HPSC Lab 11

Luna McBride, Eli Weissler

December 1 2023

1 Introduction and Lab Goals

In a High-Performance Scientific Computing setting, there are bound to be jobs so big that the system cannot build them within the allocated time slots allowed by a shared system like Alpine. This is where automating the process of restarting code can prove quite useful, as it removes the babysitting that would be required otherwise. The idea may be small, but proper implementation can remove the stress that comes from needing to be there when the process finishes in order to start it up again.

The goals of this lab are:

1. Utilize MPIIO to create a file of the current state of the mesh.
2. Inject the file with timing and mesh metadata to ensure a smooth transition upon restart.
3. Add code to periodically write these files in case the session ends.
4. Implement a read function to the beginning of the transient diffusion code so it can read this file and pick up where the last session left off.
5. Add a new Python file to check when the code ends and automatically restart it.
6. Test this file with a kill file to make sure the code both restarts and finishes correctly.

This is not a crazy lab compared to those that came before it, but it may prove even more important to long term success in similar shared-system scenarios that may come up in industry. A concept like this is not only applicable in scientific computing settings, but also may prove useful for large machine learning models and other similar jobs. It is a powerful tool to have, just as soon as it is up and running.

2 Test Case Results and Analysis

We used the 2 by 2 PE, 3 by 3 mesh test case present in the run file already. The first call evolves to time t = 0.15, while the second resumes and continues to t = 0.3. As seen in the video of the evolution, the solution continues to evolve past t = 0.15 (time-step 149 in the video), so it has yet to reach the steady state solution.

In figure 1, we compare the mesh values saved in the .plt files for the 149th time step (the final time step of the first call) to the mesh values directly after loading them into the restart call. All values are exactly the same, so the restart procedure is successfully reacquiring the necessary information.

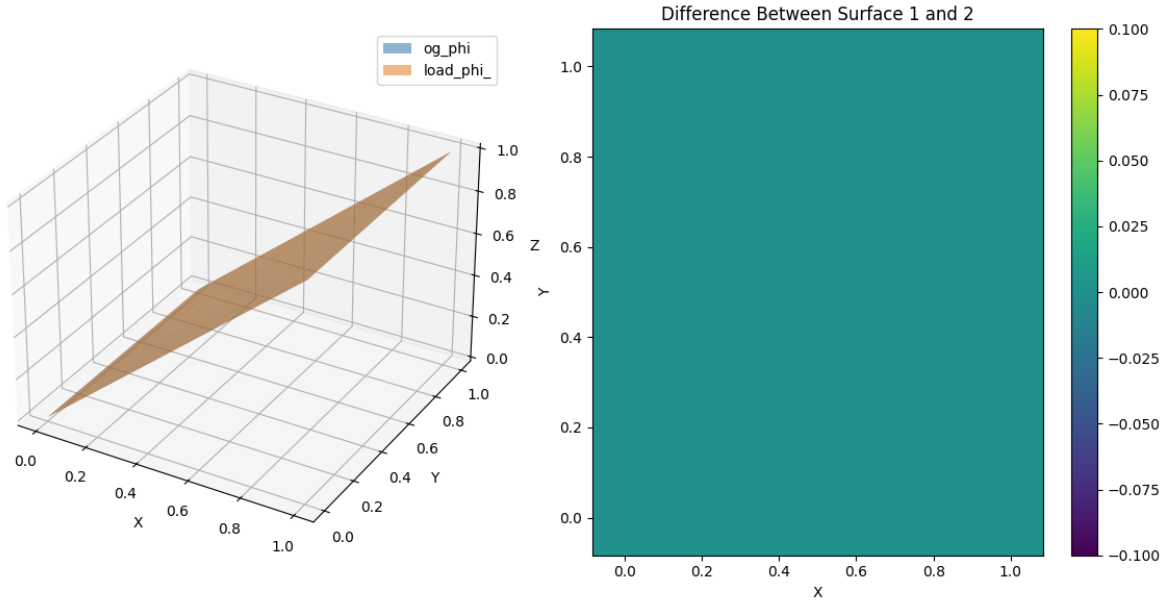

Figure 1: Comparison of the mesh values saved in the final time-step of the first call (og_phi) to the values directly after loading into the restart call (load_phi)

3 Automatic Restart Demonstration

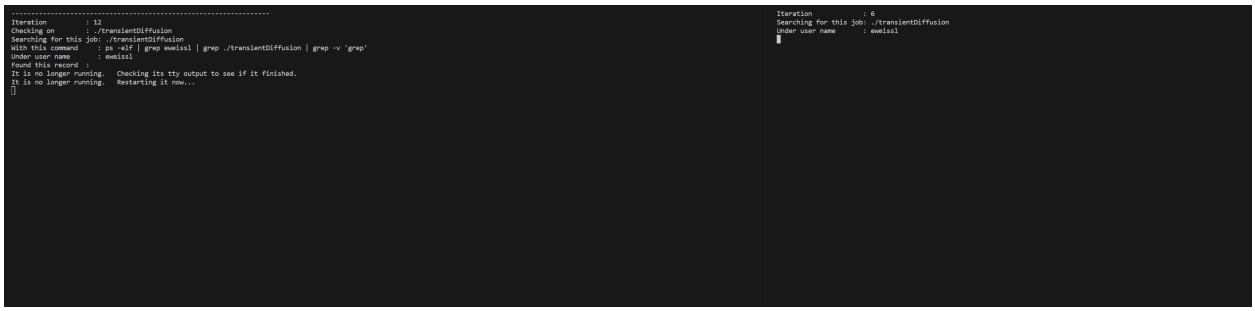

Figure 2: Screenshot showing the keep running script restarting the job after mySlurm killed it. For a full demonstration, see the Video

4 Self Evaluation

4.1 Luna

This lab was clearly not meant to be difficult. Introducing the concept and flow to such restarts is more important than making it difficult, so this was a good sandbox to play with the idea. On Demand had problems running it, however, crashing entirely while executing the task. This is a problem encountered during previous labs due to MPI and the interactive node not wanting to see eye to eye. It was simple then to just sent the code off via sbatch to avoid the issue, but this use case does not allow for it. Another option was Putty for a Windows user like myself, which accesses alpine by logging in. This makes it unable to access multiple interactive nodes at the same time. The VS Code area on On Demand was the final option, which did end up working just fine. By that point, however, I would not leave any more work for Eli if I kept going.

4.2 Eli

This lab went very smoothly! We finished shortly after lab ended on Friday, with Luna getting the fake slurm running and me recording the verification. We made good use of my python plotting utility that I had made for a previous lab, quickly plotting the differences between multiple surfaces and stitching together videos to study the evolution. Using the script we discovered that we were off by one time-step with our initial solution.Because data is saved at the end of the time-stepping loop, data saved at $t = n_{step}dt$ is actually the data you would have at the beginning of the $t = (n_{step}+1)dt$ step. Of course this difference was sufficiently small we would have never discovered it from looking at the surface plot alone. In hindsight, a time while the solution was changing faster would have been a better choice for the test case.

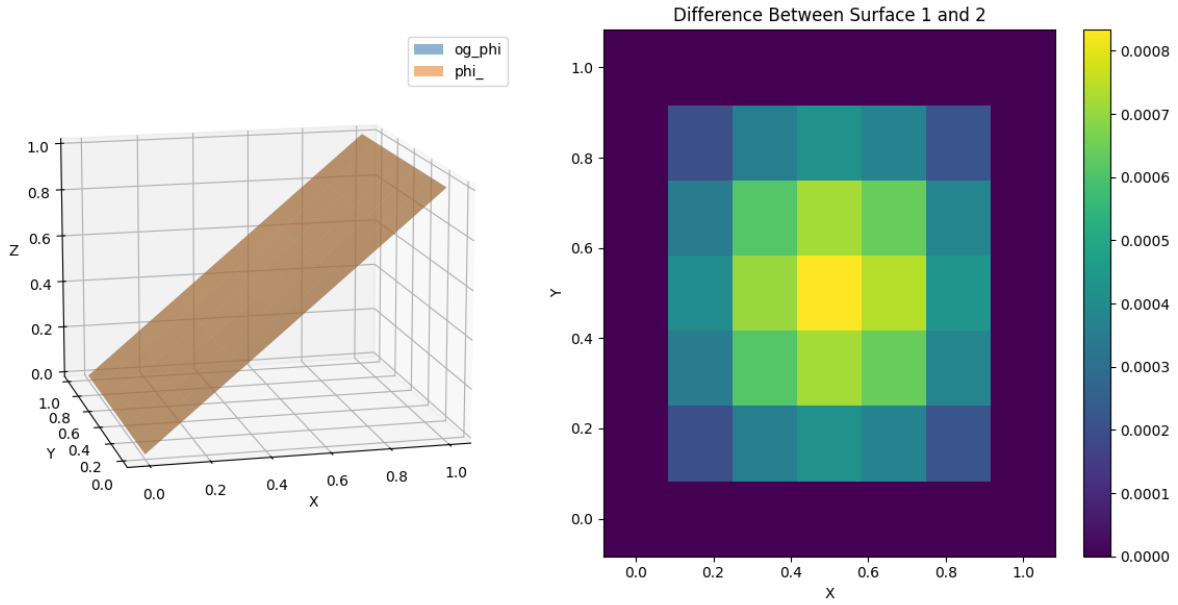

Figure 3: Comparison that showed us we were one time step off. The structure of the difference between the two surfaces is consistent with the time evolution of the surface.

5 C++ Source Code

5.1 dump.h

```
//  ================================================================================

//  ||
    ||
//  ||              transientDiffusion
    ||
//  ||              ------------------------------------------------------
    ||
//  ||              T R A N S I E N T D I F F U S I O N
    ||
//  ||
    ||
//  ||              D E M O N S T R A T I O N   C O D E
    ||
//  ||              ------------------------------------------------------
    ||
//  ||
    ||
//  ||       Developed by: Scott R. Runnels, Ph.D.
    ||
//  ||                     University of Colorado Boulder
    ||
//  ||
    ||
//  ||                For: CU Boulder CSCI 4576/5576 and associated labs
    ||
//  ||
    ||
//  ||           Copyright 2020 Scott Runnels
    ||
//  ||
    ||
//  ||                     Not for distribution or use outside of the
    ||
//  ||                     this course.
    ||
//  ||
    ||
//  ================================================================================


//  ================================================================================

//  ||
    ||
//  ||               R e s t a r t   C a p a b i l i t y
    ||
```

```
25 //  ||
       ||
26 //
       ================================================================================


27
28
29 //  ==
30 //  ||
31 //  ||  Write Restart File
32 //  ||
33 //  ==
34
35 void writeRestart(double &time             ,
36         double &dt               ,
37         double &timeSinceLastPlot,
38         int    &nCellx           ,
39         int    &nCelly           ,
40         int    &count            ,
41         string &solver           ,
42         mpiInfo &myMPI           )
43 {
44    VD headerDbls;  headerDbls.resize(4);   for ( int i = 0 ;  i < headerDbls.size
       () ; ++i ) headerDbls[i] = 0.;
45    VI headerInts;  headerInts.resize(4);   for ( int i = 0 ;  i < headerInts.size
       () ; ++i ) headerInts[i] = 0 ;
46
47    //
       ------------------------------------------------------------------------------------------

48    // Store simulation setup variables and a few scalar state variables into
       arrays for the dump:
49    //
       ------------------------------------------------------------------------------------------


50
51    // (A) Double-precision variables to be saved in the dump:
52    //
53    //    [0] Current simulation time
54    //    [1] Time step
55    //    [2] Time since last plot was written
56
57    headerDbls[0] = time;
58    headerDbls[1] = dt ;
59    headerDbls[2] = timeSinceLastPlot ;
60
61    // (B) Integers variables to be saved in the dump:
62    //
63    //    [0] Total number of cells (globally) in the x-direction for the entire
       physical problem
64    //    [1] Same for y-direction
65    //    [2] An integer reflection the choice of linear solver
66    //    [3] The number of plots files already written
67
68    headerInts[0] = nCellx * myMPI.nPEx;
```

```
69    headerInts[1] = nCelly * myMPI.nPEy;
70
71    if ( solver == "jacobi" ) headerInts[2] = 1;
72    if ( solver == "cg"     ) headerInts[2] = 2;
73
74    headerInts[3] = count;
75
76    // Write the dump
77
78    write_mpiio_dump(headerDbls, headerInts, myMPI );
79 }
80
81
82
83 //  ==
84 //  ||
85 //  ||  Read Restart File
86 //  ||
87 //  ==
88
89 void readRestart(double &time             ,
90        double &dt                 ,
91        double &timeSinceLastPlot,
92        int    &nCellx             ,
93        int    &nCelly             ,
94        int    &count              ,
95        string &solver             ,
96        mpiInfo &myMPI             )
97 {
98    VD headerDbls;  headerDbls.resize(4);   for ( int i = 0 ;  i < headerDbls.size
      () ; ++i ) headerDbls[i] = 0.;
99    VI headerInts;  headerInts.resize(4);   for ( int i = 0 ;  i < headerInts.size
      () ; ++i ) headerInts[i] = 0 ;
100
101   // Read the dump
102
103   read_mpiio_dump(headerDbls, headerInts, myMPI );
104
105   //
      ------------------------------------------------------------------------------------------
106   // Retrieve simulation setup variables and a few scalar state variables from
      arrays
107   //
      ------------------------------------------------------------------------------------------
108
109   // (A) Double-precision variables to be saved in the dump:
110   //
111   //    [0] Current simulation time
112   //    [1] Time step
113   //    [2] Time since last plot was written
114
115   time              = headerDbls[0] ;
116   dt                = headerDbls[1];
```

```
117     timeSinceLastPlot = headerDbls[2];
118
119
120     // (B) Integers variables to be saved in the dump:
121     //
122     //     [0] Total number of cells (globally) in the x-direction for the entire
        physical problem
123     //     [1] Same for y-direction
124     //     [2] An integer reflection the choice of linear solver
125     //     [3] The number of plots files already written
126
127     nCellx = headerInts[0] / myMPI.nPEx          ;
128     nCelly = headerInts[1]/myMPI.nPEy            ;
129
130     if ( headerInts[2] == 1 ) solver = "jacobi" ;
131     if ( headerInts[2] == 2 ) solver = "cg"     ;
132
133     count  = headerInts[3] ;
134 }
135
136
137
138 //
        ================================================================================
139 //  ||
        ||
140 //  ||  U t i l i t y :     S i n g l e - V a l u e   I / O   R o u t i n e s
        ||
141 //  ||
        ||
142 //
        ================================================================================
143
144
145
146 void write_mpiio_int   ( MPI_File fh ,   int  &val, MPI_Offset &offset ){
        MPI_Status status; MPI_File_write_at(fh, offset, &val, 1, MPI_INT   , &status);
         offset += sizeof(int)   ;}
147 void write_mpiio_double( MPI_File fh , double &val, MPI_Offset &offset ){
        MPI_Status status; MPI_File_write_at(fh, offset, &val, 1, MPI_DOUBLE, &status);
         offset += sizeof(double);}
148 void read_mpiio_int    ( MPI_File fh ,   int  &val, MPI_Offset &offset ){
        MPI_Status status; MPI_File_read_at (fh, offset, &val, 1, MPI_INT   , &status);
         offset += sizeof(int)   ;}
149 void read_mpiio_double ( MPI_File fh , double &val, MPI_Offset &offset ){
        MPI_Status status; MPI_File_read_at (fh, offset, &val, 1, MPI_DOUBLE, &status);
         offset += sizeof(double);}
150
151
152
153 //
        ================================================================================
```

```
154 //  ||
        ||
155 //  ||  M P I I O   D u m p   W r i t e r
        ||
156 //  ||
        ||
157 //
        ================================================================================


158
159 void write_mpiio_dump(VD &headerDbls , VI &headerInts , mpiInfo &myMPI)
160 {
161   int myPE    = myMPI.myPE;
162   int iPE     = myMPI.iPE;
163   int jPE     = myMPI.jPE;
164   int nPEx    = myMPI.nPEx;
165   int nPEy    = myMPI.nPEy;
166   int numPE   = myMPI.numPE;
167   int nTotalx = nRealx + 2;
168   int nTotaly = nRealy + 2;
169
170   // -
171   // |
172   // | Open the file to be written
173   // |
174
175   // -
176
177
178
179   MPI_File fh;
180   MPI_File_open(MPI_COMM_WORLD, "phi_dump.bin", MPI_MODE_CREATE | MPI_MODE_WRONLY,
        MPI_INFO_NULL, &fh);
181
182   // -
183   // |
184   // | Write header information (mostly, simulation setup variables)
185   // |
186   // -
187
188   MPI_Offset offset = 0;
189   MPI_Status  status;
190
191   if ( myPE == 0 )
192     {
193       for ( int i = 0 ; i < headerDbls.size() ; ++i ) write_mpiio_double( fh ,
      headerDbls[i] , offset);
194       for ( int i = 0 ; i < headerInts.size() ; ++i ) write_mpiio_int    ( fh ,
      headerInts[i] , offset);
195     }
196   MPI_Bcast( &offset , 1 , MPI_INT , 0 ,  MPI_COMM_WORLD);
197   MPI_Barrier(MPI_COMM_WORLD);
198
199
200   // -
```

```
  // |
  // | Allocate and populate A with phi
  // |
  // -

float **A = Array2D_float(nTotalx, nTotaly);
  iLOOP
    jLOOP
  {
    int p = pid(i,j);
    A[i][j] = phi[p];
  }

  // -
  // |
  // | Create MPI derived data type that will be used to represent the real nodes
    in the grid.
  // |
  // -

  MPI_Datatype myRealNodes;
  int idxStartThisPE  [2] = { 1         , 1         };  // Index coordinates of the
    sub-array inside this PE's array, A
  int AsizeThisPE     [2] = { nTotalx   , nTotaly   };  // Size of the A array on
    this PE
  int sub_AsizeThisPE [2] = { nRealx    , nRealy    };  // Size of the A-sub-array
    on this PE

  // Adjust size and start if not at the left / bottom

  if ( iPE > 0 ) { ++idxStartThisPE[0];   --sub_AsizeThisPE[0];   }
  if ( jPE > 0 ) { ++idxStartThisPE[1];   --sub_AsizeThisPE[1];   }

  // Create and commit

  MPI_Type_create_subarray(2, AsizeThisPE, sub_AsizeThisPE, idxStartThisPE,
    MPI_ORDER_C, MPI_FLOAT, &myRealNodes);
  MPI_Type_commit(&myRealNodes);

  // -
  // |
  // | Create a second derived type that will be used to describe the whole array.
  // |
  // -

  MPI_Datatype myPartOfGlobal;
  int idxStartInGlobal [2] = { iPE * nRealx   , jPE * nRealy   };          // Index
    cordinates of the sub-array inside the global array
  int AsizeGlobal      [2] = { nPEx*(nRealx-1)+1 , nPEy*(nRealy-1)+1 };  // Size
    of the global array

  // Adjust start index if not at the left/bottom edge

  if ( iPE > 0 ) {      idxStartInGlobal[0] -=  1 * (iPE-1);    }
  if ( jPE > 0 ) {      idxStartInGlobal[1] -=  1 * (jPE-1);    }
```

```
249
250   // Create and commit
251
252   MPI_Type_create_subarray(2, AsizeGlobal, sub_AsizeThisPE, idxStartInGlobal,
        MPI_ORDER_C, MPI_FLOAT, &myPartOfGlobal);
253   MPI_Type_commit(&myPartOfGlobal);
254
255   // -
256   // |
257   // | Set the "view" of the file from this PE's perspective, i.e., where and how
        this PE should write data
258   // |
259   // -
260   MPI_File_set_view (fh, offset, MPI_FLOAT, myPartOfGlobal, "native",
        MPI_INFO_NULL);
261
262   // -
263   // |
264   // | Perform the collective write operation and clean up
265   // |
266   // -
267
268   MPI_File_write_all(fh, &A[0][0], 1, myRealNodes, MPI_STATUS_IGNORE);
269
270   MPI_File_close(&fh);
271   free(A[0]);
272   free(A);
273
274   MPI_Type_free(&myPartOfGlobal);
275   MPI_Type_free(&myRealNodes);
276
277
278 }
279
280
281 //    ============================================================================
282 //   ||
      ||
283 //   ||  M P I I O   D u m p   R e a d e r
      ||
284 //   ||
      ||
285 //    ============================================================================
286
287 void read_mpiio_dump(VD &headerDbls , VI &headerInts , mpiInfo &myMPI)
288 {
289   int myPE    = myMPI.myPE;
290   int iPE     = myMPI.iPE;
291   int jPE     = myMPI.jPE;
292   int nPEx    = myMPI.nPEx;
293   int nPEy    = myMPI.nPEy;
```

```
294  int numPE   = myMPI.numPE;
295
296  int nTotalx = nRealx + 2;
297  int nTotaly = nRealy + 2;
298
299  // -
300  // |
301  // |  Open the file to be read
302  // |
303  // -
304
305  MPI_File fh;
306  cout << "Check 0" << endl;
307  MPI_File_open(MPI_COMM_WORLD, "phi_dump.bin", MPI_MODE_RDONLY, MPI_INFO_NULL, &
       fh);
308
309  // -
310  // |
311  // |  Read header information
312  // |
313  // -
314
315    MPI_Offset offset = 0;
316    MPI_Status  status;
317
318    for ( int i = 0 ; i < headerDbls.size() ; ++i ) read_mpiio_double( fh ,
       headerDbls[i] , offset);
319    for ( int i = 0 ; i < headerInts.size() ; ++i ) read_mpiio_int   ( fh ,
       headerInts[i] , offset);
320
321    MPI_Barrier(MPI_COMM_WORLD);
322
323  // -
324  // |
325  // | Allocate and populate A with default values; A will be used to read phi
326  // |
327  // -
328
329  float **A = Array2D_float(nTotalx, nTotaly);
330  iLOOP jLOOP { A[i][j] = 0.; }
331
332  // -
333  // |
334  // | Create MPI derived data type that will be used to represent the real nodes
       in the grid.
335  // |
336  // -
337
338  MPI_Datatype myRealNodes;
339  int idxStartThisPE  [2] = { 1        , 1        };  // Index coordinates of the
       sub-array inside this PE's array, A
340  int AsizeThisPE     [2] = { nTotalx  , nTotaly  };  // Size of the A array on
       this PE
341  int sub_AsizeThisPE [2] = { nRealx   , nRealy   };  // Size of the A-sub-array
       on this PE
```

```
342
343   MPI_Type_create_subarray(2, AsizeThisPE, sub_AsizeThisPE, idxStartThisPE,
        MPI_ORDER_C, MPI_FLOAT, &myRealNodes);
344   MPI_Type_commit(&myRealNodes);
345
346   // -
347   // |
348   // | Create a second derived type that will be used to describe the whole array.
349   // |
350   // -
351
352   MPI_Datatype myPartOfGlobal;
353   int idxStartInGlobal [2] = { iPE * nRealx  , jPE * nRealy  };  // Index
        coordinates of the sub-array inside the global array
354   int AsizeGlobal      [2] = { nPEx*(nRealx-1)+1 , nPEy*(nRealy-1)+1 };  // Size
        of the global array
355
356   // Adjust start index if not at the left/bottom edge
357
358   if ( iPE > 0 ) idxStartInGlobal[0] = iPE*(nRealx-1);
359   if ( jPE > 0 ) idxStartInGlobal[1] = jPE*(nRealy-1);
360
361   // Create and commit
362
363   MPI_Type_create_subarray(2, AsizeGlobal, sub_AsizeThisPE, idxStartInGlobal,
        MPI_ORDER_C, MPI_FLOAT, &myPartOfGlobal);
364   MPI_Type_commit(&myPartOfGlobal);
365
366   // -
367   // |
368   // | Set the "view" of the file from this PE's perspective, i.e., where and how
        this PE should write data
369   // |
370   // -
371
372   MPI_File_set_view (fh, offset, MPI_FLOAT, myPartOfGlobal, "native",
        MPI_INFO_NULL);
373
374   // -
375   // |
376   // | Perform the collective read operation and clean up
377   // |
378   // -
379
380   MPI_File_read_all(fh, &A[0][0], 1, myRealNodes, MPI_STATUS_IGNORE);
381
382   // Store into phi
383
384   iLOOP jLOOP { int p = pid(i,j);  phi[p] = A[i][j] ;  }
385
386   // Cleanup
387
388   MPI_File_close(&fh);
389   free(A[0]);
390   free(A);
```

```
391
392   MPI_Type_free(&myPartOfGlobal);
393   MPI_Type_free(&myRealNodes);
394
395 }
```

5.2 transientDiffusion.cpp

```
1  //
       ================================================================================
2  //   ||
       ||
3  //   ||               transientDiffusion
       ||
4  //   ||               ------------------------------------------------------
       ||
5  //   ||               T R A N S I E N T D I F F U S I O N
       ||
6  //   ||
       ||
7  //   ||               D E M O N S T R A T I O N   C O D E
       ||
8  //   ||               ------------------------------------------------------
       ||
9  //   ||
       ||
10 //   ||        Developed by: Scott R. Runnels, Ph.D.
       ||
11 //   ||                      University of Colorado Boulder
       ||
12 //   ||
       ||
13 //   ||                 For: CU Boulder CSCI 4576/5576 and associated labs
       ||
14 //   ||
       ||
15 //   ||            Copyright 2020 Scott Runnels
       ||
16 //   ||
       ||
17 //   ||                      Not for distribution or use outside of the
       ||
18 //   ||                      this course.
       ||
19 //   ||
       ||
20 //
       ================================================================================
21
22 #include "mpi.h"
23 #include "transientDiffusion.h"
24 #include "mpiInfo.h"
25 #include "LaplacianOnGrid.h"
26
```

```
27
28 //  ==
29 //  ||
30 //  ||
31 //  ||  Main Program
32 //  ||
33 //  ||
34 //  ==
35
36 int main(int argc, char *argv[])
37 {
38
39    mpiInfo myMPI;
40    MPI_Init     (&argc          , &argv       );
41    MPI_Comm_size(MPI_COMM_WORLD, &myMPI.numPE);
42    MPI_Comm_rank(MPI_COMM_WORLD,&myMPI.myPE  );
43
44
45    int nPEx, nPEy, nCellx, nCelly;
46    double tEnd, dt, tPlot;
47    string solver;
48    int restart = 0;
49
50     if ( myMPI.myPE == 0 )
51       {
52        cout << "\n";
53        cout << "-----------------------------------------------------\n";
54        cout << "\n";
55        cout << " S O L V E R S                                       \n";
56        cout << " D E M O   C O D E                                   \n";
57        cout << "\n";
58        cout << " Running on " << myMPI.numPE << " processors \n";
59        cout << "\n";
60        cout << "-----------------------------------------------------\n";
61        cout << "\n";
62       }
63
64     solver = "none";
65
66    for (int count =  0 ; count < argc; ++count)
67      {
68        if ( !strcmp(argv[count],"-nPEx"    ) ) nPEx    = atoi(argv[count+1]);
69        if ( !strcmp(argv[count],"-nPEy"    ) ) nPEy    = atoi(argv[count+1]);
70        if ( !strcmp(argv[count],"-nCellx"  ) ) nCellx  = atoi(argv[count+1]);
71        if ( !strcmp(argv[count],"-nCelly"  ) ) nCelly  = atoi(argv[count+1]);
72        if ( !strcmp(argv[count],"-solver"  ) ) solver  =      argv[count+1] ;
73        if ( !strcmp(argv[count],"-tEnd"    ) ) tEnd    = atof(argv[count+1]);
74        if ( !strcmp(argv[count],"-dt"      ) ) dt      = atof(argv[count+1]);
75        if ( !strcmp(argv[count],"-tPlot"   ) ) tPlot   = atof(argv[count+1]);
76        if ( !strcmp(argv[count],"-restart" ) ) restart = atoi(argv[count+1]);
77      }
78
79    // -
80    // |
81    // | Echo outputs
```

```
  // |
  // -

  if ( myMPI.myPE == 0 )
    {
      cout << endl;
      cout << "Input Summary: " << endl;
      cout << "-------------------------- " << endl;
      cout << "No. PE   in  x-direction: " << nPEx    << endl;
      cout << "             y-direction: " << nPEy    << endl;
      cout << "No. Cells in x-direction: " << nCellx  << endl;
      cout << "             y-direction: " << nCelly  << endl;
      cout << "Linear solver           : " << solver  << endl;
      cout << "End Time                : " << tEnd    << endl;
      cout << "Time Step               : " << dt      << endl;
      cout << "Plot Interval           : " << tPlot   << endl;
      cout << "This is a restart (1/0) : " << restart << endl;
      cout << endl;
    }

  myMPI.GridDecomposition(nPEx,nPEy,nCellx,nCelly);

  // -
  // |
  // | Parallel Grid Generation and Laplace Solver
  // |
  // -

  double totalLength = 1.;
  double eachPElength_x = totalLength / nPEx;
  double eachPElength_y = totalLength / nPEy;

  double x0 = eachPElength_x * myMPI.iPE;   double x1 = x0 + eachPElength_x;
  double y0 = eachPElength_y * myMPI.jPE;   double y1 = y0 + eachPElength_y;

  LaplacianOnGrid MESH(x0,x1,y0,y1,nCellx,nCelly, myMPI );

  // -
  // |
  // | Time Marching Loop
  // |
  // -

  double tStart = 0.;

  double timeSinceLastPlot = 0.;
  int    latestIterCount;
  int    count = 0;

  MPI_Barrier(MPI_COMM_WORLD);

  if ( restart )
    {
      MPI_Barrier(MPI_COMM_WORLD);
      MESH.readRestart(tStart, dt, timeSinceLastPlot, nCellx, nCelly, count,
```

```
    solver, myMPI);
137       MESH.plot( "load_phi"  , MESH.phi ,     count , myMPI      );
138       // Account for the fact that values are saved at the
139       // end of the loop (i.e. post update for that loop)
140       tStart += dt;
141     }
142
143   for ( double time = tStart ; time <= tEnd ; time += dt )
144     {
145
146       MPI_Barrier(MPI_COMM_WORLD);
147
148       MESH.FormLS(myMPI, dt);
149
150       MPI_Barrier(MPI_COMM_WORLD);
151
152       if      ( solver == "jacobi" ) MESH.Jacobi(MESH.Acoef , MESH.b , MESH.
    phiNew , myMPI );
153       else if ( solver == "cg"     ) MESH.CG    (MESH.Acoef , MESH.b , MESH.
    phiNew , myMPI );
154       else                           FatalError("Solver " + solver + " not found.
    ");
155
156       MESH.Transient_UpdatePhi();
157
158       // Plot / Restart
159
160       timeSinceLastPlot += dt;
161
162       if ( timeSinceLastPlot >= tPlot )
163    {
164      if ( myMPI.myPE == 0 ) cout << "Plotting at time = " << time << " Plot ID =
    " << count << endl << std::flush;
165
166      MESH.plot( "phi"  , MESH.phi ,     count , myMPI      );
167
168      timeSinceLastPlot = 0.;
169
170      MPI_Barrier(MPI_COMM_WORLD);
171      MESH.writeRestart(time, dt, timeSinceLastPlot, nCellx, nCelly, count, solver
    , myMPI);
172      ++count;
173    }
174
175     }
176
177   if ( myMPI.myPE == 0 ) printf("Execution Completed Successfully\n");
178
179   MPI_Finalize();
180   return 0;
181
182 }
```

6 Python + YAML Source Code

6.1 keepRunning_actual.yml

```
1
2  EXECUTABLE:
3
4     mpirun    : 'mpirun -n 4'
5     pathToExe : './transientDiffusion'
6
7  ARGUMENTS:
8
9     initial  : ' -nPEx 2 -nPEy 2 -nCellx  40 -nCelly 40 -solver jacobi -tEnd 3   -
      dt .001 -tPlot .1           > tmp'
10    restart  : ' -nPEx 2 -nPEy 2 -nCellx  40 -nCelly 40 -solver jacobi -tEnd 3   -
      dt .001 -tPlot .1 -restart 1 > tmp'
11
12 COMPLETION:
13
14    ttyOutput    : 'tmp'
15    completionStr: 'Execution Completed Successfully'
```

6.2 keepRunning_py3.py

```
1  #!/usr/bin/python3
2  import os
3  import sys
4  import getopt
5  import glob
6  import yaml
7  import re
8  from datetime import datetime
9  from io import StringIO
10 import re
11 import time
12
13 #
       ================================================================================================
14 #
       ================================================================================================
15 #
16 #                                      U T I L I T I E S
17 #
18 #
       ================================================================================================
19 #
       ================================================================================================
20
21
22 # Prints a user help message:
23
```

```
24 def help():
25
26     print("")
27     print("")
28     print("This script keeps a command running, restarting it after")
29     print("it has been killed by a system monitor.")
30     print("")
31     print("")
32
33 # Packs a string with blanks to be a specified width:
34
35 def pack(string):
36     ans = string
37     for i in range(len(string),88): ans += ' '
38     return ans
39
40 # Prints a nice banner with a message inside box:
41
42 def bannerDisplay(header,message):
43     tmp = message.split('\n')
44     print
45     print
46     print ("
       ================================================================================================
       ")
47     print ("||
                            ||")
48     print ("||
                            ||")
49     print ("|| " + pack(header) + ' ||')
50     print ("||
                            ||")
51     for t in tmp:
52         print ('|| '+ pack(t) + ' ||' )
53     print ("||
                            ||")
54     print ("
       ================================================================================================
       ")
55     print
56
57 def FatalError(msg):
58     bannerDisplay('*** Fatal Error *** in keepRunning.py',msg)
59     sys.exit()
60
61
62
63 #
       ================================================================================================

64 #
       ================================================================================================

65 #
66 #                                   M A I N   C O D E
```

```
#
#
    ================================================================================================

#
    ================================================================================================



# ==
# ||
# || runDone:  Returns True if it finds the string indicating successfull
    completion in the ttyFile
# ||
# ==


def runDone(ttyFile,completionIndicator):

    try:
        f = open(ttyFile,'r')
    except:
        print("tty file ("+ttyFile+")not found.  Assuming the job is not still
    running...")
        return True

    for line in f:
        if completionIndicator in line:
            f.close()
            return True

    f.close()
    return False



# ==
# ||
# || Main Program
# ||
# ==

def keepRunning(argv):

    # -
    # |
    # | Command-line arguments
    # |
    # -

    yamlFile = ""

    try:
        opts, args = getopt.getopt(argv,"h f:")

```

```
    except:
        fatalError('Error in command-line arguments.  Try -h to see help.')

    for opt, arg in opts:

        if opt == '-h':
            help()
            sys.exit()

        elif opt == "-f":
            yamlFile  = arg

    if yamlFile == '' : FatalError("You must provide a yaml yaml file")

    # -
    # |
    # | Read input
    # |
    # -

    stream = open(yamlFile, 'r')
    yamlDic = yaml.load(stream,Loader=yaml.Loader)

    mpirun         = yamlDic['EXECUTABLE']['mpirun'        ]
    exe            = yamlDic['EXECUTABLE']['pathToExe'     ]
    initialRunArgs = yamlDic['ARGUMENTS' ]['initial'       ]
    restartRunArgs = yamlDic['ARGUMENTS' ]['restart'       ]
    ttyOutput      = yamlDic['COMPLETION']['ttyOutput'     ]
    completionStr  = yamlDic['COMPLETION']['completionStr' ]

    # -
    # |
    # | Construct the run and restart commands
    # |
    # -

    runCommand     = mpirun + ' ' + exe + ' ' + initialRunArgs
    restartCommand = mpirun + ' ' + exe + ' ' + restartRunArgs

    print('Running ' + runCommand)
    os.system(runCommand + ' & ')

    time.sleep(1)

    # -
    # |
    # | Infinite loop which constantly checks to see if the exe should be
    restarted
    # |
    # -

    count = 0
    while(True):

        os.system('clear')
```

```
170
171         count += 1
172
173         print
174         print('-----------------------------------------------------------------'
    )
175         print
176         print('Iteration          : ' + str(count))
177         print('Checking on        : ' + exe )
178         print
179
180         userName = '**TO-DO**'
181         jobName = exe
182
183         psCommand = "ps -elf | grep " + userName + " | grep " + jobName + " | grep
     -v 'grep' "
184         jobStatus = os.popen(psCommand).read()
185
186         print ('Searching for this job: ' + jobName )
187         print ('With this command     : ' + psCommand)
188         print ('Under user name       : ' + userName)
189         print ('Found this record  : ' + jobStatus)
190
191
192         if len(jobStatus) > 0:
193             print ('It is still running...')
194         else:
195             print ('It is no longer running.   Checking its tty output to see if
    it finished.')
196             if runDone(ttyOutput,completionStr):
197                 print ('It did finish.  This script (keepRunning.py) is now
    exiting.')
198                 return 0
199             else:
200                 print ('It is no longer running.   Restarting it now...')
201                 os.system(restartCommand + ' & ')
202
203         time.sleep(1)
204
205
206
207
208 if __name__ == "__main__":
209     keepRunning(sys.argv[1:])
```

6.3 mySlurm_py3.py

```
1 #!/usr/bin/python3
2 import os
3 import sys
4 import getopt
5 import glob
6 import yaml
7 import re
8 from datetime import datetime
9 from io import StringIO
```

```
10 import re
11 import time
12
13 #    ============================================================================================
14 #    ============================================================================================
15 #
16 #                                    U T I L I T I E S
17 #
18 #    ============================================================================================
19 #    ============================================================================================
20
21
22 def help():
23
24     print("")
25     print("")
26     print ("This script is to be run in the background.  It mimics ")
27     print ("a batch run system with queues.  Its purpose is to enable")
28     print ("the development of automatic restart capabilities without")
29     print ("having to work on a system with those capabilities.")
30     print("")
31     print("")
32
33 def pack(string):
34     ans = string
35     for i in range(len(string),88): ans += ' '
36     return ans
37
38 def bannerDisplay(header,message):
39     tmp = message.split('\n')
40     print
41     print
42     print ("
       ================================================================================
       ")
43     print ("||
                            ||")
44     print ("||
                            ||")
45     print ("|| " + pack(header) + ' ||')
46     print ("||
                            ||")
47     for t in tmp:
48         print ('|| '+ pack(t) + ' ||' )
49     print ("||
                            ||")
50     print ("
```

```
    ================================================================================
    ")
    print

def FatalError(msg):
    bannerDisplay('*** Fatal Error *** in mySlurm.py',msg)
    sys.exit()


# If string, s = hello(" hi " , " there " )
# this routine will return hi and there.

def findSubstrings(s):
    substring = '"'
    matches = re.finditer(substring,s)

    # List containing the indices of the double quote sign
    quotes = [match.start() for match in matches]

    ans1 =  s[quotes[0]+1:quotes[1]]
    ans2 =  s[quotes[2]+1:quotes[3]]

    return ans1, ans2


def replacePhrase(input_str,delimiter1,delimiter2,target_str):
    idx1 = input_str.find(delimiter1)
    idx2 = input_str.find(delimiter2)

    tmp = input_str[idx1+1:idx2]
    result = input_str.replace(tmp,target_str)
    return result

def timeInSeconds(time_str):

    d2s = 24*3600
    h2s = 3600
    m2s = 60

    tmp = time_str.split(':')

    if len(tmp) == 1:     return int(time_str)
    if len(tmp) == 2:     return int(tmp[0]) * m2s  + int(tmp[1])
    if len(tmp) == 3:     return int(tmp[1]) * h2s  + int(tmp[1]) * m2s + int(tmp
    [0])
    if len(tmp) == 4:     return int(tmp[2]) * d2s  + int(tmp[1]) * h2s + int(tmp
    [1]) * m2s + int(tmp[0])


#
    ================================================================================

#
    ================================================================================
```

```
#
#                                   M A I N   C O D E
#
#    ================================================================================================

#    ================================================================================================



# ==
# ||
# || Main Program
# ||
# ==

def mySlurm(argv):

    # -
    # |
    # | Command-line arguments
    # |
    # -

    inputFile = ""
    maxTime   = -999.

    try:
        opts, args = getopt.getopt(argv,"h f: t:",["dir="])

    except:
        fatalError('Error in command-line arguments.  Try -h to see help.')

    for opt, arg in opts:

        if opt == '-h':
            help()
            sys.exit()

        elif opt == "-f":
            inputFile  = arg

        elif opt == "-t":
            maxTime = float(arg)

        elif opt == "--dir":
            srcDir   = arg

    if maxTime < 0.: FatalError("You must provide a max time in seconds.")


    # -
    # |
```

```
    # | Get ps -elf output
    # |
    # -

    count = 0

    while (True):
        os.system('clear')

        count += 1

        userName = "**TO-DO**"
        jobName = "./transientDiffusion"

        print
        print ('Iteration             : ' + str(count))
        print ('Searching for this job: ' + jobName )
        print ('Under user name       : ' + userName)

        psCommand = "ps -elf | grep " + userName + " | grep " + jobName + " | grep -v 'grep' "
        jobStatus = os.popen(psCommand).read()

        if len(jobStatus) <= 0:
            print ('Job not found, nothing to do.')

        if len(jobStatus) > 0:
            jobStatus = jobStatus.replace('\n','')

            statusBreakdown = re.split(' +', jobStatus)
            jobID           = statusBreakdown[3]

            psCommand = "ps -p " + jobID + " -o etime | grep -v ELAPSED"
            psElapsed = os.popen(psCommand).read()
            psElapsed = psElapsed.replace('\n','')
            psSeconds = timeInSeconds(psElapsed)

            print ('Found this record     : ' + jobStatus)
            print ('Seconds running (ps)  : ' + str(psSeconds))
            print ('Max time allowed      : ' + str(maxTime))
            print

            if int(psSeconds) > int(maxTime):
                killCommand = 'kill -9 ' + str(jobID)
                print ('Max Time Exceeded:  Killing the job with: ' + killCommand)
                os.system(killCommand)

        time.sleep(2)


if __name__ == "__main__":
    mySlurm(sys.argv[1:])
```

6.4 Python Plotter

```python
import matplotlib.pyplot as plt
from matplotlib import cm
import matplotlib.colors as mcolors
import matplotlib.animation as animation
import numpy as np
import glob


def points_to_grid(pts: np.array) -> tuple[np.array]:
    """
    Turns a set of x,y,z points into 2D arrays
    X,Y,Z which are the x,y,z values at every
    point in a 2D grid.

    Args:
        points (np.array): npoints rows, 4 columns.
                           columns are in x, y, z order.
                           The fourth column is the PE
                           that this came from.

    Returns:
        X, Y, Z    (np.arrays) with ny rows, nx columns.
    """
    # Create a mapping from unique
    # x and y vals to columns/row numbers
    xvals = np.unique(pts[:, 0])
    yvals = np.unique(pts[:, 1])

    xindx = {}
    for i, x in enumerate(xvals):
        xindx[x] = i
    yindx = {}
    for i, y in enumerate(yvals):
        yindx[y] = i

    # Fill in the Z grid
    X, Y = np.meshgrid(xvals, yvals)
    Z = np.zeros_like(X)

    for pt in pts:
        ix = xindx[pt[0]]
        iy = yindx[pt[1]]
        Z[iy, ix] = pt[2]

    return X, Y, Z


def load_points(root: str, timestamp: str = "", return_fname: bool = False) ->
    list[np.array]:
    """
    Loads the points from all files with the given root

    Args:
        root (str): root of files to load
        timestamp (str): timestamp assumed to be write before .plt
```

```
                          for plotting a single timestep in a transient sim.
        return_fnames (bool): return a dictionary that maps filename to array
                              instead of just all the points.

    Returns:
        list[np.array]: list where each entry is an np.array
                        with columns x, y, z, and file number
    """

    dfiles = sorted(glob.glob(root))
    # Filter for .plt
    dfiles = [x for x in dfiles if ".plt" in x]

    # Filter for timestamp
    if timestamp != "":
        dfiles = [x for x in dfiles if f"{timestamp}.plt" in x]
    if return_fname:
        pts = {}
    else:
        pts = []
    for i, f in enumerate(dfiles):
        from_file = np.loadtxt(f)
        data_with_pe = np.hstack((from_file, i*np.ones((from_file.shape[0], 1))))
        if return_fname:
            pts[f] = data_with_pe
        else:
            pts.append(data_with_pe)

    return pts


def single_surface_plot(root: str, outfile: str, timestamp: str = "") -> plt.
    figure:
    """
    Generates a single surface plot of the files desired

    Args:
        root (str): files to grab to plot
        outfile (str): file to write figure to
        timestamp (str): timestamp assumed to be write before .plt
                         for plotting a single timestep in a transient sim.

    Returns:
        plt.figure: handle to figure object
    """

    # Load data
    points = load_points(root+"*", timestamp)

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    for i, pts in enumerate(points):
        X, Y, Z = points_to_grid(pts)
```

```
        surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, linewidth=1,
                               alpha=0.8, label=f"PE {i}")
        surf._edgecolors2d = surf._edgecolor3d
        surf._facecolors2d = surf._facecolor3d

    ax.legend()

    for pts in points:
        ax.scatter(pts[:, 0], pts[:, 1], pts[:, 2], marker=".", c="k", s=0.5,
                   alpha=0.25)

    if outfile != "":
        plt.savefig(outfile)

    return fig


def multi_surface_plot(flist: list[str], outfile: str, timestamp: str = "") -> plt
    .figure:
    """
    Generates a surface plot for each set of files

    Args:
        flist (str): files to grab to plot
        outfile (str): file to write figure to

    Returns:
        plt.figure: handle to figure object
    """

    # Make figure
    fig, ax = plt.subplots(ncols=len(flist), subplot_kw={"projection": "3d"})
    fig.set_size_inches(6*len(flist), 6)

    for i, root in enumerate(flist):
        # Load data
        points = load_points(root+"*")

        ax[i].set_xlabel("X")
        ax[i].set_ylabel("Y")
        ax[i].set_zlabel("Z")
        for iPE, pts in enumerate(points):
            X, Y, Z = points_to_grid(pts)
            surf = ax[i].plot_surface(X, Y, Z, rstride=1, cstride=1, linewidth=1,
                                      alpha=0.8, label=f"PE {iPE}")
            surf._edgecolors2d = surf._edgecolor3d
            surf._facecolors2d = surf._facecolor3d

        # ax[i].legend()
        ax[i].set_title(root)

        for pts in points:
            ax[i].scatter(pts[:, 0], pts[:, 1], pts[:, 2], marker=".", c="k",
                          s=0.5, alpha=0.25)

```

```
    if outfile != "":
        plt.savefig(outfile)

    return fig


def double_surface_plot(root: str, root2: str, outfile: str, timestamp: str = "")
    -> plt.figure:
    """
    Generates a surface plot comparing two surfaces.
    Plots them both in the same axes, and shows a residual
    as well.

    Args:
        root (str): files to grab to plot
        root2 (str): other set of files to grab to plot
        outfile (str): file to write figure to
        timestamp (str): timestamp assumed to be write before .plt
                         for plotting a single timestep in a transient sim.

    Returns:
        plt.figure: handle to figure object
    """

    # Load data
    points = np.concatenate(load_points(root+"*", timestamp))
    points2 = np.concatenate(load_points(root2+"*", timestamp))

    fig = plt.figure(figsize=(12,6))
    ax = fig.add_subplot(1, 2, 1, projection='3d')

    # Plot surfaces
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    labels = [root, root2]
    Zvals = []
    for i, pts in enumerate([points, points2]):
        X, Y, Z = points_to_grid(pts)
        Zvals.append(Z)
        surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                               linewidth=1, alpha=0.5, label=labels[i])
        surf._edgecolors2d = surf._edgecolor3d
        surf._facecolors2d = surf._facecolor3d

    ax.legend()

    ax = fig.add_subplot(1, 2, 2)

    # Plot residual

    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_title("Difference Between Surface 1 and 2")
    cmesh = ax.pcolormesh(X, Y, np.abs(Zvals[1] - Zvals[0]))
```

```
    plt.colorbar(cmesh)

    plt.tight_layout()

    if outfile != "":
        plt.savefig(outfile)

    return fig


def make_video(root: str, outfile: str = "") -> plt.Figure:
    """
    Generates a video from the plt files gathered by root

    Args:
        root (str): captures all files that start with this
                    string and ends in .plt.
        outfile (str): file to save out as. If none is given
                       opens an interactive plot. Should be
                       a .mp4 or .gif.

    Returns:
        plt.Figure: matplotlib figure
    """

    # Load data
    points = load_points(root+"*", return_fname=True)

    # Map timestamp to data
    by_timestamp = {}
    for f in points:
        i1 = f[::-1].find(".")
        i2 = f[::-1].find("_")
        t = int(f[::-1][i1+1:i2][::-1])
        if t in by_timestamp:
            by_timestamp[t].append(points[f])
        else:
            by_timestamp[t] = [points[f]]

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    artists = []
    ax.set_xlabel("X")
    ax.set_ylabel("Y")
    ax.set_zlabel("Z")
    max_PE = max([len(by_timestamp[x]) for x in by_timestamp])
    colors = list(mcolors.TABLEAU_COLORS.keys())[:max_PE]
    for t in sorted(by_timestamp.keys()):
        points = by_timestamp[t]
        container = []
        for i, pts in enumerate(points):
            X, Y, Z = points_to_grid(pts)
            surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, linewidth=1,
                                   alpha=0.8, label=f"PE {i}", color=colors[i])
            surf._edgecolors2d = surf._edgecolor3d
            surf._facecolors2d = surf._facecolor3d
```

```
            container += [surf]

        container += [ax.legend()]
        container += [ax.text(0,0, 1.25, s=str(t), bbox={'facecolor':'w', 'alpha':0.5, 'pad':5},
                                    transform=ax.transAxes, ha="center")]

        for pts in points:
            container += [ax.scatter(pts[:, 0], pts[:, 1], pts[:, 2],
                                     marker=".", c="k", s=0.5, alpha=0.25)]
        artists.append(container)

    ani = animation.ArtistAnimation(fig=fig, artists=artists, interval=100)

    try:
        writer = animation.FFMpegWriter(fps=30)
        ani.save(filename=outfile, writer=writer)
    except:
        print("ffmpeg writer not found or filetype not supported (try .mp4), trying a gif with Pillow")
        try:
            writer = animation.PillowWriter(fps=30)
            ani.save(filename=outfile.replace(".mp4", ".gif"), writer=writer)
        except:
            print("Pillow writer didn't work either :(")

    return fig


if __name__ == "__main__":

    # Simple command line interface
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument("-f", "--file_root", type=str, default="phi_",
                        help="Surface files to plot, takes all files starting with this.")
    parser.add_argument("-f2", "--file_root2", type=str, default="",
                        help="Second set of files to plot in a comparison, takes all files starting with this.")
    parser.add_argument("-flist", "--file_list", type=str, default="",
                        help="list of file roots")
    parser.add_argument("-o", "--output", type=str, default="",
                        help="File to save figure to.")
    parser.add_argument("-t", "--timestamp", type=str, default="")
    parser.add_argument("-v", "--video", type=str, default="")

    args = parser.parse_args()

    print("Arguments: ", args)

    if args.video == "1":
        fig = make_video(args.file_root, args.output)

    elif args.file_root2 == "" and args.file_list == "" and args.video != "1":
```

```
323         fig = single_surface_plot(args.file_root, args.output, args.timestamp)
324 
325     elif args.file_list == "" and args.video != "1":
326         fig = double_surface_plot(args.file_root, args.file_root2, args.output,
    args.timestamp)
327 
328     else:
329         flist = args.file_list.split(",")
330         print(flist)
331         fig = multi_surface_plot(flist, args.output, args.timestamp)
332 
333     if args.output == "":
334         plt.show()
```