HPSC Lab 4

Luna McBride, Lily Strus

September 28 2023

1 Parallel Design

This lab takes the information presented in previous labs and cranks it up to 11. The code from the previous two labs exist here already as well. The major aspects of the code revolve around the new to implement PESum method in mpiInfo as well as the updated mesh, in the newly built LaplacianOnGrid. These pieces work together to define the electrical field of the given particles, which is the primary focus of the lab.

The field is calculated based off of particle proximity to nodes of the grid, which are pulled from the cells immediately next to the specified node. This means that each node can have anywhere between 1 and 4 cells within its jurisdiction depending on where in the mesh it is. The baseline of cell interaction is defined in LaplacianOnGrid. Much like other aspects of the grid scenario, however, the PE boundaries need to be considered when calculating the charge. The logic follows that, for any arbitrary node on the boundary between two PEs, said node represents the same place in the specified mesh even though they are physically in very different places from one another in the cores. This is where PESum comes in, as it stitches the mesh together and ensures that the charge is calculated for every cell around it like any other node with multiple cells around it.

1.1 LaplacianOnGrid

There are two major parts to LaplacianOnGrid that occur, each important, but one more important. The first part is fundamental in building out the mesh, defining ghost nodes through various BC methods, as well as implementing the Laplacian equation that is in the name. The second part, specifically the ParticlesOnMesh method, is the more important portion to our purposes here. This part starts by updating inactive particles by moving them to the PE they are set to move to. This then puts them in their proper positions to pull the charge from.

The particles are then accumulated into their proper nodes via the Qval vector, which is a vector accessible by the entire class. This is then pushed to PESum, that way the full electrical gradient could be calculated between the PEs. These are then used to compute the gradient between nodes of each cell, checking between one node and its neighbor. Now that we have each of those values, the flow from left to right and top to bottom can now be calculated. That is to say, all of the previous work on this mesh was done to define those boundaries so that the actual flow calculation could be done.

1.2 PESum

The idea of the method is simple. The node just need to sum up the cells around it just like any other node does in the LaplacianOnGrid mesh. This, however, becomes more complex by the fact that these elements are physically separated from one another. The two nodes representing the same node need to have communication to build a shared memory value between them. Luckily, theoretically, these values can be thrown back and forth using MPI commands as if they are not actually large distances away from one another. This needs to be done carefully however, as stitching from multiple directions at once may throw off everything in cases where both east-west traffic and north-south traffic need to be accounted for.

The solution here is to send all of one set first, then move to the other. For us with measily human brains, it makes more sense to get east-west traffic going first then move on to north-south traffic. This is because east-west traffic fits well with moving countably in both ij notation and numeric notation, as we tend to number these left to right as we would read in many different languages, including English. By focusing on one direction set at a time, we can guarantee that the nodes have correct values in cases where both sets need to be accounted. Otherwise, this could not be assumed due to parallel programming's problem with different cores getting things done at different rates.

2 Self Reflection

2.1 Luna

This week was rough, but not for the lab itself. From a hectic string of work days, my own heavy pain, appointments, discussions with work about restrictions turning to medical leave, to having my duo two factor authentication becoming locked, there was not much I could do. That is to say, there was not much time I could spend this week and a chunk of time I could spend had me locked out. I could not even go in and fix/test the code to no longer account for the diagonal after our conversations about it after class.

2.2 Lily

3 Appendix A: Current Code

```
1  //  ==
2  //  ||
3  //  || Routine PEsum
4  //  ||
5  //  || This routine receives a field variable named "field"; it has a value for
6  //  || each node in the mesh local to this PE.  However, its value on this PE's
7  //  || boundary nodes is lacking the contributions from the neighboring
     processor.
8  //  || Here, values in "field" are exchanged between neighboring processors so
     that
9  //  || each processor can add the contributions from the neighboring processor.
10 //  ||
11 //  ==
12
```

```
13
14  void PEsum( VD &field )
15  {
16
17    // TO-DO
18    //
19    // Use what you have learned in the past labs to sum values on the processor
20    // boundaries *and* ensure that the processes owning those shared nodes
21    // all have that same summed value.
22    //
23    //
24    double *QSend_n, *QSend_s, *QSend_e, *QSend_w;
25    double *QSend_ne, *QSend_nw, *QSend_se, *QSend_sw;
26    double *QRecv_n, *QRecv_s, *QRecv_e, *QRecv_w;
27    double *QRecv_ne, *QRecv_nw, *QRecv_se, *QRecv_sw;
28    // initialize arrays/variables you'll be sending; exclude ghost nodes
29    QSend_n  = new double [ nRealx ];
30    QSend_s  = new double [ nRealx ];
31    QSend_w  = new double [ nRealy ];
32    QSend_e  = new double [ nRealy ];
33    QSend_ne  = new double [ nRealx ];
34    QSend_nw  = new double [ nRealx ];
35    QSend_se  = new double [ nRealx ];
36    QSend_sw  = new double [ nRealx ];
37    for ( int i = 0 ; i < nRealx ; ++i ) { QSend_nw[i] = 0.; QSend_ne[i] = 0.;
      QSend_se[i] = 0.; QSend_sw[i] = 0.;}
38    cout << "Send Init" << endl;
39
40
41    // initialize arrays you'll be receiving; match dimensions with physical nodes
42    QRecv_n = new double [ nRealx ];
43    QRecv_s = new double [ nRealx ];
44    QRecv_w = new double [ nRealy ];
45    QRecv_e = new double [ nRealy ];
46    QRecv_ne = new double [ nRealx ];
47    QRecv_nw = new double [ nRealx ];
48    QRecv_se = new double [ nRealx ];
49    QRecv_sw = new double [ nRealx ];
50    for ( int i = 0 ; i < nRealx ; ++i ) { QRecv_nw[i] = 0.; QRecv_ne[i] = 0.;
      QRecv_se[i] = 0.; QRecv_sw[i] = 0.;}
51    cout << "Recieve Init" << endl;
52
53
54    // fill send arrays/variables with field vals on boundaries
55    sLOOP QSend_n[s] = field[ pid(s,nRealy + 1) ];
56    sLOOP QSend_s[s] = field[ pid(s,1) ];
57    tLOOP QSend_w[t] = field[ pid(1,t) ];
58    tLOOP QSend_e[t] = field[ pid(nRealx + 1,t) ];
59
60    cout << "Normal Definition" << endl;
61    if ( nei_ne >=0 ) sLOOP QSend_ne[s] = field[ pid(s + 1,nRealy) ];
62    if ( nei_nw >=0 ) sLOOP QSend_nw[s] = field[    pid(s - 1,nRealy)   ];
63    if ( nei_se >=0 ) sLOOP QSend_se[s] = field[    pid(s + 1,1)   ];
64    if ( nei_sw >=0 ) sLOOP QSend_sw[s] = field[       pid(s - 1,1)     ];
65    cout << "Diagonal Definition" << endl;
```

```
66
67      // Send your boundaries to neighbors
68      if ( nei_n  >= 0 ) { err = MPI_Isend(  QSend_n, nRealx, MPI_DOUBLE,  nei_n, 1,
         MPI_COMM_WORLD, &request); }
69      if ( nei_s  >= 0 ) { err = MPI_Isend(  QSend_s, nRealx, MPI_DOUBLE,  nei_s, 1,
         MPI_COMM_WORLD, &request); }
70      if ( nei_w  >= 0 ) { err = MPI_Isend(  QSend_w, nRealy, MPI_DOUBLE,  nei_w, 1,
         MPI_COMM_WORLD, &request); }
71      if ( nei_e  >= 0 ) { err = MPI_Isend(  QSend_e, nRealy, MPI_DOUBLE,  nei_e, 1,
         MPI_COMM_WORLD, &request); }
72      if ( nei_ne >= 0 ) { err = MPI_Isend( QSend_ne,      nRealx, MPI_DOUBLE,
        nei_ne, 1, MPI_COMM_WORLD, &request); }
73      if ( nei_nw >= 0 ) { err = MPI_Isend( QSend_nw,      nRealx, MPI_DOUBLE,
        nei_nw, 1, MPI_COMM_WORLD, &request); }
74      if ( nei_se >= 0 ) { err = MPI_Isend( QSend_se,      nRealx, MPI_DOUBLE,
        nei_se, 1, MPI_COMM_WORLD, &request); }
75      if ( nei_sw >= 0 ) { err = MPI_Isend( QSend_sw,      nRealx, MPI_DOUBLE,
        nei_sw, 1, MPI_COMM_WORLD, &request); }
76      cout << "Send" << endl;
77
78      // Receive your neighbors' boundaries
79      if ( nei_n >= 0 ) {
80        err = MPI_Irecv(  QRecv_n, nRealx, MPI_DOUBLE, nei_n,  MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
81          MPI_Wait( &request, &status ); }
82        cout << "RecvN" << endl;
83      if ( nei_s >= 0 ) {
84        err = MPI_Irecv(  QRecv_s, nRealx, MPI_DOUBLE, nei_s,  MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
85          MPI_Wait( &request, &status ); }
86        cout << "RecvS" << endl;
87      if ( nei_w >= 0 ) {
88        err = MPI_Irecv(  QRecv_w, nRealy, MPI_DOUBLE, nei_w, MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
89          MPI_Wait( &request, &status ); }
90        cout << "RecvW" << endl;
91      if ( nei_e >= 0 ) {
92        err = MPI_Irecv(  QRecv_e, nRealy, MPI_DOUBLE, nei_e, MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
93          MPI_Wait( &request, &status ); }
94        cout << "RecvE" << endl;
95      if ( nei_ne >= 0 ) {
96        err = MPI_Irecv( QRecv_ne,      nRealx, MPI_DOUBLE, nei_ne, MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
97          MPI_Wait( &request, &status );}
98        cout << "RecvNE" << endl;
99      if ( nei_nw >= 0 ) {
100       err = MPI_Irecv( QRecv_nw,      nRealx, MPI_DOUBLE, nei_nw, MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
101         MPI_Wait( &request, &status ); }
102       cout << "RecvNW" << endl;
103     if ( nei_se >= 0 ) {
104       err = MPI_Irecv( QRecv_se,      nRealx, MPI_DOUBLE, nei_se, MPI_ANY_TAG,
        MPI_COMM_WORLD, &request);
105         MPI_Wait( &request, &status ); }
```

```
      cout << "RecvSE" << endl;
    if ( nei_sw >= 0 ) {
      err = MPI_Irecv( QRecv_sw,      nRealx, MPI_DOUBLE, nei_sw, MPI_ANY_TAG,
   MPI_COMM_WORLD, &request);
        MPI_Wait( &request, &status ); }
    cout << "RecvSW" << endl;
    cout << "Receive" << endl;

    // Sum the boundary field vals you received with those you already have
    // first sum as if there's no contribution from NW,NE,SW,SE neighbors
    // then add those contributions as well

    // sum boundaries from N,S,E,W neighbors first
    if ( nei_n >= 0 ) { sLOOP field[ pid(s,nRealy + 1) ] += QRecv_n[s]; }
    if ( nei_s >= 0 ) { sLOOP field[   pid(s,1)    ] += QRecv_s[s]; }
    if ( nei_e >= 0 ) { tLOOP field[ pid(nRealx + 1,t) ] += QRecv_e[t]; }
    if ( nei_w >= 0 ) { tLOOP field[   pid(1,t)    ] += QRecv_w[t]; }
    cout << "Normal Sum" << endl;

    // sum boundaries from NW,NE,SW,SE neighbors
    if ( nei_ne >= 0 ) { sLOOP field[ pid(s + 1,nRealy + 1) ] += QRecv_ne[s]; }
    if ( nei_nw >= 0 ) { sLOOP field[    pid(s - 1,nRealy + 1)   ] += QRecv_nw[s];
     }
    if ( nei_se >= 0 ) { sLOOP field[    pid(s + 1,1)   ] += QRecv_se[s]; }
    if ( nei_sw >= 0 ) { sLOOP field[      pid(s - 1,1)      ] += QRecv_sw[s]; }
    cout << "Diagonal Sum" << endl;

    delete[] QSend_n;  delete[] QSend_s;  delete[] QSend_e; delete[] QSend_w;
   delete[] QSend_ne; delete[] QSend_nw; delete[] QSend_se; delete[] QSend_sw;
    delete[] QRecv_n;  delete[] QRecv_s;  delete[] QRecv_e; delete[] QRecv_w;
   delete[] QRecv_ne; delete[] QRecv_nw; delete[] QRecv_se; delete[] QRecv_sw;
  }

  int pid(int i,int j) { return (i+1) + (j)*(nRealx+2); }
};
```

Listing 1: Parallel Code (mpiInfo.h)

4 Appendix B and C: Direction Moving Forward

The first step would be to fix based off of the conversations we had on Tuesday. That would be removing checks for the diagonal, then restructuring the entire code to do the entirety of east-west traffic first. Further testing would need to happen from there.

Current testing printouts exists to find where the code is getting stuck. It would get stuck in loops either at IRecv in the diagonal area or with memory. The most recent portion added to this code was the delete[] statements to attempt to clear out memory, but it was throwing errors last I worked with it. Not crazy errors, but ones where I would need to dig deeper to see what it is trying to tell me. I mean, I literally got that typed up then realized I was running late and needed to go. Those would be areas to look into deeper. That is, if I could get in so I could look into them deeper.