HPSC Lab 6

Luna McBride

October 12 2023

1 Application Regimes

1.1 Default Run

This is the run that will be used as a default. This is simply because they are the given values in the original GitHub. This will also be used in these experiments instead of faster options simply because they are experiments. Individual characteristics need to be tested separately to see the effects they give individually. We can compound them later when we have a good idea.

1.1.1 First Run: Grid 10 x 10, flux 200, PE 1 x 1

Time: 38517352340 (ns) = 38.52 sec

The following was taken from the text output, restructured for the sake of my own brain and learning.

Structure = Instruction Type: Number of Instructions (Percentage of Total of that Type)

PROGRAM TOTALS:

Instructions:

Ir: 25,908,333,137 (100.0) I1mr: 3,014,605 (100.0) ILmr: 23,172 (100.0)

Reads:

Dr: 9,593,399,127 (100.0) D1mr: 12,666,112 (100.0) DLmr: 46,911 (100.0)

Writes:

Dw: 4,288,067,298 (100.0) D1mw: 4,073,339 (100.0) DLmw: 227,458 (100.0)
—
TOTAL INSTRUCTIONS IN ESPIC CODE:

Instructions:

Ir: 9,538,300,542 (36.82) I1mr: 467,141 (15.50) ILmr: 453 (1.95)

Reads:

Dr: 3,876,997,154 (40.41) D1mr: 11,493,070 (90.74) DLmr: 2 (0.00)

Writes:

Dw: 1,945,410,081 (45.37) D1mw: 3,494,674 (85.79) DLmw: 0 (0.00)

From these lines, it is shown that the vast majority of instructions are not actually executed in the ESPIC code, but it other calls (most likely MPI calls). The far out calls (DLmw, DLmr, ILmr) in the baseline are thus almost entirely out of my control, as it is outside of the code I can change. This means the target for improvement need to come from the near misses (D1mw, D1mr, ILmr), as the majority of the near misses come from our code (at least in the baseline). The important parts can also be pulled from this same text file:

Major Write (D1mw) Errors:

D1mr:3,244,238 (25.61) D1mw: 2,956,253 (72.58) from
"LaplacianOnGrid.h:LaplacianOnGrid::ParticlesOnMesh(particles, mpiInfo)"

This one portion of the code explains the vast majority of write errors, along with a good amount of read errors. Heck, 72.58 percent coming from just ParticlesOnMesh gives a target to look into. When the total write error represents 85 percent, this means this code explains the majority of all write errors. Looking a bit further, the vast majority of the write error from that function comes from this line:

D1mr: 0 (0.00) D1mw: 2,500,600 (61.39) from
for (int k = 1 ; k ¡= PTCL.n ; ++k) PTCL.xf[k] = PTCL.yf[k] = 0.;

along with the rest being explained by:

D1mr: 201 (0.00) D1mw: 225,104 (5.53) from
PTCL.xf[k] = -0.01*(phiR - phiL) / dx;

and

D1mr: 0 (0.00) D1mw: 225,104 (5.53) from
PTCL.yf[k] = -0.01*(phiT - phiB) / dy;

This means that the vast majority of write errors comes from the initialization of particle values to 0 and assigning the flux calculations. I would thus expect flux to directly change the write errors moving forward. The write errors from there is pretty small, so it could be explained as reasonable

error as it comes from little bits here and there.

Major Read (D1mr) Errors:

The writes may have been toward the end of the file, they were much easier to explain than the reads. The writes had one large bottleneck to explain, compared to the 3 bottlenecks that reads have. There is another 11 percent or so listed as our code but is shown as /curc/sw/install/gcc/11.2.0/include/c++/11.2.0/bits/stlvector.h:std::vector<double, std::allocator<double> >::operator[](unsigned long), but seeing that these are vector allocations, that is not really as explainable by this code. Rather, it puts more weight on vector usage putting a lot of overhead on both the reads and writes. Now, to the bottlenecks:

D1mr:3,244,238 (25.61) D1mw: 2,956,253 (72.58) from
"LaplacianOnGrid.h:LaplacianOnGrid::ParticlesOnMesh(particles, mpiInfo)"

The Particles on Mesh may hold 25 percent, but this is not from the a single instruction like with the write. Instead it comes from multiple calls like this:

D1mr: 625,200 (4.94) D1mw: 0 (0.00) from
if (PTCL.active[k] == 1)

This line, along with finding the bottom and left being more to drop in the bucket, occurs three times with this loss. That means the code is having 15 percent of its read errors checking for active particles.

D1mr: 1,991,382 (15.72) D1mw: 0 (0.00) from
particles.h:particles::move(double)

I am putting this one next because this function marks the return of if (PTCL.active[k] == 1). This has the same read error as the one in Particles on Mesh, so it alone is dropping another 5 percent. Along with the associated vector calls in the if statement, it shows how much of the read misses simply come from vector checking.

D1mr4,295,909 (33.92) D1mw: 127,981 (3.14) from
gaussseidel.h:LaplacianOnGrid::GSorJacobi(int, std::vector¡double, std::allocator¡double¿ ¿, std::vector¡double, std::allocator¡double¿ ¿, mpiInfo, int, int)

D1mr: 4,246,741 (33.53) D1mw: 0 (0.00)
for (int c = 2 ; c ¡= bandwidth ; ++c) newval -= Acoef[r][c] * Solution[Jcoef[r][c]];

The professor did warn about Jacobi and its risks, so this does not surprise me at all. Though, in this case, it is all of the vector calls that are causing the problem.

So, to summarize: Vector Calls (particles) and Jacobi bad.

1.2 Runs 2 and 3: Mesh Testing

1.2.1 Second Run: Grid 5 x 5, flux 200, PE 1 x 1

Time: 17271568496 (ns) = 17.27 sec
TOTAL INSTRUCTIONS IN ESPIC CODE:

Instructions:

Ir: 4,258,249,264 (86.24) I1mr: 347,811 (29.72) ILmr: 588 (2.54)

Reads:

Dr: 1,729,859,963 (90.13) D1mr: 4,550,320 (89.73) DLmr: 10 (0.02)

Writes:

Dw: 820,084,196 (89.07) D1mw: 2,941,066 (91.06) DLmw: 41,172 (18.13)

Interestingly, the approximate numbers of misses are about the same compared to the 10 x 10, just differing a bit in proportions/percentages. On top of that, there was a key shift in loss proportions skewing toward Particles on Mesh:

D1mr: 2,809,290 (55.40) D1mw: 2,812,886 (87.09)
LaplacianOnGrid.h:LaplacianOnGrid::ParticlesOnMesh(particles, mpiInfo)

These values are actually almost the same as the 10 x 10, they just take up more proportionally given that the amount of particles have not changed. The key change appears to be in Jacobi, as it makes up less than 0.1 percent of the read and write error in the 5 x 5 where it held 35 percent read error in the 10 x 10. This is a very significant difference to note, as going from no error to a lot of error means the 5 x 5 may just beat out the 10 x 10 given everything else has remained relatively the same. Plus, faster load times are always great.

Now, to note the long write error that occurred with the 5 x 5:

DLmw: 33,110 (14.58) from ???:???

I am going to say that is an anomaly, as this is all that it gives me. There is not much I can take from it if the computer can only give me ???.

1.2.2 Third Run: Grid 15 x 15, flux 200, PE 1 x 1

Time: 91510384351 (ns) = 91.51 sec
TOTAL INSTRUCTIONS IN ESPIC CODE:

Instructions:

Ir: 22,964,679,381 (33.52) I1mr: 19,155 (0.48) ILmr: 372 (1.61)

Reads:

Dr: 9,319,610,256 (36.81) D1mr: 83,447,522 (96.55) DLmr: 3 (0.01)

Writes:

Dw: 4,798,549,962 (42.39) D1mw: 7,770,134 (90.28) DLmw: 5 (0.00)

D1mr: 53,616,267 (62.03) D1mw: 3,421,572 (39.75) from
gaussseidel.h:LaplacianOnGrid::GSorJacobi(int, std::vector<double, std::allocator<double> >, std::vector¡double, std::allocator<double> >, mpiInfo, int, int)

The big change in this case should not be a surprise, as it is Jacobi again. This time, however, read errors jump from 4 million to 53 million. Write errors now take up 39.75 percent of the total errors where it was the particles on mesh before, which actually remained pretty stable compared to the 10 x 10. This means that you need to stay low if you are going to use Jacobi. The bigger the mesh, the bigger the jump.

1.3 Runs 4 and 5: Flux Testing

Fourth Run: Grid 10 x 10, flux 100, PE 1 x 1
Time: 28297008986 (ns) = 28.30 sec
TOTAL INSTRUCTIONS IN ESPIC CODE:

Instructions:

Ir: 7,437,608,070 (68.43) I1mr: 386,100 (26.48) ILmr: 478 (2.06)

Reads:

Dr: 3,002,878,352 (71.90) D1mr: 7,932,282 (87.12) DLmr: 2 (0.00)

Writes:

Dw: 1,491,277,635 (75.12) D1mw: 3,045,064 (91.23) DLmw: 41,455 (18.20)

The proportions of the errors did not really change. In fact, the only major change I could find was with read errors in general, that went down from 11 million to 8 million. This likely just comes from the lower number of writes in general, but the change was not very significant. I think the amount of particles will do more than the flux, but that is just how the assignment was laid out.

1.3.1 Fifth Run: Grid 10 x 10, flux 300, PI 1 x 1

Time: 43136663777 (ns) = 43.14 sec

TOTAL INSTRUCTIONS IN ESPIC CODE:

Instructions:

Ir: 10,499,156,342 (43.00) I1mr: 1,233,001 (28.20) ILmr: 459 (1.98)

Reads:

Dr: 4,275,576,775 (46.95) D1mr: 11,669,391 (89.96) DLmr: 2 (0.00)

Writes:

Dw: 2,153,910,533 (51.96) D1mw: 3,518,306 (90.63) DLmw: 1 (0.00)

Once again, there were not very many differences besides base read and base read error counts. The proportion of total instruction calls did also move away from our code to other code with different fluxes. This means less instructions were explainable from the esPIC code as flux increased, thus a lower flux would be better for the sake of our control.

1.4 Runs 6 and 7: MPI Testing

1.4.1 Sixth Run: Grid 10 x 10, flux 200, PE 2 x 1

Two PEs, two times.
Time: 52633270447 (ns) = 52.63 sec Time: 53043315342 (ns) = 53.04 sec
The txt file output with multiple PEs appears not to generate individual call percentage logs. However, it makes sense that more is being shunted away from the program to MPI calls, so there is less that is explainable by the code. The output did give me two things of note, however:

D1mr: 21,926,915 (44.11) D1mw: 814,144 (10.54) from
???:LaplacianOnGrid::GSorJacobi(int, std::vector<double, std::allocator<double> >, std::vector<double, std::allocator<double> >, mpiInfo, int, int)

This shows higher read errors occurring in Jacobi, which makes sense from the mesh examples given that more PEs means more important edges coming from multiple different processes.

D1mr: 324,851 (0.65) D1mw: 1,475,225 (19.09) from
???:mpiInfo::ExchangeBoundaryInfo(std::vector<double, std::allocator<double> >, std::vector<double, std::allocator<double> >)

A new level of write error is occurring in boundary info exchange, adding another layer of error that can be caused. This on top of Jacobi show that edges are also a limiting factor of these grids.

1.4.2 Seventh Run: Grid 10 x 10, flux 200, PE 2 x 2

Time: 78576186961 (ns) = 78.58 sec Time: 78635866150 (ns) = 78.64 sec Time: 78667445103 (ns) = 78.67 sec Time: 78681031692 (ns) = 78.68 sec

Second verse, same as the first. Once again, Jacobi kept a similar proportion of read and write errors, just with more of them given that there are more PEs. Edge cases continue to take up more cache misses as well.

2 Target for Improvement

Jacobi appears to be a key cache sink for two out of the three aspects measured. These two aspects also made the most difference, so Jacobi usage is a good thing to put attention into. Vector usage could always be tweaked to further optimize, but that holds less water compared to Jacobi as mesh and PE increases.

3 Proposed Improvement

For this improvement experiment, I intend to look into how I could lower the excess memory calls from the GSorJacobi function. There is an egregious use of memory shown especially in this line:

for (int c = 2 ; c ¡= bandwidth ; ++c) newval -= Acoef[r][c] * Solution[Jcoef[r][c]];

This line exists in one of the RLoops defined at the beginning of the semester, so this is using a double for loop, causing a Big O n^2. My first intuition would be to pull the Acoef[r] and Jcoef[r] first before the second loop so that individual elements will not have to go far to be pulled. I am hoping doing so will remove excess calls in the same way the professor described sending a whole vector to MPI speeding things up (removing the overhead of the actual send and receive by lowering how often we do it). This will hopefully pull these arrays into a closer cache for each case, removing the back and forth.

These for loops also exist in a while loop checking for convergence, cranking up that big O even further. This while loop also includes multiple MPI calls as well as ExchangeBoundaryInfo, so I will consider this function fair game in the Jacobi discussion. I do not think the global convergence calls can be optimized, so I will leave them alone. In the case of the ExchangeBoundaryInfo, I feel it is fair to assume Solution and b are in local cache due to the minimal read errors. There are high write errors, which makes me think it comes from PHISend and Recv variables. I feel this is the case since they are defined in the GridDecomposition Function. I honestly cannot see a reason why they are defined there, so moving the definitions, as well as perhaps removing the initialization, may remove a major component of the write errors.

Since the point is to type this portion up before the improvement, I am coming from a point of not trying such optimizations in years. I very well could find something different through experimentation, which I will outline in the Evaluation of improvement section. These are just my initial assumptions when looking at the function.

These tests will be done using 1x1 PEs and the baseline 10x10 described above, as the 10x10 already gives a good idea about excess read misses. It will just be interesting to see how it applies to more PEs after some experimentation has been done, mostly because of how much is added with Jacobi on multiple PEs.

4 Evaluation of Improvement

The direction I took to start did not end up doing too crazy much. There was not much I could pull out besides making the aCoef[r] occur before the c loop, but that did not do too much. Then I spruced up ExchangeBoundaryInfo to remove some redundant loops, as well as going down a rabbit hole with the definition placement of the phiSend and Recv arrays, but that did more harm than good. I am still not entirely sure why that has to be in the grid definition, but it chugs if it is not in there.

I then stumbled across this forum when trying to change vectors into arrays:

https://stackoverflow.com/questions/2923272/how-to-convert-vector-to-array , specifically calling attention to this line of code: std::vector<double> v; double* a = v[0];. If I am understanding the given explanations correctly, what this is doing is setting a pointer to the array underlying the vector implementation, thus being able to be treated like (and computationally be faster like) an array while still keeping the vector implementation, thus not needing to break the scaling functionality elsewhere in the code. This caused me to shift in a new direction.

I was able to get the Acoef and Jcoef in the Jacobi code to use the underlying arrays, along with various bits the particle methods in the ParticlesOnMesh, to work using the underlying arrays. This brought the time down on the 1x1 by half a second. On top of that, the 2x1 was brought down by a whole second, which is definitely something.

That was until I changed something to try to bring down the 1x1 further (I do not know what) that suddenly upped the 2x1 time by 10 seconds. I am not even sure what did it, because I did batches of changes on each setting to get an idea of how multiple PI sections were reacting to changes. I even tried reverting each file individually to its base state to see what file it was coming from with no change. I then tried remaking esPIC again, but that caused even more problems.

Now, I am not entirely sure how this was working before. I would not need to remake it to run it, and I could tell from the log text files that those changes were occurring without remaking it. After I did, however (and fixed a missing semi-colon here and there), the whole thing stopped working. It was now throwing segmentation faults. Of course, the time between the catalyst 10 second increase and typing this now (11:30pm before the deadline) was about 1 hour. I did not run into this in the multiple full days I had spent on this before this point. I just wanted to get my time lower. Instead of doing that, everything broke and I no longer have the time to try and fix it. I guess I should have stopped at the 0.5 and 1 second time save while I was ahead.

5 Self Evaluation

I feel like things fell apart at the end. I had things working fine, I could not figure out why something changed its timing how it did, so I tried to use the make command. Things were changing and working just fine without making, so that ended up being a huge mistake. Now it just throws segmentation faults. I do not know what this change did, or why it was working well before it, and

I do not have the time anymore to try and fix it. I spent all day on Thursday and Today trying to bring these times down already. This class' work is just not my bread and butter. I honestly spent too much time on this.

6 Appendix A: Reference Test Results

```
--------------------------------------------------------------------------------
Profile data file 'cachegrind.10x10_1x1_200.out'
--------------------------------------------------------------------------------
I1 cache:         32768 B, 64 B, 8-way associative
D1 cache:         32768 B, 64 B, 8-way associative
LL cache:         268435456 B, 64 B, direct-mapped
Profiled target:  ./esPIC -nPEx 1 -nPEy 1 -nCellx 10 -nCelly 10 -nPtcl 50000 -flux
    200 -tEnd 2 -dt .01 -tPlot .2
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:
Auto-annotation:  on

--------------------------------------------------------------------------------
Ir                      I1mr               ILmr            Dr
   D1mr                DLmr            Dw                      D1mw
   DLmw
--------------------------------------------------------------------------------
25,908,333,137 (100.0%) 3,014,605 (100.0%) 23,172 (100.0%) 9,593,399,127 (100.0%)
   12,666,112 (100.0%) 46,911 (100.0%) 4,288,067,298 (100.0%) 4,073,339 (100.0%)
   227,458 (100.0%)  PROGRAM TOTALS

--------------------------------------------------------------------------------
Ir                      I1mr             ILmr           Dr
   D1mr                DLmr            Dw                     D1mw
   DLmw               file:function
--------------------------------------------------------------------------------
10,291,035,119 (39.72%)  35,112 ( 1.16%) 6,357 (27.43%) 3,860,613,242 (40.24%)
   135,172 ( 1.07%)    396 ( 0.84%) 1,498,470,783 (34.95%)    98,905 ( 2.43%)
   33,111 (14.56%)  ???:???
 4,920,870,024 (18.99%)      21 ( 0.00%)     5 ( 0.02%) 1,604,631,528 (16.73%)
       12 ( 0.00%)      0            641,852,613 (14.97%)         1 ( 0.00%)
    0           ???:ucp_worker_progress
 2,290,995,707 ( 8.84%)      58 ( 0.00%)     1 ( 0.00%) 1,041,361,685 (10.85%)
   862,168 ( 6.81%)      0            624,817,011 (14.57%)         0
     0           /curc/sw/install/gcc/11.2.0/include/c++/11.2.0/bits/stl_vector.h:
   std::vector<double, std::allocator<double> >::operator[](unsigned long)
 2,216,711,522 ( 8.56%)   1,315 ( 0.04%)    17 ( 0.07%)   849,481,380 ( 8.85%)
   4,295,909 (33.92%)      0            367,084,667 ( 8.56%)   127,981 ( 3.14%)
       0           gauss_seidel.h:LaplacianOnGrid::GS_or_Jacobi(int, std::vector<
   double, std::allocator<double> >, std::vector<double, std::allocator<double>
   >&, mpiInfo&, int, int&)
 1,755,317,906 ( 6.78%)   3,330 ( 0.11%)    46 ( 0.20%)   693,671,899 ( 7.23%)
   3,244,238 (25.61%)      0            230,073,443 ( 5.37%) 2,956,253 (72.58%)
```

```
      0           LaplacianOnGrid.h:LaplacianOnGrid::ParticlesOnMesh(particles&,
    mpiInfo&)
29  975,823,255 ( 3.77%)      18 ( 0.00%)     0            443,556,025 ( 4.62%)
    812,632 ( 6.42%)      0            266,133,615 ( 6.21%)         0
      0           /curc/sw/install/gcc/11.2.0/include/c++/11.2.0/bits/stl_vector.h:
    std::vector<int, std::allocator<int> >::operator[](unsigned long)
30  624,490,216 ( 2.41%)      19 ( 0.00%)     1 ( 0.00%)   223,032,220 ( 2.32%)
      0              0           133,819,332 ( 3.12%)         0
      0           /curc/sw/install/gcc/11.2.0/include/c++/11.2.0/bits/stl_vector.h:
    std::vector<std::vector<double, std::allocator<double> >, std::allocator<std::
    vector<double, std::allocator<double> > > >::operator[](unsigned long)
31  534,877,183 ( 2.06%)       1 ( 0.00%)     1 ( 0.00%)   106,975,438 ( 1.12%)
      3 ( 0.00%)      0           106,975,436 ( 2.49%)         0
      0           ???:opal_common_ucx_mca_pmix_fence
32  499,637,446 ( 1.93%)      11 ( 0.00%)     1 ( 0.00%)   178,441,945 ( 1.86%)
    200 ( 0.00%)      0           107,065,167 ( 2.50%)         0
      0           /curc/sw/install/gcc/11.2.0/include/c++/11.2.0/bits/stl_vector.h:
    std::vector<std::vector<int, std::allocator<int> >, std::allocator<std::vector<
    int, std::allocator<int> > > >::operator[](unsigned long)
33  373,883,400 ( 1.44%)     452 ( 0.01%)     7 ( 0.03%)   117,552,520 ( 1.23%)
    1,991,382 (15.72%)      0            48,776,960 ( 1.14%)         0
      0           particles.h:particles::move(double)
34  299,668,502 ( 1.16%)      10 ( 0.00%)     1 ( 0.00%)   128,429,358 ( 1.34%)
      0              0            85,619,572 ( 2.00%)         0
      0           LaplacianOnGrid.h:LaplacianOnGrid::pid(int, int)
35  181,235,333 ( 0.70%) 861,483 (28.58%)   109 ( 0.47%)    37,269,353 ( 0.39%)
    328 ( 0.00%)      5 ( 0.01%)    21,743,272 ( 0.51%)       205 ( 0.01%)
      0           ???:__printf_fp_l
36  149,187,348 ( 0.58%)      20 ( 0.00%)     2 ( 0.01%)    74,593,674 ( 0.78%)
    499 ( 0.00%)      0            24,864,558 ( 0.58%)         0
      0           /curc/sw/install/gcc/11.2.0/include/c++/11.2.0/bits/stl_vector.h:
    std::vector<double, std::allocator<double> >::size() const
37  114,944,410 ( 0.44%)   3,148 ( 0.10%)    27 ( 0.12%)    54,752,865 ( 0.57%)
    52,721 ( 0.42%)      0            11,557,195 ( 0.27%)   314,375 ( 7.72%)
      0           mpiInfo.h:mpiInfo::ExchangeBoundaryInfo(std::vector<double, std::
    allocator<double> >&, std::vector<double, std::allocator<double> >&)
38   62,097,680 ( 0.24%) 826,756 (27.43%)   101 ( 0.44%)    14,238,587 ( 0.15%)
     1,660 ( 0.01%)     16 ( 0.03%)    11,968,818 ( 0.28%)     1,664 ( 0.04%)
    19 ( 0.01%)  ???:vfprintf
39   60,231,796 ( 0.23%)     837 ( 0.03%)     6 ( 0.03%)    15,225,652 ( 0.16%)
    159,271 ( 1.26%)      0             5,342,534 ( 0.12%)    29,616 ( 0.73%)
      0           particles.h:particles::add(std::vector<double, std::allocator<
    double> >&, std::vector<double, std::allocator<double> >&, std::vector<double,
    std::allocator<double> >&, std::vector<double, std::allocator<double> >&)
40   50,622,772 ( 0.20%)     164 ( 0.01%)     6 ( 0.03%)    15,531,406 ( 0.16%)
      0              0             7,708,005 ( 0.18%)        20 ( 0.00%)
      0           ???:hack_digit
41   38,700,762 ( 0.15%)     182 ( 0.01%)     5 ( 0.02%)     5,373,376 ( 0.06%)
      0              0             3,654,726 ( 0.09%)         0
      0           ???:__mpn_mul_1
42   38,361,960 ( 0.15%)      20 ( 0.00%)     1 ( 0.00%)    16,440,840 ( 0.17%)
      0              0            10,960,560 ( 0.26%)       200 ( 0.00%)
      0           mpiInfo.h:mpiInfo::pid(int, int)
43   18,250,756 ( 0.07%)   3,866 ( 0.13%)     5 ( 0.02%)     3,434,317 ( 0.04%)
     10 ( 0.00%)      0             4,294,029 ( 0.10%)         0
```

```
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
include/bits/locale_facets.tcc:std::ostreambuf_iterator<char, std::char_traits<
char> > std::num_put<char, std::ostreambuf_iterator<char, std::char_traits<char
> > >::_M_insert_float<double>(std::ostreambuf_iterator<char, std::char_traits<
char> >, std::ios_base&, char, char, double) const
44  14,011,600 ( 0.05%)   3,488 ( 0.12%)    39 ( 0.17%)     5,352,600 ( 0.06%)
10,000 ( 0.08%)      1 ( 0.00%)     2,461,600 ( 0.06%)    55,800 ( 1.37%)
 0           LaplacianOnGrid.h:LaplacianOnGrid::FormLS(mpiInfo&)
45  13,400,901 ( 0.05%)   5,803 ( 0.19%)    14 ( 0.06%)     2,538,260 ( 0.03%)
   773 ( 0.01%)     29 ( 0.06%)     1,592,041 ( 0.04%)     7,182 ( 0.18%)
1,563 ( 0.69%)  ???:__memcpy_avx_unaligned_erms
46  13,008,588 ( 0.05%)      99 ( 0.00%)     3 ( 0.01%)     3,776,682 ( 0.04%)
     0              0              1,678,512 ( 0.04%)         0
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
include/bits/fstream.tcc:std::basic_filebuf<char, std::char_traits<char> >::
xsputn(char const*, long)
47  12,591,090 ( 0.05%)      41 ( 0.00%)     2 ( 0.01%)     2,938,006 ( 0.03%)
     0              0              2,098,530 ( 0.05%)         0
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
include/bits/streambuf.tcc:std::basic_streambuf<char, std::char_traits<char>
>::xsputn(char const*, long)
48  11,634,548 ( 0.04%)     140 ( 0.00%)    12 ( 0.05%)     4,053,032 ( 0.04%)
57,201 ( 0.45%)      0              1,145,419 ( 0.03%)        10 ( 0.00%)
 0           particle_plotter.h:particles::plot(std::__cxx11::basic_string<char
, std::char_traits<char>, std::allocator<char> >, int, int)
49  11,278,229 ( 0.04%)     703 ( 0.02%)     5 ( 0.02%)     3,609,216 ( 0.04%)
     6 ( 0.00%)      2 ( 0.00%)     3,545,112 ( 0.08%)       615 ( 0.02%)
131 ( 0.06%)  ???:_IO_default_xsputn
50  11,133,830 ( 0.04%)   1,687 ( 0.06%)    37 ( 0.16%)     3,376,506 ( 0.04%)
149,160 ( 1.18%)  3,081 ( 6.57%)     1,284,407 ( 0.03%)       698 ( 0.02%)
 25 ( 0.01%)  ???:do_lookup_x
51  10,989,179 ( 0.04%)     349 ( 0.01%)     4 ( 0.02%)     2,370,173 ( 0.02%)
     0              0              2,801,127 ( 0.07%)       144 ( 0.00%)
 3 ( 0.00%)  ???:vsnprintf
52  10,450,860 ( 0.04%)      23 ( 0.00%)     2 ( 0.01%)     2,612,715 ( 0.03%)
    44 ( 0.00%)      0              2,612,715 ( 0.06%)         0
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
include/bits/ostream.tcc:std::ostream::sentry::sentry(std::ostream&)
53  10,202,306 ( 0.04%)   1,382 ( 0.05%)     8 ( 0.03%)       914,381 ( 0.01%)
   869 ( 0.01%)    172 ( 0.37%)             0                  0
 0           ???:__strchrnul_avx2
54   9,671,760 ( 0.04%)      74 ( 0.00%)     3 ( 0.01%)     1,289,568 ( 0.01%)
     0              0              4,513,488 ( 0.11%)        29 ( 0.00%)
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
include/x86_64-pc-linux-gnu/bits/c++locale.h:std::__convert_from_v(
__locale_struct* const&, char*, int, char const*, ...)
55   9,456,920 ( 0.04%)      86 ( 0.00%)     2 ( 0.01%)     3,868,740 ( 0.04%)
   145 ( 0.00%)      0              1,719,440 ( 0.04%)         0
 0           ???:uselocale
56   8,882,896 ( 0.03%)      36 ( 0.00%)     3 ( 0.01%)     3,249,840 ( 0.03%)
   165 ( 0.00%)      0              1,516,592 ( 0.04%)         9 ( 0.00%)
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
libsupc++/../../../../libstdc++-v3/libsupc++/dyncast.cc:__dynamic_cast
57   8,834,516 ( 0.03%) 228,059 ( 7.57%)     3 ( 0.01%)     3,016,664 ( 0.03%)
     0              0              3,016,664 ( 0.07%)        27 ( 0.00%)
```

```
 0           /curc/sw/build/gcc/11.2.0/build/x86_64-pc-linux-gnu/libstdc++-v3/
libsupc++/../../../../libstdc++-v3/libsupc++/vmi_class_type_info.cc:__cxxabiv1
::__vmi_class_type_info::__do_dyncast(long, __cxxabiv1::__class_type_info::
__sub_kind, __cxxabiv1::__class_type_info const*, void const*, __cxxabiv1::
__class_type_info const*, void const*, __cxxabiv1::__class_type_info::
__dyncast_result&) const
```

Listing 1: Reference Baseline Problem Areas

```
1       2,000 ( 0.00%)  11 ( 0.00%) 2 ( 0.01%)           0                   0
     0                1,600 ( 0.00%)       0          0           void
GS_or_Jacobi(int max_iter , VD RHS, VD &Solution , mpiInfo &myMPI , int
GSorJacobi, int &finalIterCount )
2            .            .          .                   .                   .
     .                    .               .          .             {
3            .            .          .                   .                   .
     .                    .               .          .               int
converged, it_converged;
4         200 ( 0.00%)   0          0                   0                   0
     0                  200 ( 0.00%)       0          0               int iter
= 0;
5            .            .          .                   .                   .
     .                    .               .          .               double
newval;
6         400 ( 0.00%)   0          0                   0                   0
     0                  200 ( 0.00%)       0          0               double
cur_delta = 0.;
7         400 ( 0.00%)  11 ( 0.00%) 1 ( 0.00%)          0                   0
     0                  200 ( 0.00%)       0          0               double
max_delta = 0.;
8            .            .          .                   .                   .
     .                    .               .          .               double
tol;
9            .            .          .                   .                   .
     .                    .               .          .
10           .            .          .                   .                   .
     .                    .               .          .
MPI_Status status;
11        200 ( 0.00%)   0          0                   0                   0
     0                  200 ( 0.00%)       0          0               int
  tag = 0;
12           .            .          .                   .                   .
     .                    .               .          .               int
  err;
13           .            .          .                   .                   .
     .                    .               .          .               int
  global_converged;
14        200 ( 0.00%)   0          0                   0                   0
     0                  200 ( 0.00%)       0          0               int
  zero = 0;
15        200 ( 0.00%)   0          0                   0                   0
     0                  200 ( 0.00%)       0          0               int
  one  = 1;
16           .            .          .                   .                   .
     .                    .               .          .
```

```
        .            .          .                   .                  .
    .                  .               .          .
     2,200 ( 0.00%)  13 ( 0.00%) 1 ( 0.00%)        200 ( 0.00%)          0
    0                 600 ( 0.00%)       0          0                VD
SolutionNew; SolutionNew.resize(Solution.size());
        .            .          .                   .                  .
    .                  .               .          .
        .            .          .                   .                  .
    .                  .               .          .                //
rLOOP Solution[r] = 0.;
       400 ( 0.00%)   0          0                 200 ( 0.00%)          0
    0                 200 ( 0.00%)       0          0                tol
 = 1.e-04;
        .            .          .                   .                  .
    .                  .               .          .
        .            .          .                   .                  .
    .                  .               .          .                //
=====================================================
        .            .          .                   .                  .
    .                  .               .          .                // Begin
Iterations
        .            .          .                   .                  .
    .                  .               .          .                //
=====================================================
        .            .          .                   .                  .
    .                  .               .          .
       200 ( 0.00%)   0          0                  0                  0
    0                 200 ( 0.00%)       0          0                converged
 = 0;
        .            .          .                   .                  .
    .                  .               .          .
        .            .          .                   .                  .
    .                  .               .          .
   157,085 ( 0.00%)  10 ( 0.00%) 1 ( 0.00%)     52,295 ( 0.00%)          0
    0                  0               0          0                while (
converged == 0 )
        .            .          .                   .                  .
    .                  .               .          .                  {
        .            .          .                   .                  .
    .                  .               .          .              //
-----------------------------------------------
        .            .          .                   .                  .
    .                  .               .          .              // (1)
Parallel communication on PE Boundaries
        .            .          .                   .                  .
    .                  .               .          .              //
-----------------------------------------------
        .            .          .                   .                  .
    .                  .               .          .
   366,065 ( 0.00%)  13 ( 0.00%) 1 ( 0.00%)    156,885 ( 0.00%)          0
    0              52,295 ( 0.00%)       0          0              myMPI.
ExchangeBoundaryInfo(Solution,b);
        .            .          .                   .                  .
    .                  .               .          .
        .            .          .                   .                  .
```

```
      .                  .               .          .            //
   ---------------------------------------------
         .            .          .                   .                 .
      .                  .               .          .            // (2)
   Convergence
         .            .          .                   .                 .
      .                  .               .          .            //
   ---------------------------------------------
         .            .          .                   .                 .
      .                  .               .          .
     313,770 ( 0.00%)   0         0             156,885 ( 0.00%)         0
        0                    0               0          0            if ( ++iter
    > max_iter )
         .            .          .                   .                 .
      .                  .               .          .                {
         .            .          .                   .                 .
      .                  .               .          .
   finalIterCount = iter;
         .            .          .                   .                 .
      .                  .               .          .                    return;
         .            .          .                   .                 .
      .                  .               .          .                }
         .            .          .                   .                 .
      .                  .               .          .
     156,885 ( 0.00%)  10 ( 0.00%) 1 ( 0.00%)          0                 0
        0             104,590 ( 0.00%)       0          0            max_delta
     = 0.;  it_converged = 1;
         .            .          .                   .                 .
      .                  .               .          .
         .            .          .                   .                 .
      .                  .               .          .            //
   ---------------------------------------------
         .            .          .                   .                 .
      .                  .               .          .            // (3) One
   Jacobi Iteration
         .            .          .                   .                 .
      .                  .               .          .            //
   ---------------------------------------------
         .            .          .                   .                 .
      .                  .               .          .
  44,503,045 ( 0.17%) 200 ( 0.01%) 1 ( 0.00%)  35,508,305 ( 0.37%)         0
        0              52,295 ( 0.00%)       0          0            rLOOP
         .            .          .                   .                 .
      .                  .               .          .                {
         .            .          .                   .                 .
      .                  .               .          .
         .            .          .                   .                 .
      .                  .               .          .
         .            .          .                   .                 .
      .                  .               .          .                  //
   (3.1) Compute new guess for row r
         .            .          .                   .                 .
      .                  .               .          .
  79,540,695 ( 0.31%)   0         0           26,513,565 ( 0.28%)    35,610 (
   0.28%) 0            17,675,710 ( 0.41%)       0          0
```

```
    newval = b[r];
1,573,138,190 ( 6.07%) 225 ( 0.01%) 3 ( 0.01%) 592,136,285 ( 6.17%) 4,246,741
    (33.53%) 0          256,297,795 ( 5.98%)       0          0                  for
     ( int c = 2 ; c <= bandwidth ; ++c ) newval -=  Acoef[r][c] * Solution[Jcoef[r
    ][c]];
  123,729,970 ( 0.48%) 200 ( 0.01%) 1 ( 0.00%)  35,351,420 ( 0.37%)          0
        0           26,513,565 ( 0.62%)       0          0                  newval
    /= Acoef[r][1];
              .            .          .                    .                   .
         .                    .               .          .
              .            .          .                    .                   .
         .                    .               .          .                   //
    (3.2) Convergence check
              .            .          .                    .                   .
         .                    .               .          .
   97,216,405 ( 0.38%) 200 ( 0.01%) 1 ( 0.00%)  44,189,275 ( 0.46%)       200 (
    0.00%) 0           17,675,710 ( 0.41%)       0          0
    cur_delta  = fabs(Solution[r] - newval);
              .            .          .                    .                   .
         .                    .               .          .
   30,562,142 ( 0.12%)   0          0           17,675,710 ( 0.18%)          0
        0           4,048,577 ( 0.09%)       0          0                  if (
    cur_delta > tol ) it_converged = 0;
              .            .          .                    .                   .
         .                    .               .          .
              .            .          .                    .                   .
         .                    .               .          .                   //
    (3.3) Record new value in solution
              .            .          .                    .                   .
         .                    .               .          .
   70,702,840 ( 0.27%) 199 ( 0.01%) 1 ( 0.00%)  17,675,710 ( 0.18%)          0
        0           17,675,710 ( 0.41%) 127,843 ( 3.14%) 0
    SolutionNew[r]       = newval;
              .            .          .                    .                   .
         .                    .               .          .
   17,675,710 ( 0.07%)   0          0             8,837,855 ( 0.09%)          0
        0                    0               0          0                  if (
    GSorJacobi == 1 ) Solution[r] = newval;  // Gauss-Seidel, update Solution as we
     go
              .            .          .                    .                   .
         .                    .               .          .
              .            .          .                    .                   .
         .                    .               .          .                 }
              .            .          .                    .                   .
         .                    .               .          .
  177,070,870 ( 0.68%)  10 ( 0.00%) 1 ( 0.00%)  70,859,725 ( 0.74%)    13,283 (
    0.10%) 0           26,565,860 ( 0.62%)       0          0              rLOOP
    Solution[r] = SolutionNew[r];
              .            .          .                    .                   .
         .                    .               .          .
              .            .          .                    .                   .
         .                    .               .          .              //
    ----------------------------------------------
              .            .          .                    .                   .
         .                    .               .          .              // (4) Make
```

```
    note of the convergence state
82          .            .          .                    .                    .
       .                    .                .          .             //
   -----------------------------------------------
83          .            .          .                    .                    .
       .                    .                .          .
84        104,590 ( 0.00%)   0          0               52,295 ( 0.00%)         0
          0               52,295 ( 0.00%)       0          0             converged =
    it_converged;
85          .            .          .                    .                    .
       .                    .                .          .
86          .            .          .                    .                    .
       .                    .                .          .             //
   -----------------------------------------------
87          .            .          .                    .                    .
       .                    .                .          .             // (5)
   Gather convergence information from PEs
88          .            .          .                    .                    .
       .                    .                .          .             //
   -----------------------------------------------
89          .            .          .                    .                    .
       .                    .                .          .
90         52,295 ( 0.00%)   0          0                0                    0
          0               52,295 ( 0.00%)       0          0             int root =
   0;
91        679,835 ( 0.00%)  13 ( 0.00%) 1 ( 0.00%)     104,590 ( 0.00%)         0
          0              156,885 ( 0.00%)       0          0             err =
   MPI_Reduce( &converged       , &global_converged, one , MPI_INT , MPI_MIN ,
   zero , MPI_COMM_WORLD );
92        470,655 ( 0.00%)   0          0               104,590 ( 0.00%)         0
          0              104,590 ( 0.00%)       0          0             err =
   MPI_Bcast ( &global_converged,                    one , MPI_INT,
   zero , MPI_COMM_WORLD );
93        104,590 ( 0.00%)   0          0                52,295 ( 0.00%)         0
          0               52,295 ( 0.00%)       0          0             converged =
    global_converged;
94          .            .          .                    .                    .
       .                    .                .          .
95        156,885 ( 0.00%)   0          0                52,295 ( 0.00%)         0
          0                     0                0          0             if (
   converged == 1 )
96          .            .          .                    .                    .
       .                    .                .          .                  {
97          600 ( 0.00%)   0          0                  400 ( 0.00%)        75 (
   0.00%) 0                     200 ( 0.00%)     138 ( 0.00%) 0
   finalIterCount = iter;
98          400 ( 0.00%)   0          0                  0                    0
          0                     0                0          0                 return;
```

Listing 2: Reference Baseline Jacobi

```
1
2  ------------------------------------------------------------------------------
3  Ir                    I1mr             ILmr          Dr                    D1mr
                  DLmr        Dw                     D1mw               DLmw
4  ------------------------------------------------------------------------------
```

```
9,538,300,542 (36.82%) 467,141 (15.50%) 453 ( 1.95%) 3,876,997,154 (40.41%)
    11,493,070 (90.74%) 2 ( 0.00%) 1,945,410,081 (45.37%) 3,494,674 (85.79%) 0
          events annotated
```

Listing 3: Reference Baseline End Run

7 Appendix B: Source Code Listing

```

echo Running $1 $2 $3 $4 $5 $6...

mpirun -n $4 --oversubscribe -mca btl ^openib valgrind --tool=cachegrind --log-
    file="cachegrind.$1x$2_$5x$6_$3.log" --cachegrind-out-file="cachegrind.
    $1x$2_$5x$6_$3.out" ./esPIC -nPEx $5 -nPEy $6 -nCellx $1 -nCelly $2 -nPtcl
    50000  -flux $3  -tEnd 2 -dt .01 -tPlot .2
mpirun -n $4 --oversubscribe -mca btl ^openib valgrind --tool=callgrind  --log-
    file="callgrind.$1x$2_$5x$6_$3.log"  --callgrind-out-file="callgrind.
    $1x$2_$5x$6_$3.out" ./esPIC -nPEx $5 -nPEy $6 -nCellx $1 -nCelly $2 -nPtcl
    50000  -flux $3  -tEnd 2 -dt .01 -tPlot .2


callgrind_annotate --auto=yes cachegrind.$1x$2_$5x$6_$3.out >  cachegrind.
    $1x$2_$5x$6_$3.txt
```

Listing 4: runcase to Run Everything

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

//  ||
    ||
//  ||              esPIC
    ||
//  ||              ------------------------------------------------------
    ||
//  ||              E L E C T R O S T I C   P A R T I C L E - I N - C E L L
    ||
//  ||
    ||
//  ||              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.                                        ||
//  ||                                                                         ||
//  ================================================================================


  //  ==
  //  ||
  //  ||  Perform Jacobiiterations on nField X nField linear
  //  ||  system:  A*Solution = RHS
  //  ||
  //  ==

void GS_or_Jacobi(int max_iter , VD RHS, VD &Solution , mpiInfo &myMPI , int
    GSorJacobi, int &finalIterCount )
  {
    int converged, it_converged;
    int iter = 0;
    double newval;
    double cur_delta = 0.;
    double max_delta = 0.;
    double tol;
    double* aCoeff = &Acoef[0][0];
    int* jCoeff = &Jcoef[0][0];

    MPI_Status status;
    int        tag = 0;
    int        err;
    int        global_converged;
    int        zero = 0;
    int        one  = 1;


    //VD SolutionNew; SolutionNew.resize(Solution.size());

    //rLOOP Solution[r] = 0.;
    tol       = 1.e-04;

    // ====================================================
    // Begin Iterations
    // ====================================================

    converged = 0;
    int* jCo;
    double* aCo;


```

```
 while ( converged == 0 )
   {
// ------------------------------------------------
// (1) Parallel communication on PE Boundaries
// ------------------------------------------------

myMPI.ExchangeBoundaryInfo(Solution,b);

// ------------------------------------------------
// (2) Convergence
// ------------------------------------------------

if ( ++iter > max_iter )
  {
    finalIterCount = iter;
    return;
  }

max_delta    = 0.;  it_converged = 1;

// ------------------------------------------------
// (3) One Jacobi Iteration
// ------------------------------------------------

rLOOP
  {


    // (3.1) Compute new guess for row r

    newval = b[r];
    jCo = &jCoeff[r];
    aCo = &aCoeff[r];
    for ( int c = 2 ; c <= bandwidth ; ++c ) newval -=  aCo[c] * Solution[jCo[c
]];
    newval /= aCo[1];

    // (3.2) Convergence check

    cur_delta  = fabs(Solution[r] - newval);

    if ( cur_delta > tol ) it_converged = 0;

    // (3.3) Record new value in solution

    Solution[r]       = newval;

    //if ( GSorJacobi == 1 ) Solution[r] = newval;  // Gauss-Seidel, update
 Solution as we go

  }

//rLOOP Solution[r] = SolutionNew[r];

// ------------------------------------------------
```

```
  // (4) Make note of the convergence state
  // -------------------------------------------

  converged = it_converged;

  // -------------------------------------------
  // (5) Gather convergence information from PEs
  // -------------------------------------------

  int root = 0;
  err = MPI_Reduce( &converged       , &global_converged, one , MPI_INT , MPI_MIN
    , zero , MPI_COMM_WORLD );
  converged = global_converged;
  err = MPI_Bcast ( &global_converged,                    one , MPI_INT,
      zero , MPI_COMM_WORLD );

  if ( converged == 1 )
    {

      finalIterCount = iter;
      return;
    }

     }

}
```

Listing 5: Updated gaussseidel.h

```

 void ExchangeBoundaryInfo(VD &Solution, VD &b)
{


 // -------------------------------------------
 // (1) Parallel communication on PE Boundaries:   ** See fd.h for tLOOP and
  sLOOP macros **
 // -------------------------------------------

 // (1.2) Put them into communication arrays
 int location;
 sLOOP {
   phiSend_n[s] = Solution[ pid(    s , nRealy-1 ) ];
   phiSend_s[s] = Solution[ pid(    s ,    2     ) ];}
 tLOOP {
   phiSend_w[t] = Solution[ pid(    2 ,    t     ) ];
   phiSend_e[t] = Solution[ pid( nRealx-1 ,    t ) ];}

 // (1.3) Send them to neighboring PEs

 if ( nei_n >= 0 )  err = MPI_Isend(phiSend_n, countx , MPI_DOUBLE , nei_n , tag
   , MPI_COMM_WORLD , &request );    // LabReplace("phiSend_n, countx ,
  MPI_DOUBLE , nei_n , tag , MPI_COMM_WORLD , &request " , "  TO-DO in LAB    " )
 if ( nei_s >= 0 )  err = MPI_Isend(phiSend_s, countx , MPI_DOUBLE , nei_s , tag
   , MPI_COMM_WORLD , &request );    // LabReplace("phiSend_s, countx ,
  MPI_DOUBLE , nei_s , tag , MPI_COMM_WORLD , &request " , "  TO-DO in LAB    " )
```

```
23  if ( nei_e >= 0 )  err = MPI_Isend(phiSend_e, county , MPI_DOUBLE , nei_e , tag
      , MPI_COMM_WORLD , &request );    // LabReplace("phiSend_e, county ,
    MPI_DOUBLE , nei_e , tag , MPI_COMM_WORLD , &request " , "  TO-DO in LAB    " )
24  if ( nei_w >= 0 )  err = MPI_Isend(phiSend_w, county , MPI_DOUBLE , nei_w , tag
      , MPI_COMM_WORLD , &request );    // LabReplace("phiSend_w, county ,
    MPI_DOUBLE , nei_w , tag , MPI_COMM_WORLD , &request " , "  TO-DO in LAB    " )
25
26  // (1.4) Receive values from neighobring PEs' physical boundaries.
27
28  if ( nei_n >= 0 ) { err = MPI_Irecv(phiRecv_n, countx , MPI_DOUBLE , nei_n ,
    tag , MPI_COMM_WORLD , &request );   MPI_Wait(&request,&status); }   //
    LabReplace("phiRecv_n, countx , MPI_DOUBLE , nei_n , tag , MPI_COMM_WORLD , &
    request ","      TO-DO in LAB   ")
29  if ( nei_s >= 0 ) { err = MPI_Irecv(phiRecv_s, countx , MPI_DOUBLE , nei_s ,
    tag , MPI_COMM_WORLD , &request );   MPI_Wait(&request,&status); }   //
    LabReplace("phiRecv_s, countx , MPI_DOUBLE , nei_s , tag , MPI_COMM_WORLD , &
    request ","      TO-DO in LAB   ")
30  if ( nei_e >= 0 ) { err = MPI_Irecv(phiRecv_e, county , MPI_DOUBLE , nei_e ,
    tag , MPI_COMM_WORLD , &request );   MPI_Wait(&request,&status); }   //
    LabReplace("phiRecv_e, county , MPI_DOUBLE , nei_e , tag , MPI_COMM_WORLD , &
    request ","      TO-DO in LAB   ")
31  if ( nei_w >= 0 ) { err = MPI_Irecv(phiRecv_w, county , MPI_DOUBLE , nei_w ,
    tag , MPI_COMM_WORLD , &request );   MPI_Wait(&request,&status); }   //
    LabReplace("phiRecv_w, county , MPI_DOUBLE , nei_w , tag , MPI_COMM_WORLD , &
    request ","      TO-DO in LAB   ")
32
33  // (1.5) If new information was received, store it in the candy-coating values
34  double phiT;
35
36  if ( nei_n >= 0 ){ sLOOP {
37      phiT = phiRecv_n[s];
38      location = pid(s, nRealy + 1);
39      Solution[location] = phiT ;
40      b[location] = phiT ;} }
41  if ( nei_s >= 0 ){sLOOP{
42      phiT = phiRecv_s[s];
43      location = pid(s,0);
44      Solution[location] = phiT ;
45      b[location] = phiT ; }}  // LabStrip
46  if ( nei_e >= 0 ){ tLOOP{
47      phiT = phiRecv_e[t];
48      location = pid(nRealx + 1, t);
49      Solution[location] = phiT ;
50      b[location] = phiT ;}}   // LabStrip
51  if ( nei_w >= 0 ){ tLOOP{
52      phiT = phiRecv_w[t];
53      location = pid(0,t);
54      Solution[location] = phiT ;
55      b[location] = phiT ;}}// LabStrip
56
```

Listing 6: Updated mpiInfo.h

```
1
2  void ParticlesOnMesh(particles &PTCL, mpiInfo &myMPI)
3  {
```

```
    double hx, hy;
    double w[5];
    int* active = &PTCL.active[0];
    double* pX = &PTCL.x[0];
    double* pY = &PTCL.y[0];
    int pN = PTCL.n;
    double* phiE = &phi[0];

    int    p[5];
    int    iL, iR, jB, jT;
    int    iPEnew, jPEnew;            // These store the i-j indicies of the processor receiving a particle,
                                      //    if that particle is leaving the mesh.
    VI ptcl_send_list, ptcl_send_PE;  // These collect information about particles that
                                      //    have left this processor and are heading onto
                                      //    another processor.

    // -
    // |
    // | Determine which particles are still on this mesh and which have left
    // |
    // -

    for ( int k = 1 ; k <= pN ; ++k )
      {
   // First, check to be sure the particle is still in the mesh.  If it is not, set its
   // "active" flag to zero, and note the processor to which it is going for MPI exchange.

   if ( active[k] == 1 )
     {
       iPEnew = myMPI.iPE;
       jPEnew = myMPI.jPE;


       if ( pX[k] < x0     ) { active[k] = -1;  iPEnew = myMPI.iPE - 1 ; }
       if ( pX[k] > x1     ) { active[k] = -1;  iPEnew = myMPI.iPE + 1 ; }
       if ( pY[k] < y0     ) { active[k] = -1;  jPEnew = myMPI.jPE - 1 ; }
       if ( pY[k] > y1     ) { active[k] = -1;  jPEnew = myMPI.jPE + 1 ; }

     }

   // The particle is not in the mesh.  Collect this particle into a holding array that will
   // be sent to the neighboring processor.

   if ( active[k] == -1)
     {
       if ( iPEnew >= 0 && iPEnew < myMPI.nPEx )
       if ( jPEnew >= 0 && jPEnew < myMPI.nPEy )
         {
     ptcl_send_list.push_back(k);
```

```
      ptcl_send_PE  .push_back(iPEnew + jPEnew * myMPI.nPEx);
         }

       active[k] = 0;  // Remove it from the list of active particles
     }
      }

    // -
    // |
    // | Give and receive particles to/with other processors
    // |
    // -

    myMPI.ParticleExchange( ptcl_send_list , ptcl_send_PE , PTCL);

    // -
    // |
    // | Accumulate particles to the nodes
    // |
    // -

    //for ( int k = 1 ; k <= nField ; ++k ) Qval[k] = 0.;

    for ( int k = 1 ; k <= pN ; ++k )
      {
   if ( active[k] == 1 )
     {
       iL = int ( (pX[k]-x0) / dx ) + 1;    // point to the left
       jB = int ( (pY[k]-y0) / dy ) + 1;    // point below
       iR = iL + 1;                             // point to the right
       jT = jB + 1;                             // point above

       // 1-2-3-4 are temporary numbers that refer to the lower left, lower right,
       // upper right, upper left, in that order.

          p[1] = pid(iL,jB);
          p[2] = pid(iR,jB);
          p[3] = pid(iR,jT);
          p[4] = pid(iL,jT);

          // Compute weights for spreading particle k to the four surrounding
    nodes.
          // Here, hx is the fractional x-distance from particle k to node 1.
    Similar for hy.
          // See [1].

          hx = (pX[k] - x[ p[1] ])/dx;
          hy = (pY[k] - y[ p[1] ])/dy;

          w[1] = ( 1. - hx ) * ( 1. - hy );
          w[2] =        hx   * ( 1. - hy );
          w[3] =        hx   *        hy  ;
          w[4] = ( 1. - hx ) *        hy  ;

          // Spread particle k to the four surrounding points
```

```
106
107       for ( int i = 1 ; i <= 4 ; ++i ) Qval[ p[i] ] +=  w[i] * PTCL.Qp;
108        }
109       }
110
111     // -
112     // |
113     // | Sum Qval on processor boundaries
114     // |
115     // -
116
117     //    iLOOP jLOOP { int p = pid(i,j) ; Qval[ p ] = x[p]; }
118
119     myMPI.PEsum(Qval);
120
121     // -
122     // |
123     // | Compute forces on particles
124     // |
125     // -
126
127     //for ( int k = 1 ; k <= PTCL.n ; ++k ) PTCL.xf[k] = PTCL.yf[k] = 0.;
128
129     for ( int k = 1 ; k <= pN ; ++k )
130       {
131    if ( active[k] == 1 )
132      {
133       iL = int ( (pX[k]-x0) / dx ) + 1;    // point to the left
134       jB = int ( (pY[k]-y0) / dy ) + 1;    // point below
135       iR = iL + 1;                             // point to the right
136       jT = jB + 1;                             // point above
137
138       // Compute a phi value for the top, bottom right, and left sides
139       // of the cell in which this particle lies by averaging the
140       // endpoints that define that side.
141       //
142       //     -->   phiT  <--   (average of top two node)
143       //     +-------------+
144       //     |             |
145       //     |        *    |
146       //     |    particle | phiR  (is the average of the two right-hand side
     nodes)
147       //     |             |
148       //     |             |
149       //     +-------------+
150       //     -->   phiB  <--  (average of bottom two nodes)
151       //
152       //
153
154       double phiT = ( phiE[ pid(iL,jT) ] + phiE[ pid(iR,jT) ] )/2.;
155       double phiB = ( phiE[ pid(iL,jB) ] + phiE[ pid(iR,jB) ] )/2.;
156       double phiL = ( phiE[ pid(iL,jT) ] + phiE[ pid(iL,jB) ] )/2.;
157       double phiR = ( phiE[ pid(iR,jT) ] + phiE[ pid(iR,jB) ] )/2.;
158
159       // Compute the gradient of the electric field using the average phi
```

```
160         // values on each of the four sides
161
162         PTCL.xf[ k ] = -0.01*( phiR - phiL ) / dx;
163         PTCL.yf[ k ] = -0.01*( phiT - phiB ) / dy;
164          }
165         }
166
167
168   }
169
170
171   //  ==
172   //  ||
173   //  ||  Utility routines
174   //  ||
175   //  ==
176
177   int pid(int i,int j) { return (i+1) + (j)*(nRealx+2); }  // Given i-j, return
      point ID.  Here i-j is the physical grid.
178                                                            // The row/col numbers
      must include the candy-coating.
179   #include "plotter.h"
180   #include "gauss_seidel.h"
181 };
```

Listing 7: Updated LaplacianOnGrid.h