Parallel I/O and Parallel Refinement

Chris Richardson
Garth Wells

DCSE project – NAG

lp:~cam-fenics/dolfin/phdf5
lp:~cam-fenics/dolfin/parallel-refine

File access in parallel

Motivation: using HPC systems needs I/O that is scalable.

- XMLSAXParser reader is very slow – every process reads whole file and parses part needed
- VTK output – every process outputs a separate file, for each timestep.
  This can make a lot of files.

49Mcell mesh

```
Nodes allocated:
================
sand-10-19 sand-10-18 sand-10-11 sand-10-10

numprocs=64, numnodes=4, ppn=16

Executing command:
==================
mpirun -tune -ppn 16 -np 64 python /home/cnr12/python/bigmesh.py

Process 0:
Process 0: Summary of timings                                | Average time  Total time  Reps
------------------------------------------------------------------------------------------------
Build mesh number mesh entities                              |   3.0994e-06  3.0994e-06     1
Compute local dual graph                                     |       2.3142      4.6284     2
Compute non-local dual graph                                 |       3.8669      7.7337     2
HDF5: read mesh                                              |      0.16474     0.32948     2
HDF5: reorder vertex values                                  |      0.08997     0.26991     3
HDF5: write mesh to file                                     |       6.0029      18.009     3
Init MPI                                                     |        1.576       1.576     1
PARALLEL 1a: Build distributed dual graph (calling ParMETIS) |       2.5287      2.5287     1
PARALLEL 1b: Compute graph partition (calling ParMETIS)      |        1.951       1.951     1
PARALLEL 2: Distribute mesh (cells and vertices)             |         1.74      5.2201     3
PARALLEL 3: Build mesh (from local mesh data)                |       10.994      32.983     3
Partition graph (calling SCOTCH)                             |       30.229      60.457     2
XML: readSAX                                                 |       92.945      92.945     1
compute connectivity 0 - 3                                   |      0.16687     0.50061     3
compute connectivity 2 - 3                                   |      0.21143     0.63429     3
compute connectivity 3 - 3                                   |        2.251       6.753     3
compute entities dim = 2                                     |       7.0417      21.125     3
```

HDF5 and XDMF

- HDF5 is a binary data format
- XDMF is an XML metadata format
- Internally, H5 files look like a filesystem
- H5 files allow parallel access using MPI-IO
- Visualisation software (paraview, visit etc.) can read XDMF/H5 in combination
- Can also store multiple datasets / time series

Example XDMF/HDF5

```
<?xml version="1.0"?>
<Xdmf Version="2.0" xmlns:xi="http://www.w3.org/2001/XInclude">
  <Domain>
      <Grid Name="f_0" GridType="Uniform">
        <Topology NumberOfElements="800" TopologyType="Triangle">
          <DataItem Format="HDF" Dimensions="800 3">new.h5:/Mesh/0/topology</DataItem>
        </Topology>
        <Geometry GeometryType="XY">
          <DataItem Format="HDF" Dimensions="527 2">new.h5:/Mesh/0/coordinates</DataItem>
        </Geometry>
        <Attribute Name="f" AttributeType="Scalar" Center="Cell">
          <DataItem Format="HDF" Dimensions="800 1">new.h5:/VisualisationVector/0</DataItem>
        </Attribute>
      </Grid>
      …....
```

HDF5 binary file – view with h5dump:

```
HDF5 "new.h5" {
GROUP "/" {
   GROUP "Mesh" {
      GROUP "0" {
         DATASET "coordinates" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 527, 2 ) / ( 527, 2 ) }
            DATA {
            (0,0): 0, 0,
            (1,0): 0.005, 0,
            (2,0): 0.01, 0,
            (3,0): 0.015, 0,
            (4,0): 0.02, 0,
            (5,0): 0.025, 0,
```

Implemented methods

- **Function visualisation**

  XDMFFile << Function

- **Read and write Mesh, MeshFunction**

  XDMFFile << Mesh XDMFFile >> Mesh

  XDMFFile << MeshFunction XDMFFile >> MeshFunction

  HDF5File.write(mesh, 'name') HDF5File.read(mesh, 'name')

  HDF5File.write(meshfunction, 'name')

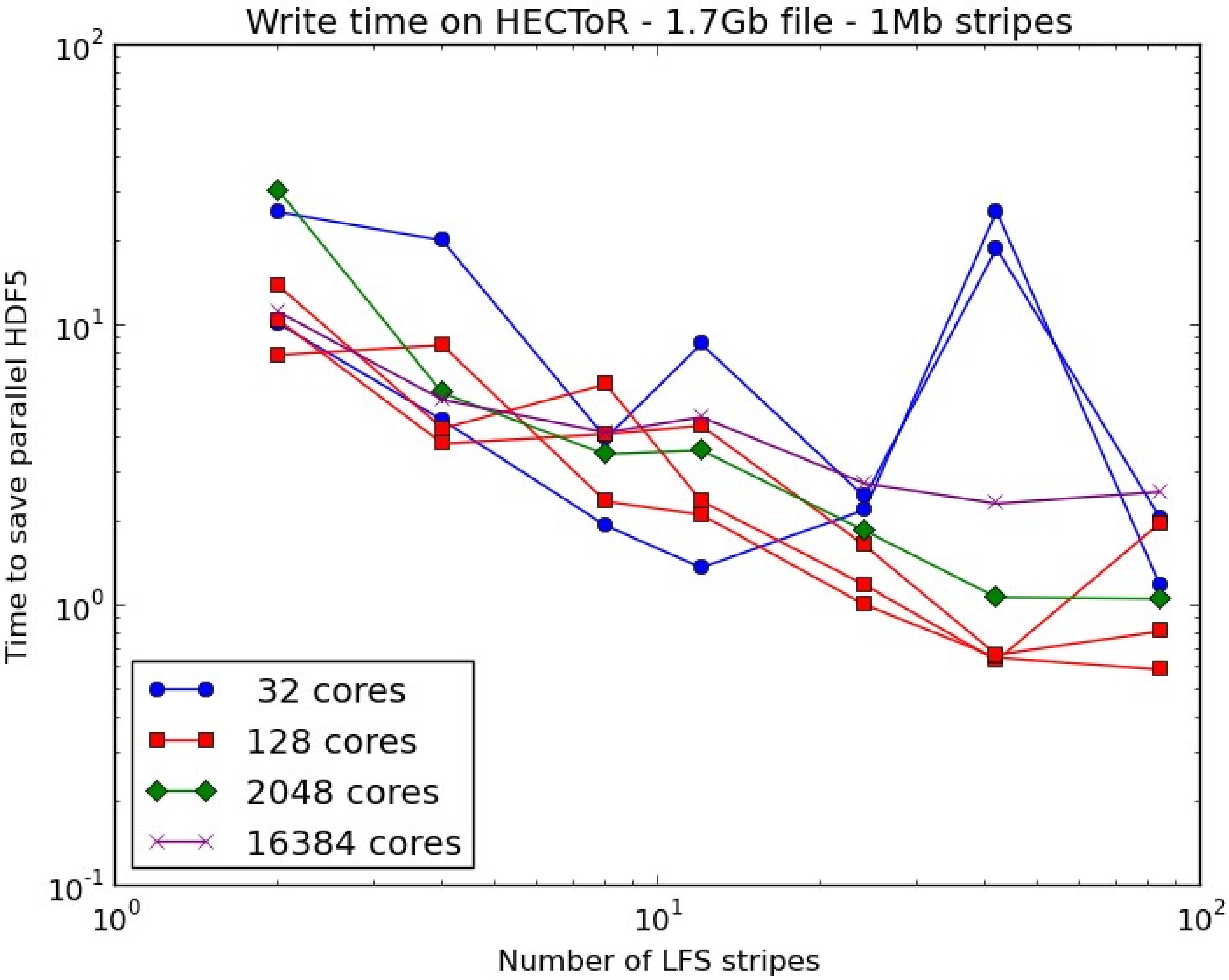
  HDF5File.read(meshfunction, 'name')

- **Read and write Vector**

  HDF5File.read(vector, 'name')

  HDF5File.write(vector, 'name')

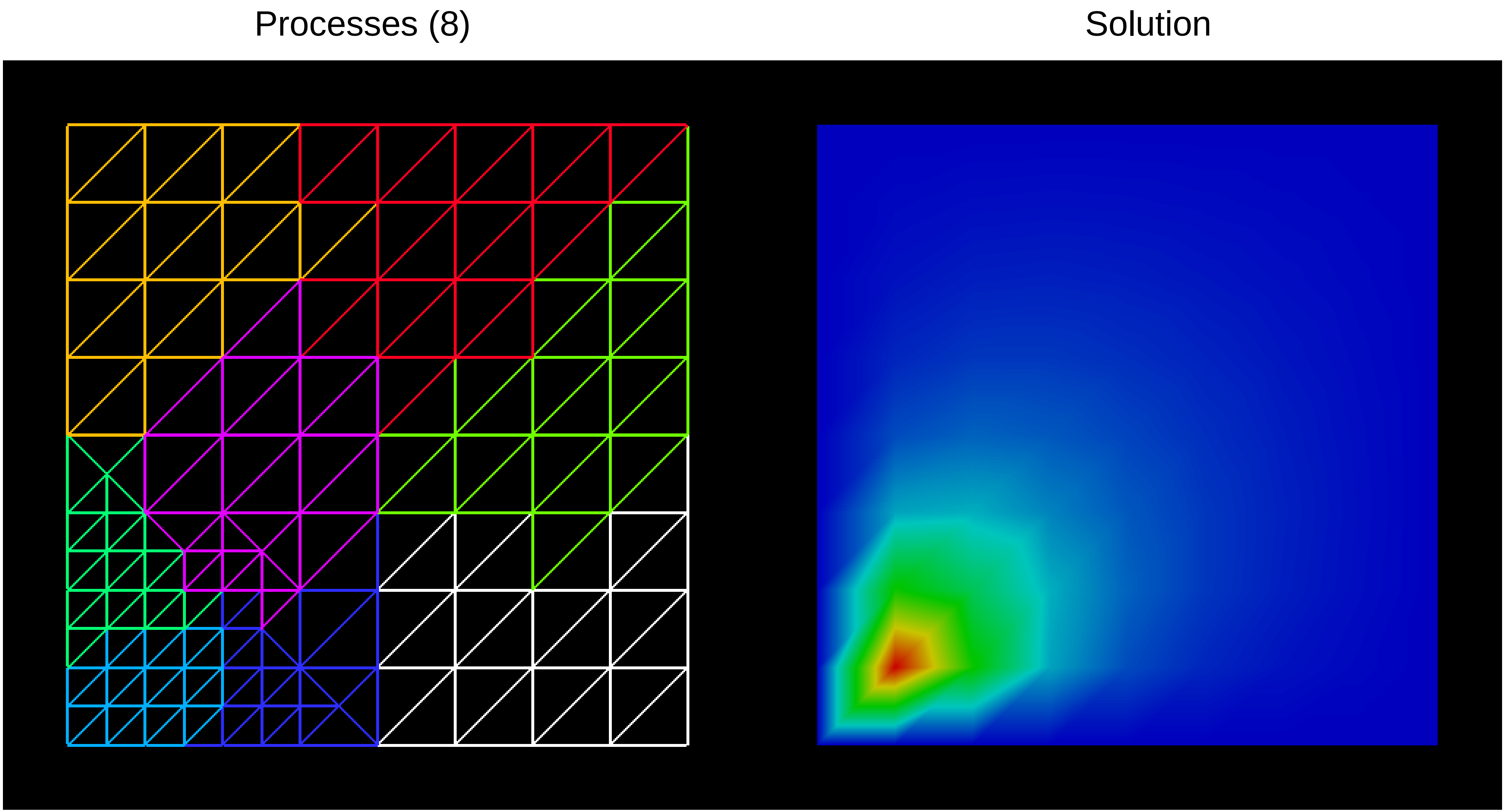
- Mostly already in dolfin trunk – try it out...


Write time on HECToR - 1.7Gb file - 1Mb stripes
Time to save parallel HDF5
Number of LFS stripes
10^2
10^1
10^0
10^{-1}
10^0
10^1
10^2
32 cores
128 cores
2048 cores
16384 cores

Refinement in parallel

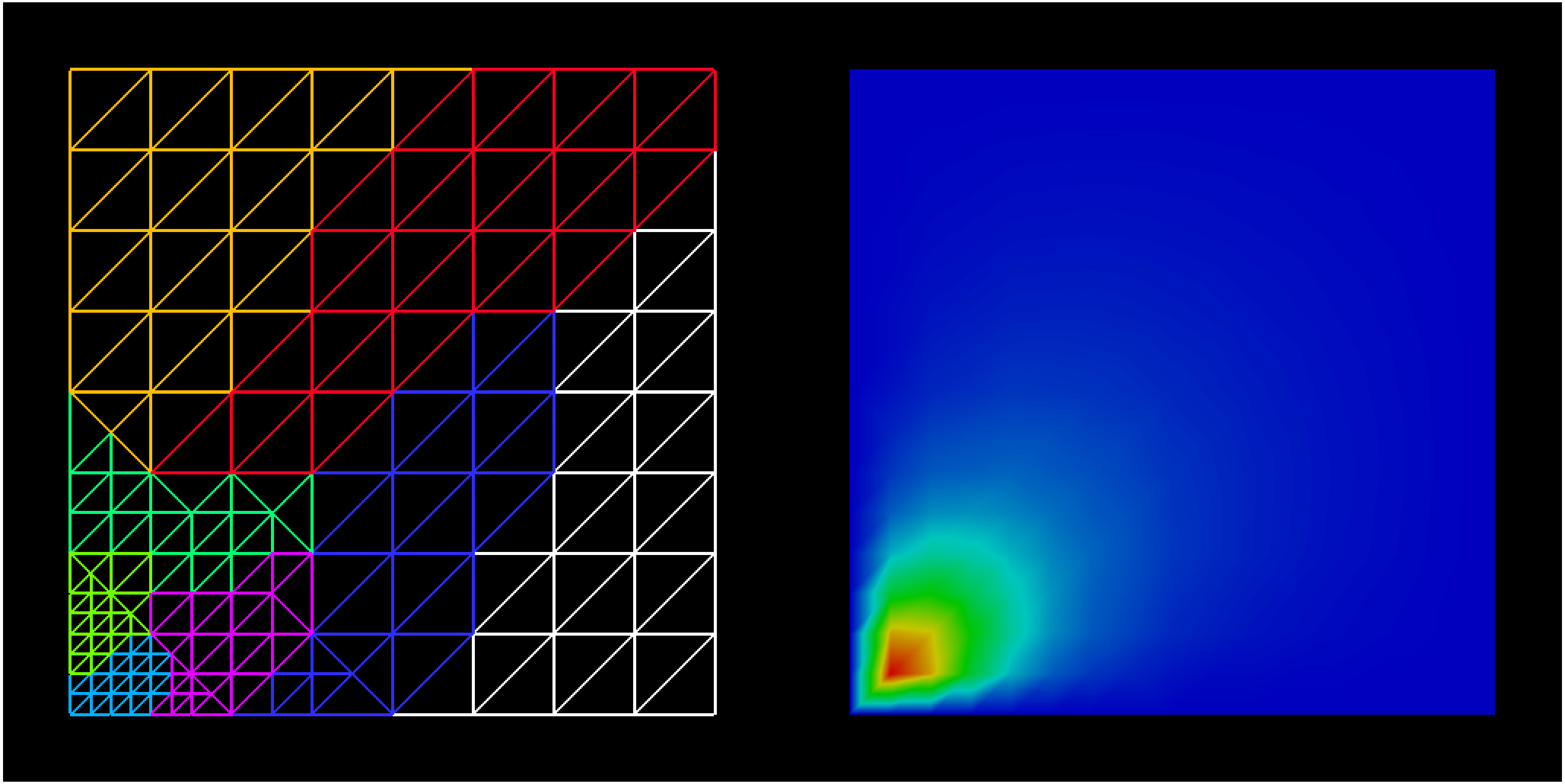
- Refine cells by edge bisection
- Same algorithms as in serial can be implemented
- Need to communicate new vertices between processes before connecting topology
- Possible to repartition and do rudimentary load balancing using ParMETIS or Zoltan PHG to repartition

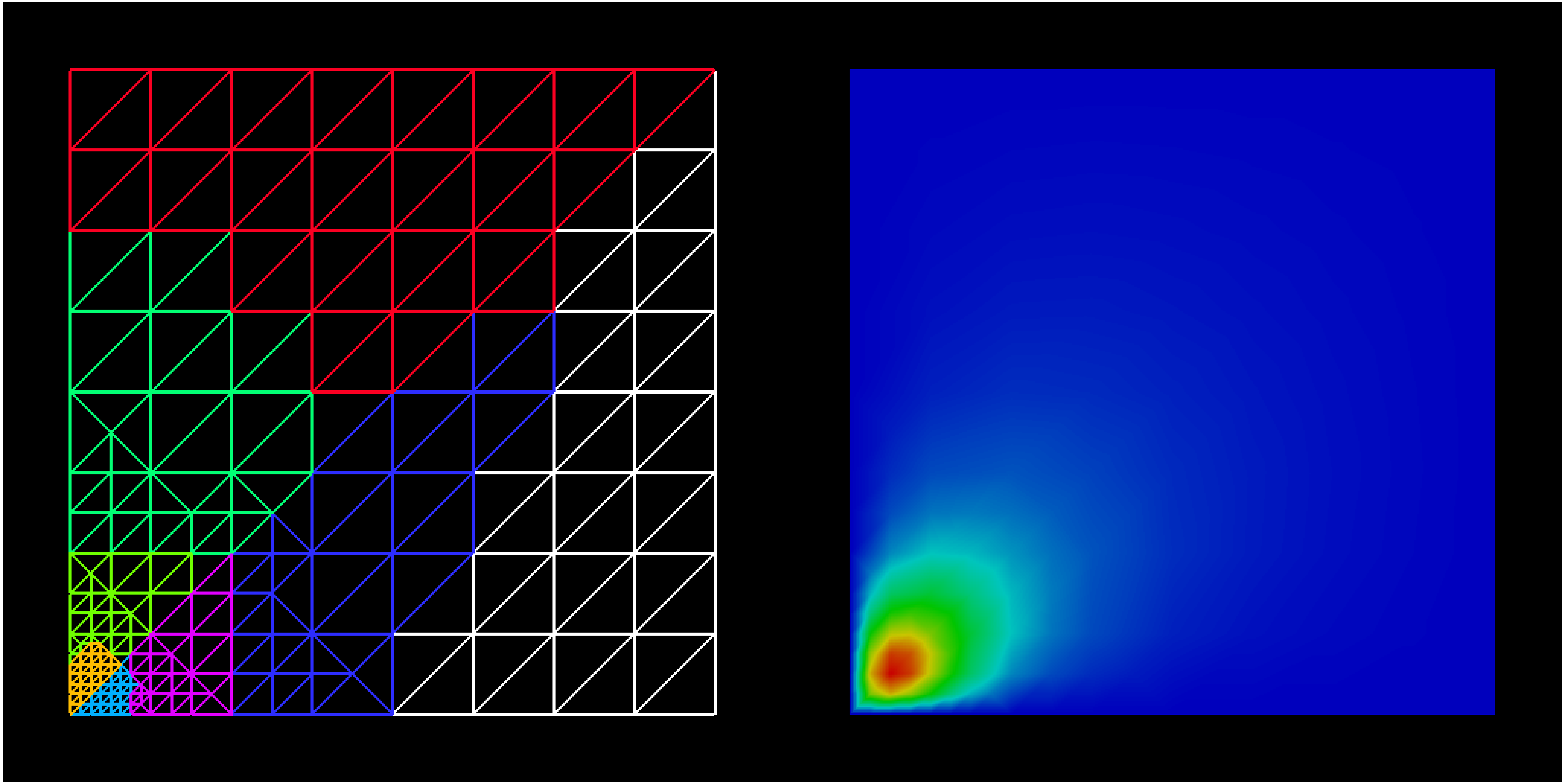
demo_adaptive_poisson.py

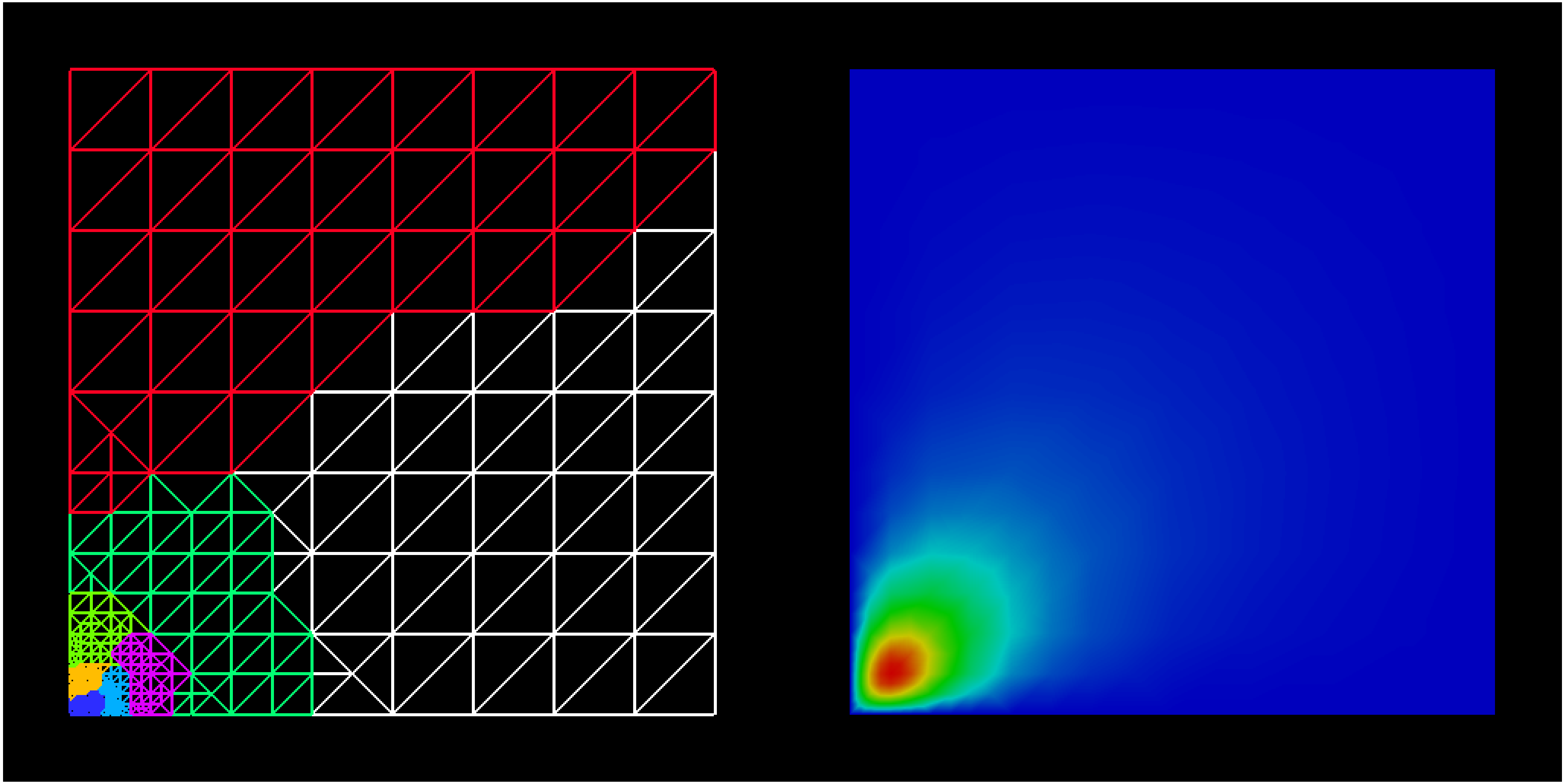


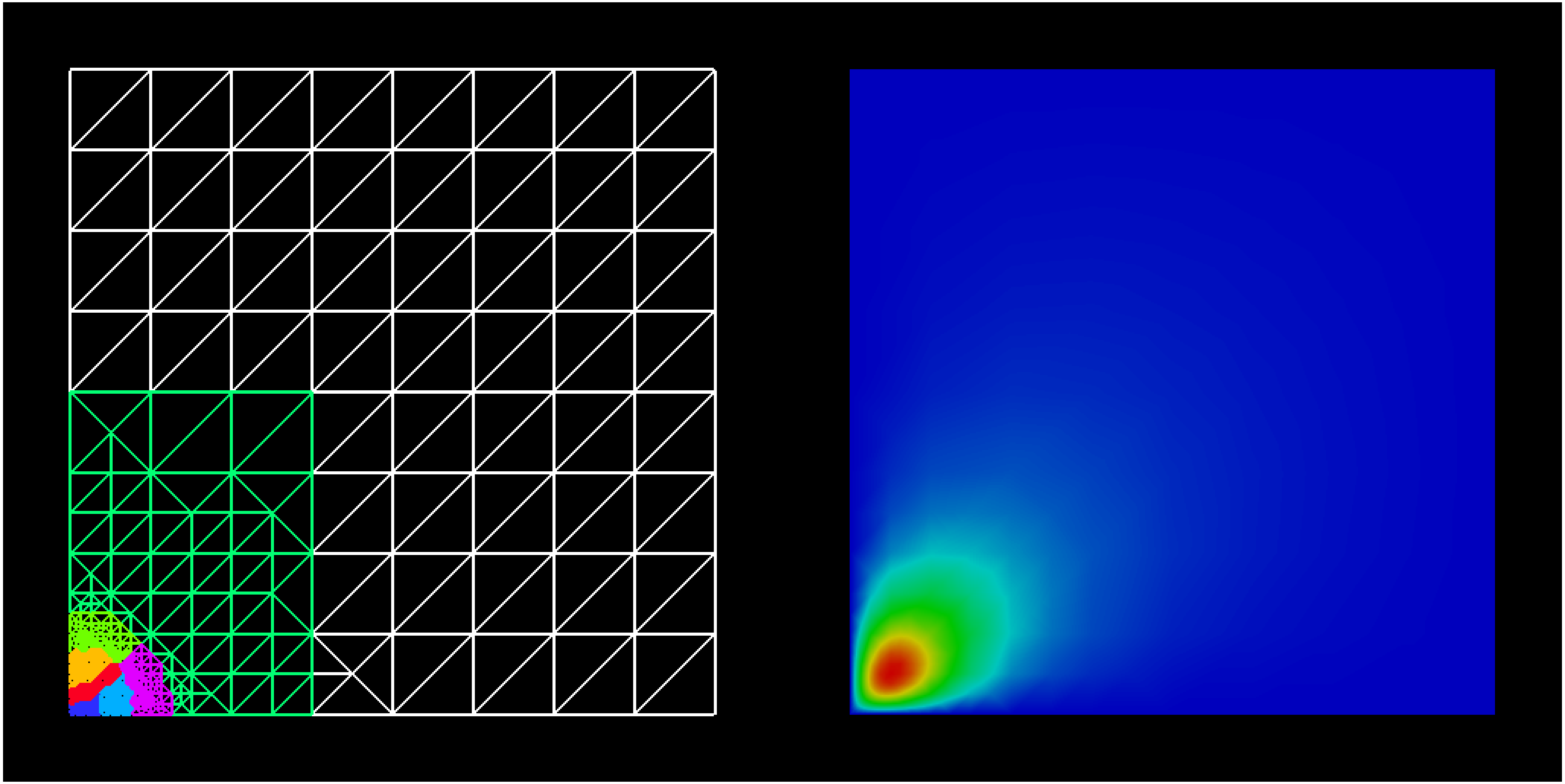
Processes (8)

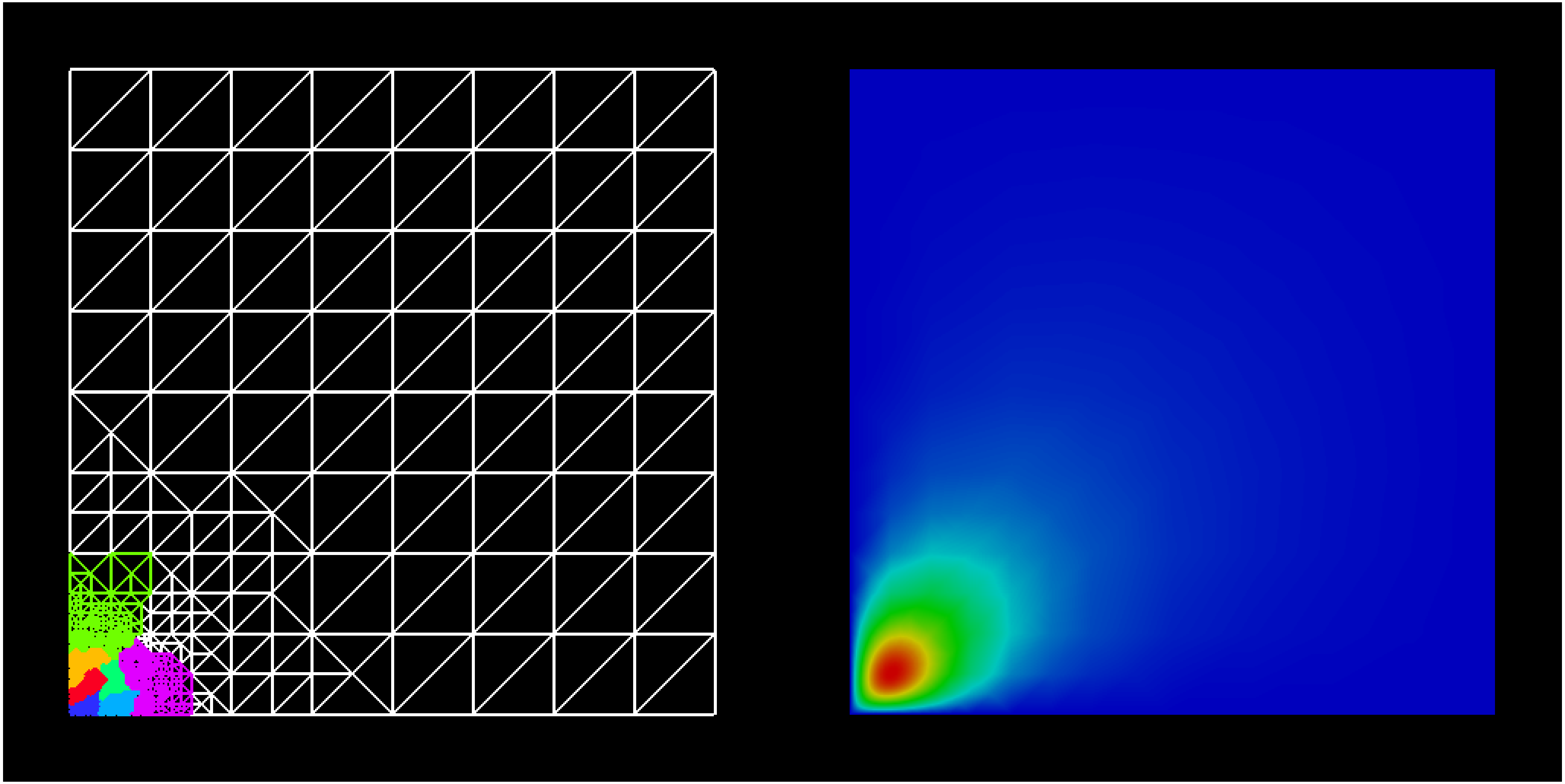
Solution

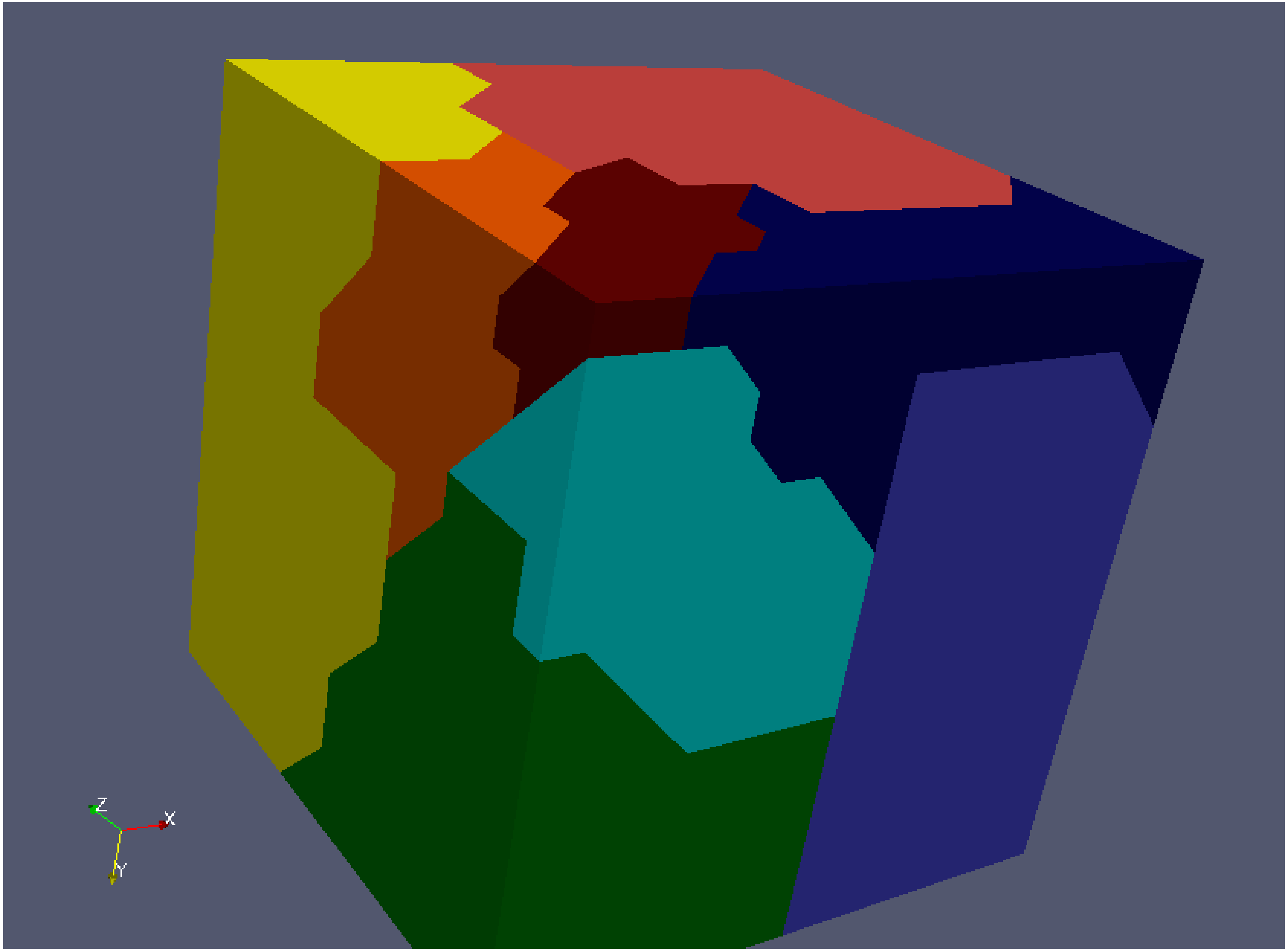


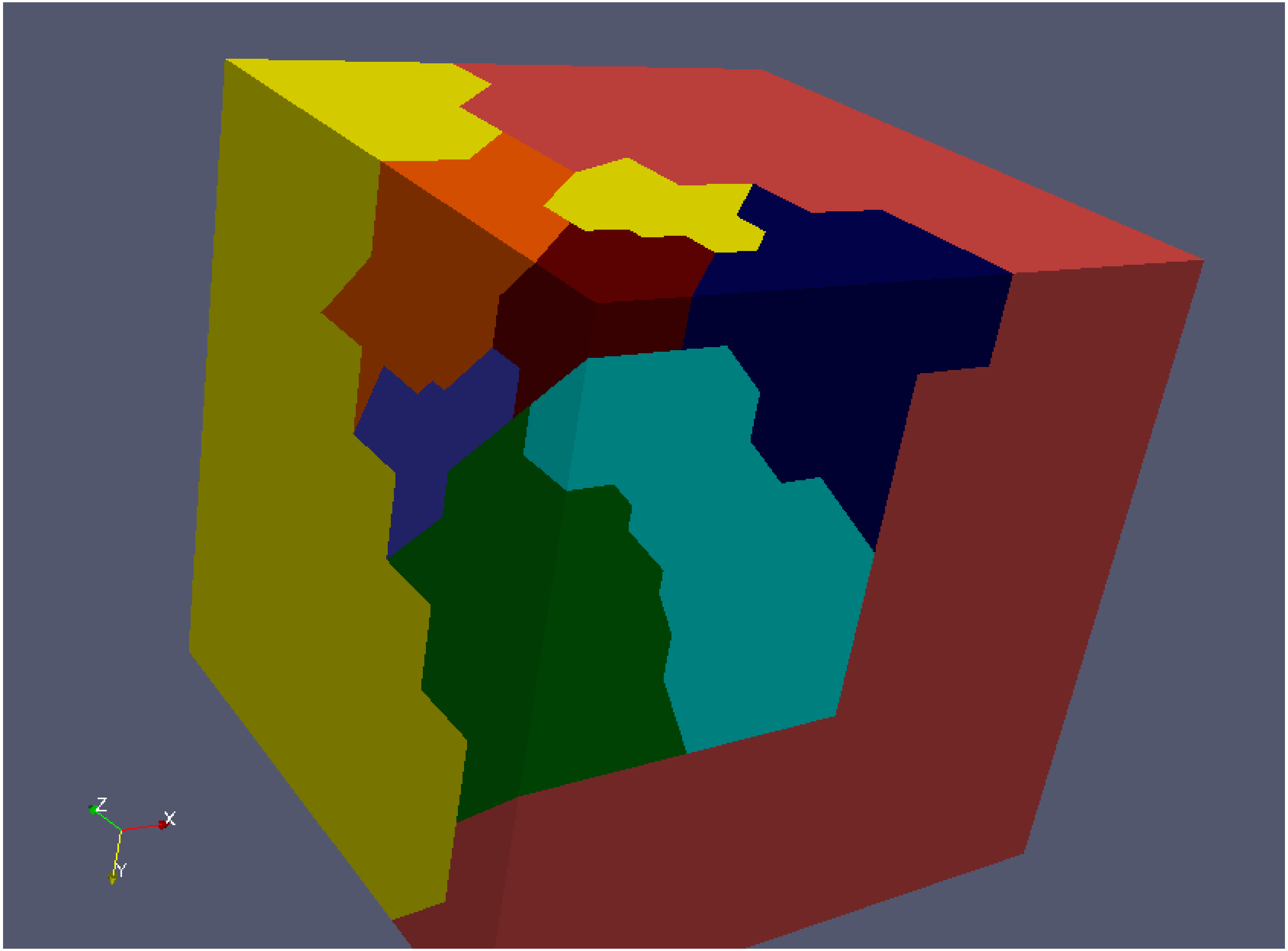


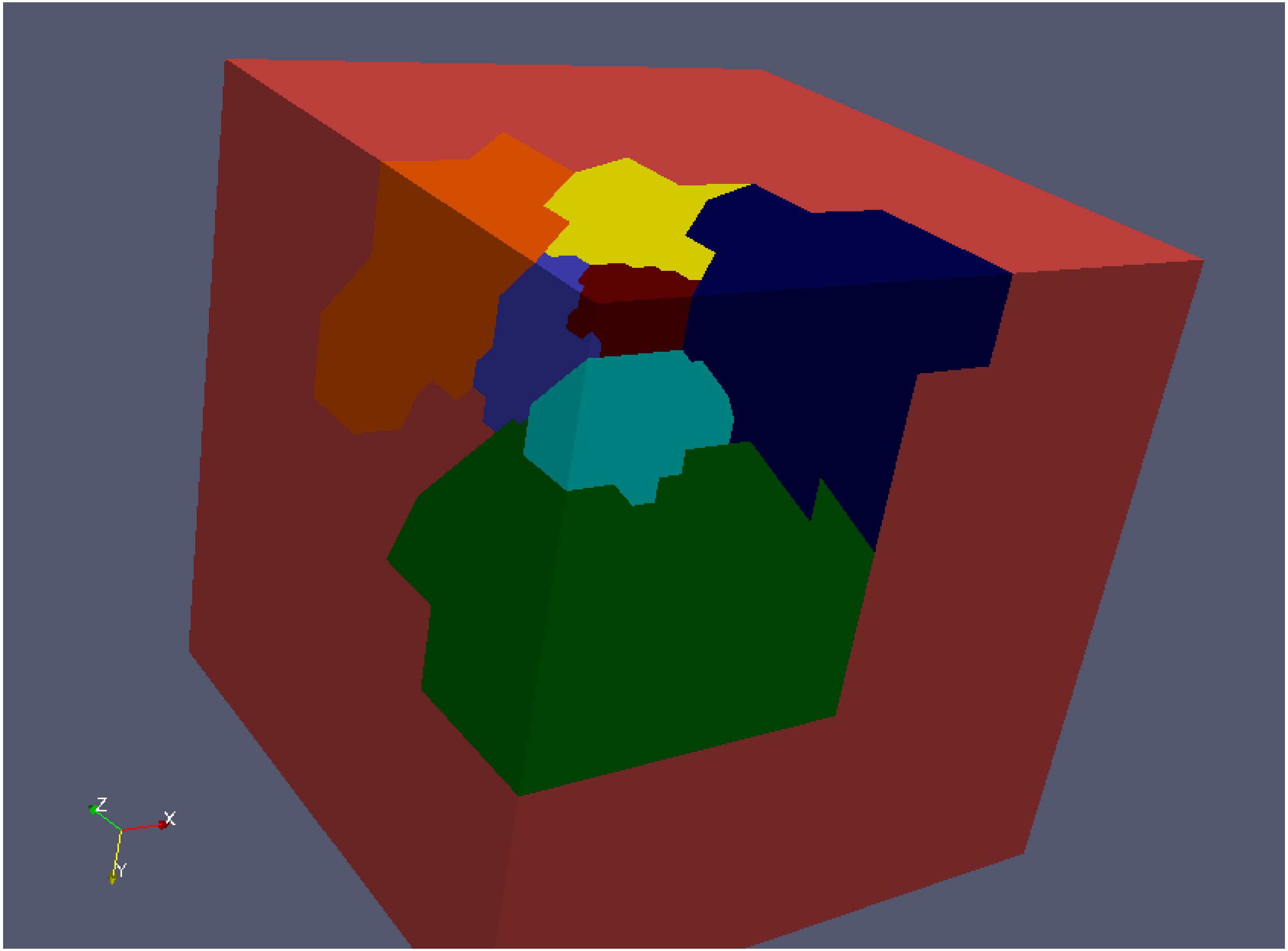


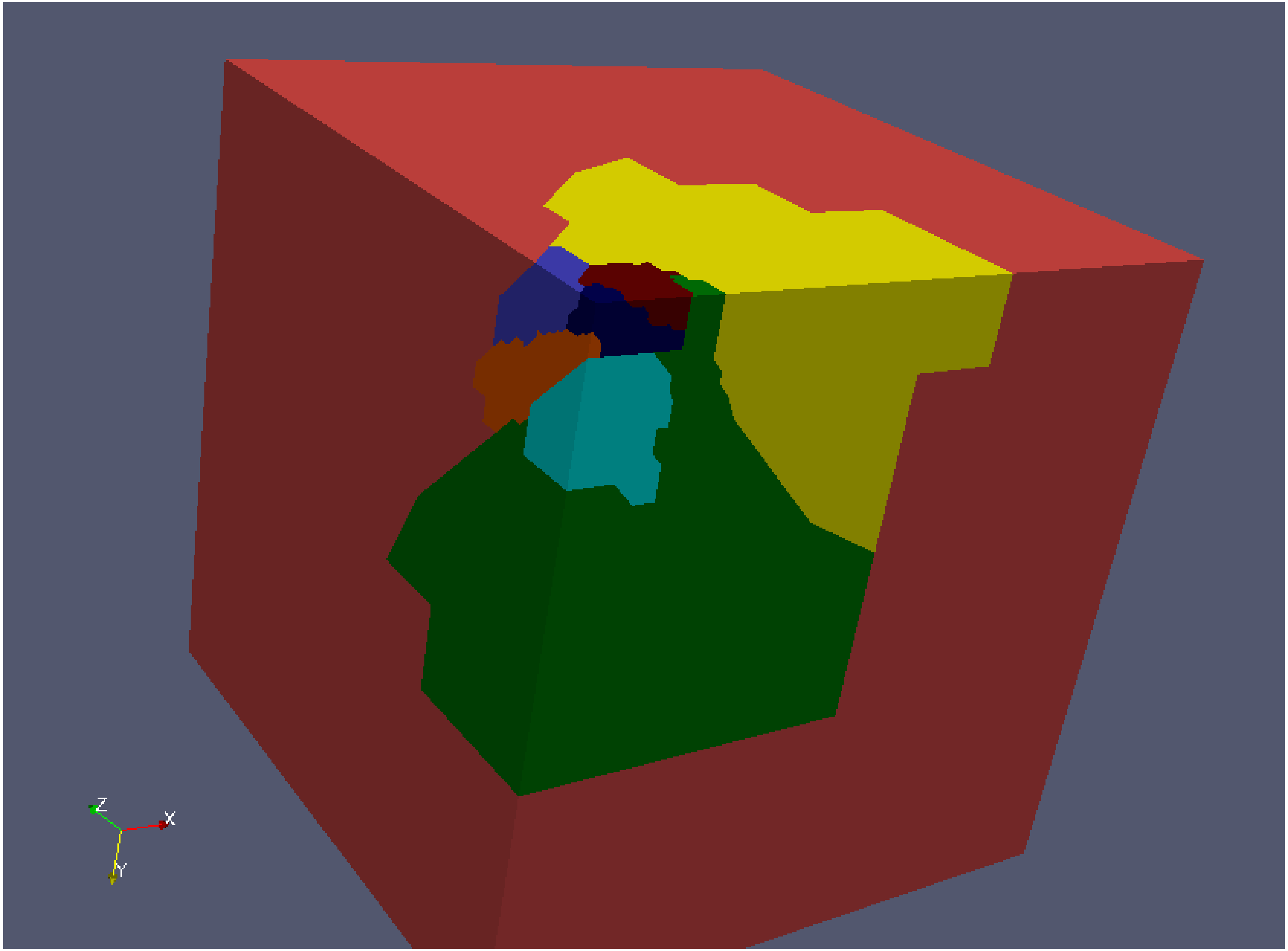


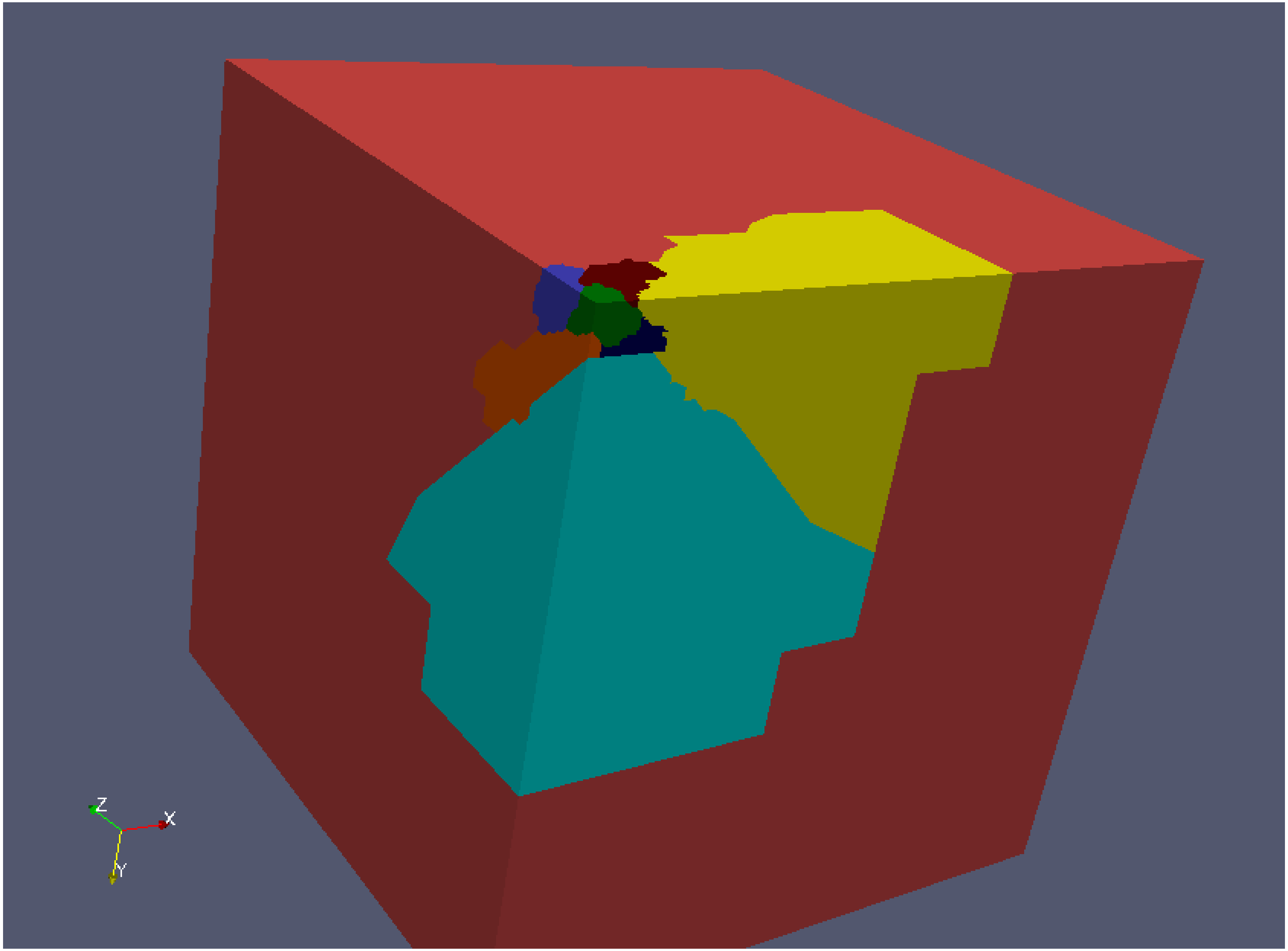


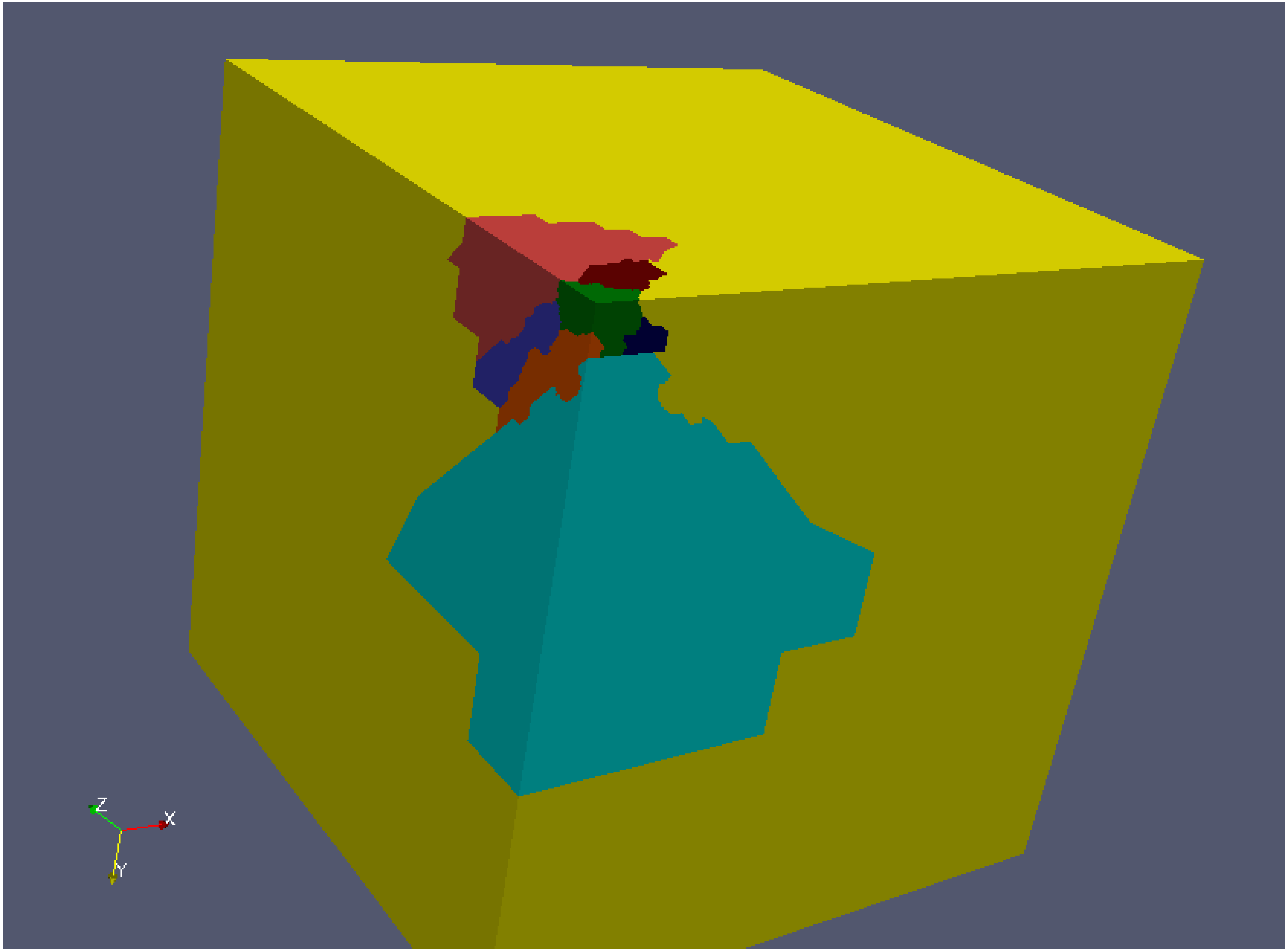

Z
X
Y


Z
X
Y


Z
X
Y


Z
X
Y


Z
X
Y


Z
X
Y

Parallel Refinement: Issues

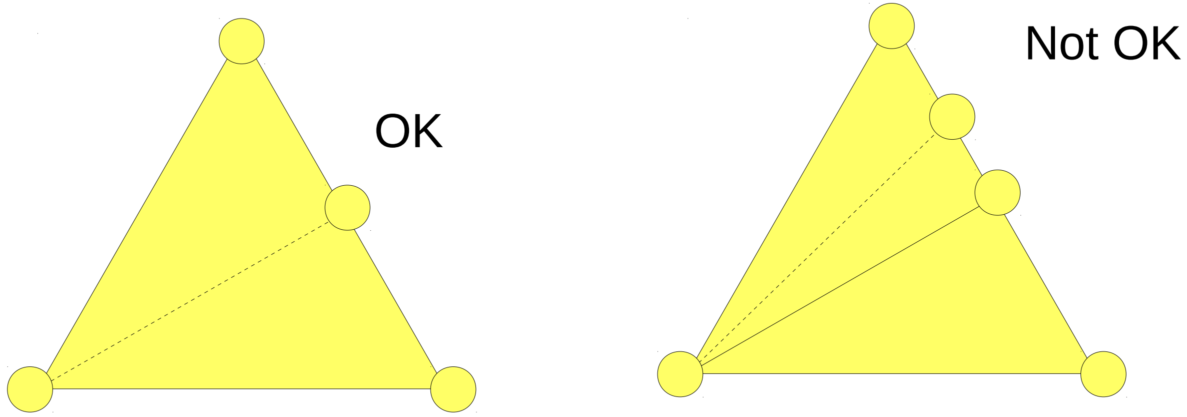
- Choosing refinement algorithms – storing mesh data between refinements to ensure quality
- Interpolating user data between meshes
- Coarsening and multilevel algorithms

Mesh Quality

- In 2D, judicious bisection can preserve the similarity shapes of the mesh

  (e.g. Carstensen algorithm)

- In 3D, it is more difficult (!)



- Need to remember bisected cells and

  re-refine them properly if touched again