FEP: Assignment 3

Justin Clough, RIN:661682899

May 30, 2017

Algorithm Description

Operation 1

The number of mesh regions for each part was determined by first iterating over all mesh vertices. For each mesh vertex, its third dimensional adjacencies, surrounding mesh regions, were collected into a temporary vector. The contents of this temporary vector were then added to another data structure if it did not already exist in the other structure. The size of this data structure was then the number of regions on each part. The mesh vertices were then iterated over again. For each mesh vertex, it was first checked if it was owned by the operating process. If it was owned, then the local and global identification numbers were recorded. The number of adjacent regions was recorded as well for each vertex. The results for this operation are shown in the results section. Printed out content from process zero is followed by process one.

Operation 2

A similar process as shown in Operation 1 was used. The number of mesh faces for each part was determined by first iterating over all mesh vertices. For each mesh vertex, its first dimensional adjacencies, surrounding mesh edges, were collected into a temporary vector. The contents of this temporary vector were then added to another data structure if it did not already exist in the other structure. The size of this data structure was then the number of edges on each part. The mesh vertices were then iterated over again. For each mesh vertex, it was first checked if it was owned by the operating process. If it was owned, then the local and global identification numbers were recorded. The number of adjacent faces was recorded as well for each vertex. The results for this operation are shown in the results section. Printed out content from process zero is followed by process one.

Operation 3

The number of mesh faces on the partition model face before migration was determined by iterating over all mesh faces. An integer counter was started at zero and was incremented if a mesh face was on the part boundary. After all mesh faces were iterated on, the value of the counter was recorded. Next, a migration plan was constructed. Process zero then iterated over all mesh regions. For each mesh region, it was first checked if it was owned by process zero. If it was, then its second dimension adjacencies were collected in a temporary vector. The faces in this temporary vector were then iterated over and each checked if they were on the part boundary. If any of each region's faces were on the part boundary, then the region was added to another data structure. The region was only added to this other data structure if it had not already been added. The regions in this data structure were then iterated over. Each region was then added to the migration plan and its local identification number was recorded. Mesh migration was then performed and the resulting mesh was verified. The mesh faces were then iterated over again. A counter was stared at zero and was incremented each time a face was found on the boundary. The results for this operation are shown in the results section. Printed out content from process zero is followed by process one. The code to execute the above three process begins on page 5.

Operation 1 Results

```
Process 0 found 28 regions.

Vertex number owned by this part (G: Global ID, L: Local ID) and
    number of bounded regions (R):
G:       L:       R:
0        6        2
1        7        5
2        8        2
3        9        5
4        10       2
5        11       10
6        15       10
7        16       5
8        17       5
9        18       16
10       19       2
```

```
Process 1 found 20 regions.

Vertex number owned by this part (G: Global ID, L: Local ID) and
    number of bounded regions (R):
G:       L:       R:
11       0        4
12       1        2
13       2        4
14       3        3
15       4        2
16       5        2
17       6        4
18       7        4
19       8        4
20       9        4
21       10       4
```

```
22          11          12
23          12          2
24          13          3
25          14          4
26          15          2
27          16          2
28          17          2
29          18          2
30          19          16
31          20          4
32          21          2
33          22          4
34          23          2
35          24          2
36          25          8
37          26          4
38          27          2
39          28          2
```

Operation 2 Results

```
Process 0 found 65 edges.

Vertex number owned by this part (G: Global ID, L: Local ID) and
    number of bounded faces (F):
G:          L:          F:
0           6           5
1           7           10
2           8           5
3           9           10
4           10          5
5           11          19
6           15          19
7           16          10
8           17          10
9           18          28
10          19          5
```

```
Process 1 found 74 edges.

Vertex number owned by this part (G: Global ID, L: Local ID) and
    number of bounded faces (F):
G:          L:          F:
11          0           9
12          1           5
13          2           8
14          3           7
```

```
15          4           5
16          5           5
17          6           8
18          7           8
19          8           8
20          9           8
21          10          8
22          11          22
23          12          5
24          13          7
25          14          8
26          15          5
27          16          5
28          17          5
29          18          5
30          19          24
31          20          8
32          21          5
33          22          8
34          23          5
35          24          5
36          25          16
37          26          8
38          27          5
39          28          5
```

Operation 3 Results

```
Process 0 found 8 faces on the part boundary before migration.

The following regions (local ID) on part 1 have at least one
   face on the part boundary before migration:
0
1
3
5
7
8
9
Process 0 found 5 faces on the part boundary after migration.
```

```
Process 1 found 8 faces on the part boundary before migration.
Process 1 found 5 faces on the part boundary after migration.
```

Code

```
//PUMI Headers
#include <PCU.h>
#include <pumi.h>

//STL Headers
#include <iostream>
#include <fstream>
#include <set>
#include <vector>
#include <string>
#include <sstream>

using std::cout;
using std::endl;

class collection_t
{
  public:
    std::set<pMeshEnt> set;
    std::vector<pMeshEnt> vector;
    void add_unqs(std::vector<pMeshEnt> ents)
    {// Add element only if unique
      for(int i=0; i<int(ents.size());i++)
      {
        pMeshEnt e = ents[i];
        add_unq(e);
      }
      return;
    }
    void add_unq(pMeshEnt e)
    {
      if(set.count(e) == 0)
      {
        set.insert(e);
        vector.push_back(e);
      }
      return;
    }
    void vectorize(std::vector<pMeshEnt>& in_vec)
    {
      in_vec = vector;
    }
    int size()
    { set.size();}
};
```

```
template <typename T>
  std::string make_string( T num)
  {
    std::ostringstream ss;
    ss << num;
    return ss.str();
  }

void operation_1(pGeom geom, pMesh mesh)
{ // Determine the number of regions using each mesh vertex
   accounting for the full mesh (including edges on other parts)
  // For each part, list only the vertices owned by that part
     with local vertex number and the number of mesh
  //    regions that vertex bounds (on all parts).

  // Create a container for pointer to mesh regions
  collection_t regions;

  // Loop over mesh vertices, place unique adjacent regions in a
      set
  pMeshIter it;
  pMeshEnt vertex;
  it = mesh->begin(0);
  while((vertex = mesh->iterate(it)))
  {
    std::vector<pMeshEnt> adj_ents;
    pumi_ment_getAdj(vertex, 3, adj_ents);
    regions.add_unqs(adj_ents);
    adj_ents.clear();
  }
  mesh->end(it);

  // Have each process write its results for part 1 to a file
  std::ofstream Op1Results;
  std::string fName_s = "Op1Results_Proc" + make_string(
     pumi_rank()) +".txt";
  char* fName = &fName_s[0u];
  Op1Results.open( fName);
  Op1Results << "Process " << pumi_rank() << " found " <<
     regions.size() << " regions.\n";

  Op1Results << "\nVertex number owned by this part (G: Global
     ID, L: Local ID) and number of bounded regions (R):\n";
  Op1Results << "G: \tL:\tR:\n" ;
  it = mesh->begin(0);
  while((vertex = mesh->iterate(it)))
  {
    if(pumi_ment_isOwned(vertex))
    {
      int localID = pumi_ment_getID(vertex);
      int globalID = pumi_ment_getGlobalID(vertex);
      Op1Results << globalID << "\t" ;
      Op1Results << localID << "\t" ;
      int num_adj = pumi_ment_getNumAdj(vertex, 3);
      Op1Results << num_adj << "\n";
    }
  }
  mesh->end(it);
  Op1Results.close();

  return;
} //END operation_1()

void operation_2(pGeom geom, pMesh mesh)
{ // Determine the number of edges using each mesh vertex
   accounting for the full mesh (included other parts).
  // For each part, list only the vertices owned by that part
     with local vertex number and the number fo mesh
  //    faces that vertex bounds (on all parts).

  // Create a container for pointers to mesh edges
  collection_t edges;

  // Loop over mesh vertices, place unique adjacent edges in a
     the container
  pMeshIter it;
  pMeshEnt vertex;
  it = mesh->begin(0);
  while((vertex=mesh->iterate(it)))
  {
    std::vector<pMeshEnt> adj_ents;
    pumi_ment_getAdj(vertex, 1, adj_ents);
    edges.add_unqs(adj_ents);
    adj_ents.clear();
  }
  mesh->end(it);

  // Have each process write its results for part 2 to a file
  std::ofstream Op2Results;
  std::string fName_s = "Op2Results_Proc" + make_string(
     pumi_rank()) + ".txt";
  char* fName = &fName_s[0u];
  Op2Results.open( fName);
  Op2Results << "Process " << pumi_rank() << " found " << edges.
     size() << " edges.\n";
  Op2Results << "\nVertex number owned by this part (G: Global
     ID, L: Local ID) and number of bounded faces (F):\n";
  Op2Results << "G:\tL:\tF:\n";
  it = mesh->begin(0);
  while((vertex = mesh->iterate(it)))
  {
    if(pumi_ment_isOwned(vertex))
    {
      int localID = pumi_ment_getID(vertex);
      int globalID = pumi_ment_getGlobalID(vertex);
      Op2Results << globalID << "\t";
      Op2Results << localID << "\t";
      int num_adj = pumi_ment_getNumAdj(vertex, 2);
      Op2Results << num_adj << "\n";
    }
  }
  mesh->end(it);
  Op2Results.close();

  return;
} //END operation_2()

void operation_3(pGeom geom, pMesh mesh)
{ // Count the number of mesh faces on the partition model face
   before migration.
  // List the mesh regions on Part 1 that have at least one mesh
      face on the partition model face between 1 and 2.
  // Migrate mesh regions from Part 1 to 2.
  // Count the number of mesh faces on the partition model face
     after migration.

  std::ofstream Op3Results;
  std::string fName_s = "Op3Results_Proc" + make_string(
     pumi_rank()) + ".txt";
  char* fName = &fName_s[0u];
  Op3Results.open( fName);

  int pre_mig = 0;
  pMeshEnt face;
  pMeshIter it;
  it = mesh->begin(2);
  while((face = mesh->iterate(it)))
  {
    if(pumi_ment_isOnBdry( face))
    {
      pre_mig++;
    }
  }
```

```
mesh->end(it);
Op3Results << "Process " << pumi_rank() << " found "
        << pre_mig << " faces on the part boundary before
           migration.\n" ;

Migration* plan = new Migration(mesh);
if( pumi_rank() == 0)
{
  Op3Results << "\nThe following regions (local ID) on part 1
     have at least one face on the part boundary before
     migration:\n";
  collection_t mig_regs;
  pMeshEnt region;
  it = mesh->begin(3);
  while((region = mesh->iterate(it)))
  {
    if(pumi_ment_isOwned(region))
    {
      std::vector<pMeshEnt> adj_faces;
      pumi_ment_getAdj(region, 2, adj_faces);
      for(int i=0; i<(int)adj_faces.size(); i++)
      {
        if(pumi_ment_isOnBdry(adj_faces[i]))
        {
          mig_regs.add_unq(region);
        }
      }
      adj_faces.clear();
    }
  }
  mesh->end(it);
  std::vector<pMeshEnt> regs;
  mig_regs.vectorize(regs);
  for(int i=0; i<(int)regs.size();i++)
  {
    Op3Results << pumi_ment_getID(regs[i]) << "\n";
  }

  for(int i=0; i<(int)mig_regs.size(); i++)
  {
    plan->send(mig_regs.vector[i], pumi_size()-1);
  }
}

pumi_mesh_migrate(mesh, plan);
pumi_mesh_verify(mesh);

int post_mig = 0;
```

```
  it = mesh->begin(2);
  while((face = mesh->iterate(it)))
  {
    if(pumi_ment_isOnBdry( face))
    {
      post_mig++;
    }
  }
  mesh->end(it);
  Op3Results << "Process " << pumi_rank() << " found "
          << post_mig << " faces on the part boundary after
             migration.\n" ;

  Op3Results.close();
  return;
} //END operation_3()

int main(int argc, char** argv)
{
  MPI_Init(&argc,&argv);
  pumi_start();
  pGeom geom = pumi_geom_load("cube.dmg","mesh");
  pMesh mesh = pumi_mesh_load(geom,"parallelMesh.smb",2);

  operation_1( geom, mesh);
  operation_2( geom, mesh);
  operation_3( geom, mesh);

  pumi_mesh_delete(mesh);
  pumi_finalize();
  MPI_Finalize();
}
```