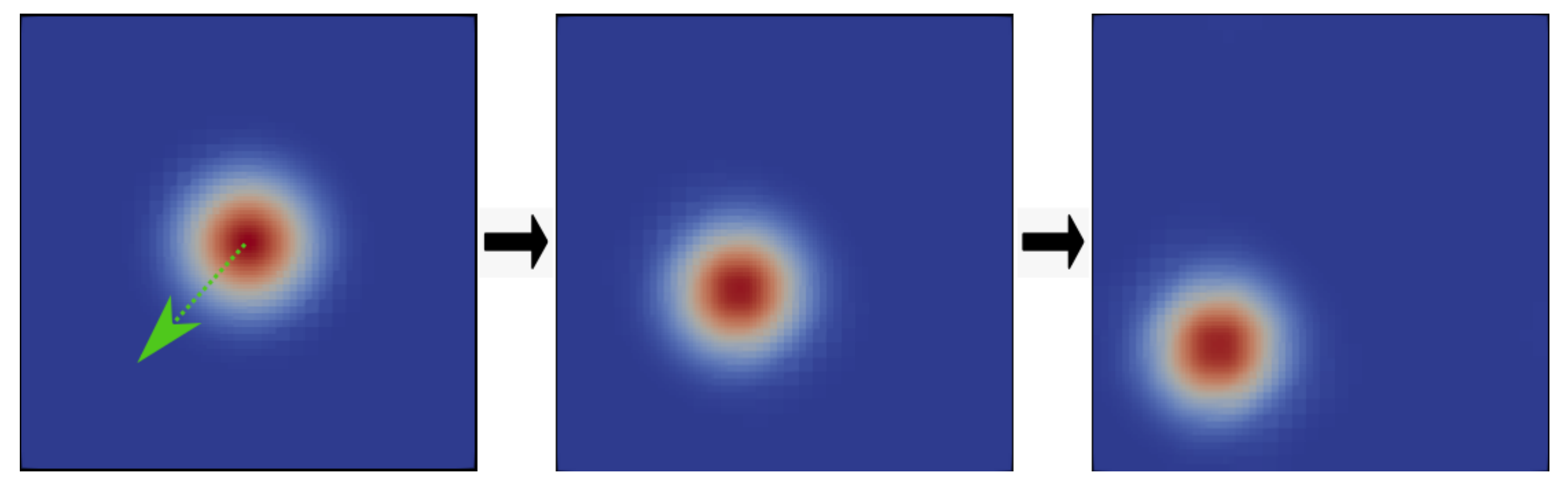
```
ierr = DMBFSetCellData(dm,&in,&out);CHKERRQ(ierr);
ierr = DMBFCommunicateGhostCells(dm);CHKERRQ(ierr);
ierr = DMBFIterateOverFaces(dm,face_fn,(void *)user);CHKERRQ(ierr);
ierr = DMBFGetCellData(dm,PETSC_NULL,&out);CHKERRQ(ierr);
```

- `DMBFSetCellData` attaches the values of the vectors `in` and `out` to the DMBF cells as read only data and read-write data, respectively
- Data for ghost quadrants is sent and received via `DMBFCommunicateGhostCells`
- `DMBFIterateOverFaces` calls `face_fn` on each face of the `p4est` mesh, operating on the read-only data and storing the result in the read-write data
- `DMBFGetCellData` replaces vector `out` with the modified read-write data in the cells.

Similarly, adaptive meshing is done by setting cell callbacks for projecting cell data from coarse to fine, restricting cell data from fine to coarse, and flagging cells for refinement or coarsening. The simple interfaces to `p4est` iterators for discretization and AMR provide a flexible environment accommodating many different problem types and discretizations.



Adaptive mesh colored by MPI rank (left) and refinement level (right) for the advection example



Solution to the advection problem at three different times. The problem was solved using PETSc's built-in multirate time-stepping. We used a simple flux, $\mathbf{a} = \begin{bmatrix} -1 & -1 \end{bmatrix}^\top$ and the initial condition $u_0 = \cos^6(2\pi r/r_0)$ if $r < r_0$ and 0 otherwise. The green arrow in the first image shows the direction of advection, while the black arrows in between images indicate the flow of time.

References

[1] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

Example: Advection with upwinding.

We solve the advection equation

$$\frac{\partial u}{\partial t}(x,t) = \nabla \cdot (\mathbf{a}(x)u(x,t)), \quad (x,t) \in [-1,1]^2 \times [0,.25],$$

with periodic boundary conditions on the spatial domain using an upwind MUSCL scheme. Initially defined for a uniform mesh on $\mathbb{R}$, fluxes are computed by extrapolating function values on cell faces:

$$U^-_{i+\frac{1}{2}} = U_i + \frac{\Delta x}{2}\text{minmod}_{\text{SI}}(r_i)\frac{U_{i+1}-U_i}{\Delta x}$$
$$U^+_{i+\frac{1}{2}} = U_{i+1} - \frac{\Delta x}{2}\text{minmod}_{\text{SI}}(r_{i+1})\frac{U_{i+2}-U_{i+1}}{\Delta x},$$

where r_i is the ratio of slopes

$$r_i = \frac{\frac{U_i - U_{i-1}}{\Delta x}}{\frac{U_{i+1}-U_i}{\Delta x}}$$

and

$$\text{minmod}_{\text{SI}}(r_i) = \max(0, \min(1, r_i)),$$

and then computing an upwind flux by

$$F_i = \frac{1}{\Delta x}\begin{cases} a_{i+\frac{1}{2}}U^-_{i+\frac{1}{2}} & \text{if } a_{i+\frac{1}{2}} > 0 \\ a_{i+\frac{1}{2}}U^+_{i+1} & \text{if } a_{i+1/2} < 0 \end{cases}.$$

To compute $U^+_{i+1/2}$ requires U_i, U_{i+1}, and U_{i+2}. Therefore, we set the block refinement level to 2 to match the stencil width.

Below are the possible arrangements of two cells sharing a face and all possible stencils for computing U^+ on the cell interfaces.

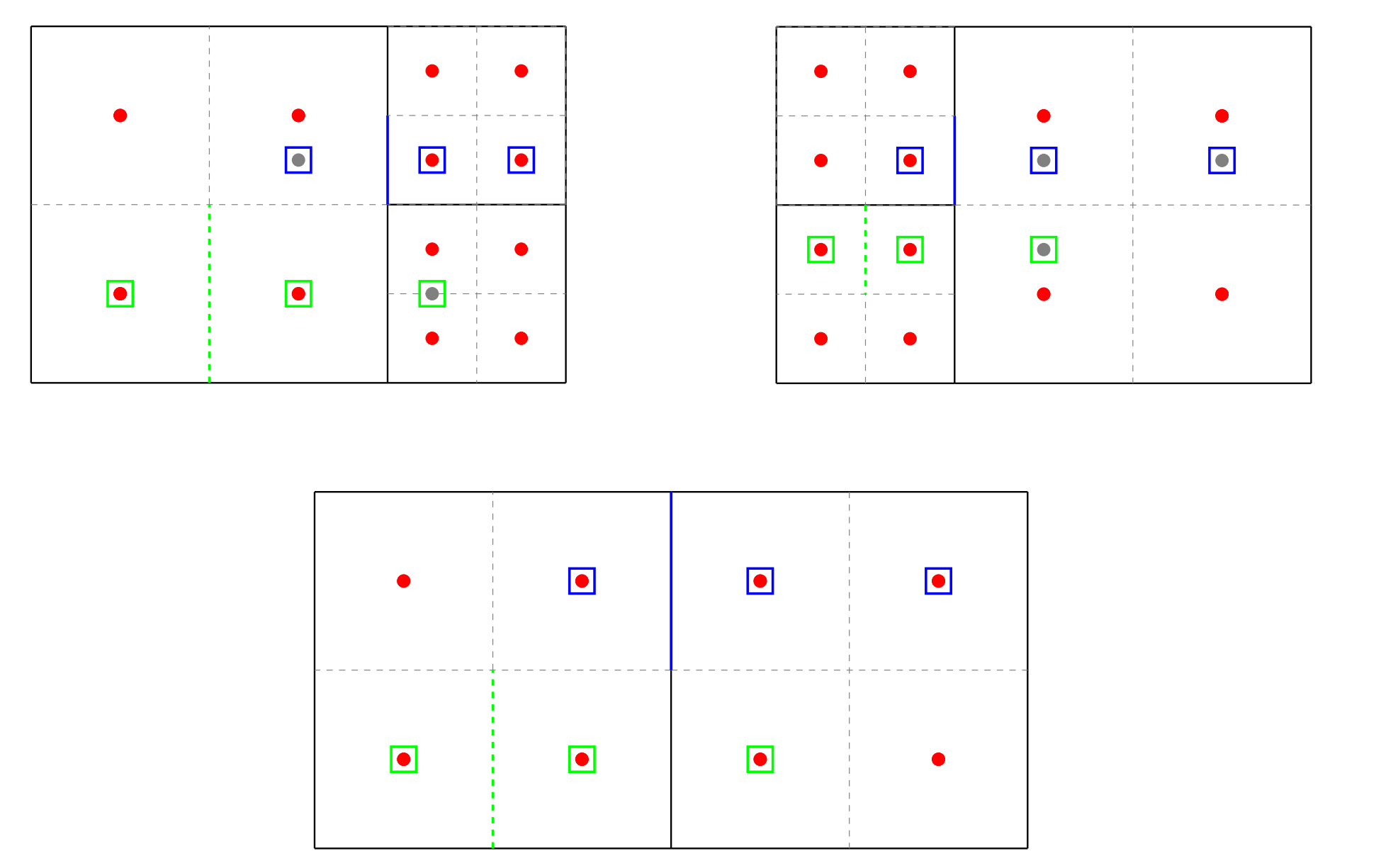


Image key:

- Outlined in black: cells from the underlying `p4est` mesh
- Additional cells created by block refinement are constructed by the gray dashed lines
- Red nodes at the centers of the cells: unknowns for the finite volume method
- Arrangements of colored rectangles: Upwind stencils for the finite volume scheme
  - Colored edges: edges over which the flux associated with the stencil of the same color is being computed
  - Red nodes at the centers of the rectangles represent unknowns involved in the stencil
  - Gray nodes represent ghost values involved in the stencil