Summary of changes

Modifications are shown in green, original code from the repository `tristan-mp-pu-master` is shown in red. Comments, interspersed throughout, are shown in shaded boxes (comments in red boxes should be read as warnings). The normal `diff` notation (e.g., 73,74d72, 166,187c163,165, $\cdots$) indicates the lines compared between the two files, and the action performed (d→'deleted', c→'changed', a→'added'). Each section corresponds to a file in `tristan-mp-pu-master`, and subsections indicate the modified subroutine or function.

The ghost-cell modifications have been tested in 2D and 3D, for periodic boundaries in x, y, and z. The the user file `user_weibel.F90` shows an example of necessary modifications for the Weibel problem, and similar changes would be required for other user files (not implemented).

1 Makefile

Add compile time options to specify current deposit scheme (also the associated mover). Also use the unroll option (I get some modest speedup by unrolling loops; can further customize by specifying the unroll depth, e.g. -unroll=4)

```
26c25
< CUSTOMO= -DMPI -DHDF5 -DserIO -Ddd3 -unroll
---
> CUSTOMO= -DMPI -DHDF5 -DserIO
```

This comment is added to document the different options for current deposit scheme: zigzag, 1st, 2nd, and 3rd order Esirkepov density decomposition.

```
50,53d48
< # -Dzzag use zigzag current deposit
< # -Ddd1 use 1st order density decomposition
< # -Ddd2 use 2nd order density decomposition
< # -Ddd3 use 3rd order density decomposition
```

2 fields.F90

2.1 read_input_grid()

Define two variables `nghost` and `nghostz` for the number of ghost cells in the (x, y), and z directions, respectively:

```
73,74d72
<
<         integer :: nghost, nghostz
125,126c123
<                         highorder, corr, x1in, x2in, y1in, y2in, z1in, z2in, &
<                           nghost, nghostz
---
>                         highorder, corr, x1in, x2in, y1in, y2in, z1in, z2in
```

Depending on the choice of current deposit scheme (via compile-time option `Dzzag`, `Ddd1`, `Ddd2`, or `Ddd3`), set the number of ghost cells. The choice `nghost` $= 7$ for the 2nd order current deposit is not the minimum number of cells required for a 2nd order shape function, however we keep with the convention that the number of ghost cells is odd, and in this case more than 5 ghost cells are required. The number of ghost cells at lower boundary is $\lfloor$`nghost`$/2\rfloor$, and the number of ghost cells at the upper boundary is $\lfloor$`nghost`$/2\rfloor + 1$.

```
166,187c163,165
< #ifdef zzag
<         nghost=5
<         nghostz=5
< #endif
< #ifdef dd1
<         nghost=5
<         nghostz=5
< #endif
< #ifdef dd2
<         nghost=7
<         nghostz=7
< #endif
< #ifdef dd3
<         nghost=7
<         nghostz=7
< #endif
```

In 2D, no reference is made to particle shape along the z-direction, so only the default 5 ghost cells are necessary; the Eqs. of [1], Sec. 5, are followed for the reduction to 2D.

```
< #ifdef twoD
<          nghostz=5
< #endif
```

Since `tristan-mp-pu-master` assumes 5 ghost cells, and the modified code assumes (`nghost`, `nghost`, `nghostz`) ghost cells for the (x, y, z) directions, any hardcoded reference to ghost cells should be changed; this sort of change is common in the remainder of the document, and will generally be listed without comment.

```
<       mx0=lmx0+nghost
<       my0=lmy0+nghost
---
>
>       mx0=lmx0+5
>       my0=lmy0+5
189c167
<       mz0=lmz0+nghostz
---
>       mz0=lmz0+5
```

2.2 allocate_fields()

```
237c215
<       if(modulo(int(my0-nghost),sizey).ne.0) then
---
>       if(modulo(int(my0-5),sizey).ne.0) then
240c218
<       if(modulo(int(mz0-nghostz),sizez).ne.0) then
---
>       if(modulo(int(mz0-5),sizez).ne.0) then
260,262c238,240
<               mx=(mx0-nghost)/sizex+nghost
<               my=(my0-nghost)/sizey+nghost
<               mz=(mz0-nghostz)/sizez+nghostz
---
>               mx=(mx0-5)/sizex+5
>               my=(my0-5)/sizey+5
>               mz=(mz0-5)/sizez+5
265c243
<                       if(mx0 .ne. (mx-nghost)*sizex+nghost) mx=mx0-(mx-nghost)*(sizex-1)
---
>                       if(mx0 .ne. (mx-5)*sizex+5) mx=mx0-(mx-5)*(sizex-1)
268c246
<                       if(my0 .ne. (my-nghost)*sizey+nghost) my=my0-(my-nghost)*(sizey-1)
---
>                       if(my0 .ne. (my-5)*sizey+5) my=my0-(my-5)*(sizey-1)
279c257
<                       if(mz0 .ne. (mz-nghostz)*sizez+nghostz) mz=mz0-(mz-nghostz)*(sizez-1)
---
>                       if(mz0 .ne. (mz-5)*sizez+5) mz=mz0-(mz-5)*(sizez-1)
299c277,278
<               mx0=sum(mxl(i1:i2)-nghost)+nghost
---
>               mx0=sum(mxl(i1:i2)-5)+5
>
302c281
<               my0=sum(myl(j1:j2:sizex)-nghost)+nghost
---
>               my0=sum(myl(j1:j2:sizex)-5)+5
310c289
<               mz0=sum(mzl(k1:k2:(sizex*sizey))-nghostz)+nghostz
---
>               mz0=sum(mzl(k1:k2:(sizex*sizey))-5)+5
318c297
<       mxcum=sum(mxl(i1:i2)-nghost)-(mxl(i2)-nghost)
---
>       mxcum=sum(mxl(i1:i2)-5)-(mxl(i2)-5)
322c301
<       mycum=sum(myl(j1:j2:sizex)-nghost)-(myl(j2)-nghost)
---
>       mycum=sum(myl(j1:j2:sizex)-5)-(myl(j2)-5)
327c306
<       mzcum=sum(mzl(k1:k2:(sizex*sizey))-nghostz)-(mzl(k2)-nghostz)
---
>       mzcum=sum(mzl(k1:k2:(sizex*sizey))-5)-(mzl(k2)-5)
```

When using a higher-order shape function, current may be deposited beyond the default 2 (3) ghost cells at the bottom (top) of the domain; 2 ghost cells at the bottom, and 3 at the top is enough for a 1^{st} order shape, but a particle with higher-order shape is wider, and requires more ghost cells (the exact number depends on the particle shape). Because of this, buffers used for the exchange of current must be resized. `sendbufy` and `sendbufz` are left unchanged.

```
357,360c336,339
<       allocate( bufferin1(mx,my,nghostz/2+1),bufferin2(mx,my,nghostz/2), &
<       bufferin1y(mx,nghost/2+1,mz),bufferin2y(mx,nghost/2,mz), &
<       bufferin1x(nghost/2+1,my,mz),bufferin2x(nghost/2,my,mz), &
<       sendbufy(mx,2,mz),sendbufz(mx,my,2))
---
>       allocate(bufferin1(mx,my,2),bufferin2(mx,my,2), &
>       bufferin1y(mx,2,mz),bufferin2y(mx,2,mz), &
>       bufferin1x(2,my,mz),bufferin2x(2,my,mz), &
>       sendbufy(mx,2,mz),sendbufz(mx,my,2))
365,366c344
<       call create_MPI_filter_datatypes(nghost/2+1,mx-(nghost/2+1),nghost/2+1,&
<       my-(nghost/2+1),(nghostz/2+1),mz-(nghostz/2+1))
---
>       call create_MPI_filter_datatypes(3,mx-3,3,my-3,3,mz-3)
```

2.3 iloc()

```
388,389c366,367
<  if(iloc < nghost/2+1) iloc=1
<  if(iloc > mx-(nghost/2)) iloc=mx
---
>  if(iloc < 3) iloc=1
>  if(iloc > mx-2) iloc=mx
```

2.4 jloc()

```
401,402c379,380
<  if(jloc < (nghost/2+1)) jloc=1
<  if(jloc > my-(nghost/2)) jloc=my
---
>  if(jloc < 3) jloc=1
>  if(jloc > my-2) jloc=my
```

2.5 kloc()

```
414,415c392,393
<  if(kloc < (nghostz/2+1)) kloc=1
<  if(kloc > mz-(nghostz/2)) kloc=mz
---
>  if(kloc < 3) kloc=1
>  if(kloc > mz-2) kloc=mz
```

2.6 advance_b_halfstep()

The indices for field advance are modified to extend over physical cells, in the general case $\texttt{nghost} \neq 5$.

```
600,601c578,579
<               i1=nghost/2+1
<               i2=mx-(nghost/2+1)
---
>               i1=3
>               i2=mx-3
603,604c581,582
<               i1=(nghost/2+1)
<               i2=mx-(nghost/2+1)
---
>               i1=3
>               i2=mx-3
607c585
<                       i2=mx-(nghost/2+1)
---
>                       i2=mx-3
610c588
<                       i1=(nghost/2+1)
---
>                       i1=3
621,622c599,600
<               j1=(nghost/2+1)
<               j2=my-(nghost/2+1)
---
>               j1=3
>               j2=my-3
```

```
624,625c602,603
<                 j1=(nghost/2+1)
<                 j2=my-(nghost/2+1)
---
>                 j1=3
>                 j2=my-3
628c606
<                         j2=my-(nghost/2+1)
---
>                         j2=my-3
631c609
<                         j1=(nghost/2+1)
---
>                         j1=3
645,646c623,624
<                         k1=(nghostz/2+1)
<                         k2=mz-(nghostz/2+1)
---
>                         k1=3
>                         k2=mz-3
648,649c626,627
<                         k1=(nghostz/2+1)
<                         k2=mz-(nghostz/2+1)
---
>                         k1=3
>                         k2=mz-3
652c630
<                                 k2=mz-(nghostz/2+1)
---
>                                 k2=mz-3
655c633
<                                 k1=(nghostz/2+1)
---
>                                 k1=3
```

2.7 advance_e_fullstep()

The indices for field advance are updated, similar to `advance_b_halfstep()`

```
753,754c725,726
<                 i1=(nghost/2+1)
<                 i2=mx-(nghost/2+1)
---
>                 i1=3
>                 i2=mx-3
756,757c728,729
<                 i1=(nghost/2+1)
<                 i2=mx-(nghost/2+1)
---
>                 i1=3
>                 i2=mx-3
759,760c731,732
<                         i1=(nghost/2)
<                         i2=mx-(nghost/2+1)
---
>                         i1=2
>                         i2=mx-3
763c735
<                         i1=(nghost/2+1)
---
>                         i1=3
767c739
<                         i1=(nghost/2)
---
>                         i1=2
774,775c746,747
<                 j1=(nghost/2+1)
<                 j2=my-(nghost/2+1)
---
>                 j1=3
>                 j2=my-3
777,778c749,750
<                 j1=(nghost/2+1)
<                 j2=my-(nghost/2+1)
---
>                 j1=3
>                 j2=my-3
780,781c752,753
<                         j1=(nghost/2)
<                         j2=my-(nghost/2+1)
---
```

```
>                       j1=2
>                       j2=my-3
784c756
<                       j1=(nghost/2+1)
---
>                       j1=3
788c760
<                       j1=(nghost/2)
---
>                       j1=2
799,800c771,772
<                       k1=(nghostz/2+1)
<                       k2=mz-(nghostz/2+1)
---
>                       k1=3
>                       k2=mz-3
802,803c774,775
<                       k1=(nghostz/2+1)
<                       k2=mz-(nghostz/2+1)
---
>                       k1=3
>                       k2=mz-3
805,806c777,778
<                               k1=(nghostz/2)
<                               k2=mz-(nghostz/2+1)
---
>                               k1=2
>                               k2=mz-3
809c781
<                               k1=(nghostz/2+1)
---
>                               k1=3
813c785
<                               k1=(nghostz/2)
---
>                               k1=2
818c790
<               k1=(nghostz/2)
---
>               k1=2
```

2.8 advance_b_halfstep_1D()

> We do not test any cases that use the 1D field update routines. Changes have been implemented, but not tested.

```
892,893c864,865
<               i1=(nghost/2+1)
<               i2=mx-(nghost/2+1)
---
>               i1=3
>               i2=mx-3
900,901c872,873
<               j1=(nghost/2+1)
<               j2=my-(nghost/2+1)
---
>               j1=3
>               j2=my-3
903,904c875,876
<               j1=(nghost/2+1)
<               j2=my-(nghost/2+1)
---
>               j1=3
>               j2=my-3
907c879
<                       j2=my-(nghost/2+1)
---
>                       j2=my-3
910c882
<                       j1=(nghost/2+1)
---
>                       j1=3
920,921c892,893
<               k1=(nghostz/2+1)
<               k2=mz-(nghostz/2+1)
---
>               k1=3
>               k2=mz-3
927c899
<                       k2=mz-(nghostz/2+1)
---
>                       k2=mz-3
```

```
930c902
<                       k1=(nghostz/2+1)
---
>                       k1=3
```

2.9 advance_e_fullstep_1D()

```
971,972c943,944
<               i1=(nghost/2+1)
<               i2=mx-(nghost/2+1)
---
>               i1=3
>               i2=mx-3
974c946
<               i1=(nghost/2)
---
>               i1=2
979,980c951,952
<               j1=(nghost/2+1)
<               j2=my-(nghost/2+1)
---
>               j1=3
>               j2=my-3
982,983c954,955
<               j1=(nghost/2+1)
<               j2=my-(nghost/2+1)
---
>               j1=3
>               j2=my-3
985,986c957,958
<                       j1=(nghost/2)
<                       j2=my-(nghost/2+1)
---
>                       j1=2
>                       j2=my-3
989c961
<                       j1=(nghost/2+1)
---
>                       j1=3
993c965
<                       j1=(nghost/2)
---
>                       j1=2
999,1000c971,972
<               k1=(nghostz/2+1)
<               k2=mz-(nghostz/2+1)
---
>               k1=3
>               k2=mz-3
1002,1003c974,975
<               k1=(nghostz/2+1)
<               k2=mz-(nghostz/2+1)
---
>               k1=3
>               k2=mz-3
1005,1006c977,978
<                       k1=(nghostz/2)
<                       k2=mz-(nghostz/2+1)
---
>                       k1=2
>                       k2=mz-3
1009c981
<                       k1=(nghostz/2+1)
---
>                       k1=3
1013c985
<                       k1=(nghostz/2)
---
>                       k1=2
```

2.10 advance_b_halfstep_42()

```
1055,1056c1027,1028
<               i1=nghost/2+1
<               i2=mx-(nghost/2+1)
---
>               i1=3
>               i2=mx-3
1058,1059c1030,1031
<               i1=(nghost/2)
<               i2=mx-(nghost/2)
---
>               i1=2
```

```
>                i2=mx-2
1068,1069c1040,1041
<                j1=nghost/2+1
<                j2=my-(nghost/2+1)
---
>                j1=3
>                j2=my-3
1071,1072c1043,1044
<                j1=(nghost/2)
<                  j2=my-(nghost/2)
---
>                j1=2
>                  j2=my-2
1075c1047
<                          j2=my-(nghost/2+1)
---
>                          j2=my-3
1078c1050
<                          j1=nghost/2+1
---
>                          j1=3
1082,1083c1054,1055
<                          j1=(nghost/2)
<                          j2=my-(nghost/2)
---
>                          j1=2
>                          j2=my-2
1090,1091c1062,1063
<                        k1=(nghostz/2+1)
<                        k2=mz-(nghostz/2+1)
---
>                        k1=3
>                        k2=mz-3
1093,1094c1065,1066
<                        k1=(nghostz/2+1)
<                        k2=mz-(nghostz/2+1)
---
>                        k1=3
>                        k2=mz-3
1097c1069
<                          k2=mz-(nghostz/2+1)
---
>                          k2=mz-3
1100c1072
<                          k1=(nghostz/2+1)
---
>                          k1=3
1104,1105c1076,1077
<                          k1=(nghostz/2)
<                          k2=mz-(nghostz/2)
---
>                          k1=2
>                          k2=mz-2
```

2.11 advance_e_fullstep_42()

```
1239,1240c1211,1212
<                i1=nghost/2+1
<                i2=mx-(nghost/2+1)
---
>                i1=3
>                i2=mx-3
1242c1214
<                i1=(nghost/2+1)
---
>                i1=3
1252,1253c1224,1225
<                j1=nghost/2+1
<                j2=my-(nghost/2+1)
---
>                j1=3
>                j2=my-3
1255c1227
<                j1=(nghost/2+1)
---
>                j1=2 +1
1261,1262c1233,1234
<                        k1=(nghostz/2+1)
<                        k2=mz-(nghostz/2+1)
---
>                        k1=3
>                        k2=mz-3
1266,1267c1238,1239
<                                k1=(nghostz/2+1)
<                                k2=mz-(nghostz/2+1)
```

```
---
>                               k1=2 +1
>                               k2=mz-3
1270c1242
<                               k1=(nghostz/2+1)
---
>                               k1=3
1274c1246
<                               k1=(nghostz/2)
---
>                               k1=2
```

2.12 lambda()

The function `lambda()` is defined only when using the (undocumented) option `DABSORB`, which has not been tested with the ghost cell modifications.

```
1636,1639c1609,1612
<       gc_x = 1.*(nghost/2+1) + (mx0 - 1.*nghost)/2 !x coord. of psr center
<       gc_y = 1.*(nghost/2+1) + (my0 - 1.*nghost)/2
<       gc_z = 1.*(nghostz/2+1) + (mz0 - 1.*nghostz)/2
<       rabs=(0.9*(gc_x-1.(nghost/2+1)))**2
---
>       gc_x = 3. + (mx0 - 5.)/2 !x coord. of psr center
>       gc_y = 3. + (my0 - 5.)/2
>       gc_z = 3. + (mz0 - 5.)/2
>       rabs=(0.9*(gc_x-3.))**2
1644c1617
<        rmax=gc_x - 1.*(nghost/2+1)
---
>        rmax=gc_x - 3.
```

3 fieldboundaries.F90

3.1 read_input_boundaries()

Below is a modification that is relevant when periodicx=0, however the ghost cell changes are tested only in the case of periodic boundaries in x, y, and z. Use of the ghost cell modifications and, for example, outflow boundary conditions, would require additional changes and testing (in general, the required changes should be implemented, but it would be worth double checking `deposit_particles` subroutine in `particles_movedeposit.F90` and filter subroutines in `optimized_filters.F90`; the specific user file would also require appropriate changes).

```
98,99c98,99
<       x1in=nghost/2+1 !set the location of planes where particles are removed from simulation, perpendicular to x.
<       x2in=mx0-(nghost/2)
---
>       x1in=3 !set the location of planes where particles are removed from simulation, perpendicular to x.
>       x2in=mx0-2
```

3.2 bc_b1()

The range of copied layers in the `bc_b1` routine is extended to account for higher-order shape functions. Fields in ghost cells are used not only in the field update, but also in the `mover` routines. With higher-order shape functions, the particle stencil can extend past the first layer of ghost cells, so to compute the correct force on a particle, the field should be copied into any cell over which the particle's stencil extends. For zigzag current deposit, the routine as written below is copying a layer that is not needed, unless we use the `highorder` field advance. As written below, the copy statements that would be called when using the `highorder` option are redundant, so they are commented out.

```
185d184
<         integer iter
189,192c188
<                         do iter=1,nghost/2
<                            call copy_layrx1_opt(bx,by,bz,mx,my,mz,&
<                                 nghost/2+1-iter,mx-(nghost/2+iter),mx-(nghost/2+1)+iter,nghost/2+iter)
<                         enddo
---
>                       call copy_layrx1_opt(bx,by,bz,mx,my,mz,2,mx-3,mx-2,3)
194c190
<                               !call copy_layrx1_opt(bx,by,bz,mx,my,mz,1,mx-4,mx-1,4)
---
>                               call copy_layrx1_opt(bx,by,bz,mx,my,mz,1,mx-4,mx-1,4)
```

```
197,200c193
<                           do iter=1,nghost/2
<                              call copy_layrx2_opt(bx,by,bz,mx,my,mz,&
<                                   nghost/2+1-iter,mx-(nghost/2+iter),mx-(nghost/2+1)+iter,nghost/2+iter)
<                           enddo
---
>                         call copy_layrx2_opt(bx,by,bz,mx,my,mz,2,mx-3,mx-2,3)
202c195
<                                  !call copy_layrx2_opt(bx,by,bz,mx,my,mz,1,mx-4,mx-1,4)
---
>                                  call copy_layrx2_opt(bx,by,bz,mx,my,mz,1,mx-4,mx-1,4)
207,210c200
<                           do iter=1,nghost/2
<                              call copylayrx(bx,by,bz,mx,my,mz,&
<                                   nghost/2+1-iter,mx-(nghost/2+iter),mx-(nghost/2+1)+iter,nghost/2+iter)
<                           enddo
---
>                         call copylayrx(bx,by,bz,mx,my,mz,2,mx-3,mx-2,3)
212c202
<                                  !call copylayrx(bx,by,bz,mx,my,mz,1,mx-4,mx-1,4)
---
>                                  call copylayrx(bx,by,bz,mx,my,mz,1,mx-4,mx-1,4)
215a206
>
218,221c209
<                           do iter=1,nghost/2
<                              call copy_layry1_opt(bx,by,bz,mx,my,mz,&
<                                   nghost/2+1-iter,my-(nghost/2+iter),my-(nghost/2+1)+iter,nghost/2+iter)
<                           enddo
---
>                         call copy_layry1_opt(bx,by,bz,mx,my,mz,2,my-3,my-2,3)
223c211
<                                  !call copy_layry1_opt(bx,by,bz,mx,my,mz,1,my-4,my-1,4)
---
>                                  call copy_layry1_opt(bx,by,bz,mx,my,mz,1,my-4,my-1,4)
226,229c214
<                           do iter=1,nghost/2
<                              call copy_layry2_opt(bx,by,bz,mx,my,mz,&
<                                   nghost/2+1-iter,my-(nghost/2+iter),my-(nghost/2+1)+iter,nghost/2+iter)
<                           enddo
---
>                         call copy_layry2_opt(bx,by,bz,mx,my,mz,2,my-3,my-2,3)
231c216
<                                  !call copy_layry2_opt(bx,by,bz,mx,my,mz,1,my-4,my-1,4)
---
>                                  call copy_layry2_opt(bx,by,bz,mx,my,mz,1,my-4,my-1,4)
236,239c221
<                           do iter=1,nghost/2
<                              call copylayry(bx,by,bz,mx,my,mz,&
<                                   nghost/2+1-iter,my-(nghost/2+iter),my-(nghost/2+1)+iter,nghost/2+iter)
<                           enddo
---
>                         call copylayry(bx,by,bz,mx,my,mz,2,my-3,my-2,3)
241c223
<                                  !call copylayry(bx,by,bz,mx,my,mz,1,my-4,my-1,4)
---
>                                  call copylayry(bx,by,bz,mx,my,mz,1,my-4,my-1,4)
244a227
>
246,249c229
<                 do iter=1,nghostz/2
<                    call copy_layrz1_opt(bx,by,bz,mx,my,mz,&
<                         nghostz/2+1-iter,mz-(nghostz/2+iter),mz-(nghostz/2+1)+iter,nghostz/2+iter)
<                 enddo
---
>               call copy_layrz1_opt(bx,by,bz,mx,my,mz,2,mz-3,mz-2,3)
251c231
<                         !call copy_layrz1_opt(bx,by,bz,mx,my,mz,1,mz-4,mz-1,4)
---
>                         call copy_layrz1_opt(bx,by,bz,mx,my,mz,1,mz-4,mz-1,4)
255,258c235
<                 do iter=1,nghostz/2
<                    call copy_layrz2_opt(bx,by,bz,mx,my,mz,&
<                         nghostz/2+1-iter,mz-(nghostz/2+iter),mz-(nghostz/2+1)+iter,nghostz/2+iter)
<                 enddo
---
>               call copy_layrz2_opt(bx,by,bz,mx,my,mz,2,mz-3,mz-2,3)
260c237
<                         !call copy_layrz2_opt(bx,by,bz,mx,my,mz,1,mz-4,mz-1,4)
---
>                         call copy_layrz2_opt(bx,by,bz,mx,my,mz,1,mz-4,mz-1,4)
262a240
>
```

3.3 bc_e1()

The changes here are similar to those needed in `bc_b1()`.

```
310,311d287
<          integer iter
<
314,317c290
<                          do iter=1,nghost/2
<                             call copy_layrx1_opt(ex,ey,ez,mx,my,mz,&
<                                  nghost/2+1-iter,mx-(nghost/2+iter),mx-(nghost/2+1)+iter,nghost/2+iter)
<                          enddo
---
>                        call copy_layrx1_opt(ex,ey,ez,mx,my,mz,2,mx-3,mx-2,3)
319c292
<                                !call copy_layrx1_opt(ex,ey,ez,mx,my,mz,1,mx-4,mx-1,4)
---
>                                call copy_layrx1_opt(ex,ey,ez,mx,my,mz,1,mx-4,mx-1,4)
322,325c295
<                          do iter=1,nghost/2
<                             call copy_layrx2_opt(ex,ey,ez,mx,my,mz,&
<                                  nghost/2+1-iter,mx-(nghost/2+iter),mx-(nghost/2+1)+iter,nghost/2+iter)
<                          enddo
---
>                        call copy_layrx2_opt(ex,ey,ez,mx,my,mz,2,mx-3,mx-2,3)
327c297
<                                !call copy_layrx2_opt(ex,ey,ez,mx,my,mz,1,mx-4,mx-1,4)
---
>                                call copy_layrx2_opt(ex,ey,ez,mx,my,mz,1,mx-4,mx-1,4)
332,335c302
<                          do iter=1,nghost/2
<                             call copylayrx(ex,ey,ez,mx,my,mz,&
<                                  nghost/2+1-iter,mx-(nghost/2+iter),mx-(nghost/2+1)+iter,nghost/2+iter)
<                          enddo
---
>                        call copylayrx(ex,ey,ez,mx,my,mz,2,mx-3,mx-2,3)
337c304
<                                !call copylayrx(ex,ey,ez,mx,my,mz,1,mx-4,mx-1,4)
---
>                                call copylayrx(ex,ey,ez,mx,my,mz,1,mx-4,mx-1,4)
344,347c311
<                          do iter=1,nghost/2
<                             call copy_layry1_opt(ex,ey,ez,mx,my,mz,&
<                                  nghost/2+1-iter,my-(nghost/2+iter),my-(nghost/2+1)+iter,nghost/2+iter)
<                          enddo
---
>                        call copy_layry1_opt(ex,ey,ez,mx,my,mz,2,my-3,my-2,3)
349c313
<                                !call copy_layry1_opt(ex,ey,ez,mx,my,mz,1,my-4,my-1,4)
---
>                                call copy_layry1_opt(ex,ey,ez,mx,my,mz,1,my-4,my-1,4)
352,356c316
<                          do iter=1,nghost/2
<                             call copy_layry2_opt(ex,ey,ez,mx,my,mz,&
<                                  nghost/2+1-iter,my-(nghost/2+iter),my-(nghost/2+1)+iter,nghost/2+iter)
<                          enddo
<                          !call copy_layry2_opt(ex,ey,ez,mx,my,mz,nghost/2-4,my-(nghost/2+5),my-(nghost/2-4),(nghost/2+5))
---
>                        call copy_layry2_opt(ex,ey,ez,mx,my,mz,2,my-3,my-2,3)
358c318
<                                !call copy_layry2_opt(ex,ey,ez,mx,my,mz,1,my-4,my-1,4)
---
>                                call copy_layry2_opt(ex,ey,ez,mx,my,mz,1,my-4,my-1,4)
363,366c323
<                          do iter=1,nghost/2
<                             call copylayry(ex,ey,ez,mx,my,mz,&
<                                  nghost/2+1-iter,my-(nghost/2+iter),my-(nghost/2+1)+iter,nghost/2+iter)
<                          enddo
---
>                        call copylayry(ex,ey,ez,mx,my,mz,2,my-3,my-2,3)
368c325
<                                !call copylayry(ex,ey,ez,mx,my,mz,1,my-4,my-1,4)
---
>                                call copylayry(ex,ey,ez,mx,my,mz,1,my-4,my-1,4)
374,377c331
<                do iter=1,nghostz/2
<                   call copy_layrz1_opt(ex,ey,ez,mx,my,mz,&
<                        nghostz/2+1-iter,mz-(nghostz/2+iter),mz-(nghostz/2+1)+iter,nghostz/2+iter)
<                enddo
---
>              call copy_layrz1_opt(ex,ey,ez,mx,my,mz,2,mz-3,mz-2,3)
379c333
<                       !call copy_layrz1_opt(ex,ey,ez,mx,my,mz,1,mz-4,mz-1,4)
```

```
---
>                       call copy_layrz1_opt(ex,ey,ez,mx,my,mz,1,mz-4,mz-1,4)
383,386c337
<                   do iter=1,nghostz/2
<                      call copy_layrz2_opt(ex,ey,ez,mx,my,mz,&
<                           nghostz/2+1-iter,mz-(nghostz/2+iter),mz-(nghostz/2+1)+iter,nghostz/2+iter)
<                   enddo
---
>                 call copy_layrz2_opt(ex,ey,ez,mx,my,mz,2,mz-3,mz-2,3)
388c339
<                       !call copy_layrz2_opt(ex,ey,ez,mx,my,mz,1,mz-4,mz-1,4)
---
>                       call copy_layrz2_opt(ex,ey,ez,mx,my,mz,1,mz-4,mz-1,4)
```

3.4 copy_layrz2()

```
1573c1524
<       call MPI_SendRecv(by(:,:,mz-(nghostz/2+1)),count,mpi_read,uprank,uptag, &
---
>       call MPI_SendRecv(by(:,:,mz-3),count,mpi_read,uprank,uptag, &
1577c1528
<       if(rank/(sizex*sizey) .ne. 0) by(:,:,nghostz/2)=bufferin
---
>       if(rank/(sizex*sizey) .ne. 0) by(:,:,2)=bufferin
```

3.5 exchange_current()

The indices for exchange current are generalized to `nghost` $\neq 5$; also, current from **all** ghost cells is deposited into physical cells (the final ghost cell at the top of the domain is not ignored). Without this change, current that is deposited in the last ghost cell at the top of the domain will be discarded. This change necessitates separate variables `count, count2` corresponding to transfer of ghost cells from the top or bottom of the domain, respectively.

```
1774c1725
<       integer :: uprank, dwnrank, comm,count,count2,uptag,dwntag,ierr, &
---
>       integer :: uprank, dwnrank, comm,count,uptag,dwntag,ierr, &
1798,1811c1748,1761
<                 bufferin1x(1:nghost/2+1,:,:)=curx((mx-nghost/2):(mx),:,:)
<                 bufferin2x(1:nghost/2,:,:)=curx(1:nghost/2,:,:)
<                 curx((nghost/2+1):(nghost),:,:)=curx((nghost/2+1):(nghost),:,:)+bufferin1x(1:nghost/2+1,:,:)
<                 curx((mx-(nghost-1)):(mx-(nghost/2+1)),:,:)=curx(&
<                         (mx-(nghost-1)):(mx-(nghost/2+1)),:,:)+bufferin2x(1:nghost/2,:,:)
<
<                 bufferin1x(1:nghost/2+1,:,:)=cury((mx-nghost/2):(mx),:,:)
<                 bufferin2x(1:nghost/2,:,:)=cury(1:nghost/2,:,:)
<                 cury((nghost/2+1):(nghost),:,:)=cury((nghost/2+1):(nghost),:,:)+bufferin1x(1:nghost/2+1,:,:)
<                 cury(mx-(nghost-1):(mx-(nghost/2+1)),:,:)=cury(&
<                        mx-(nghost-1):(mx-(nghost/2+1)),:,:)+bufferin2x(1:nghost/2,:,:)
<
<                 bufferin1x(1:nghost/2+1,:,:)=curz((mx-nghost/2):(mx),:,:)
<                 bufferin2x(1:nghost/2,:,:)=curz(1:nghost/2,:,:)
<                 curz((nghost/2+1):(nghost),:,:)=curz((nghost/2+1):(nghost),:,:)+bufferin1x(1:nghost/2+1,:,:)
<                 curz((mx-(nghost-1)):(mx-(nghost/2+1)),:,:)=curz(&
<                         (mx-(nghost-1)):(mx-(nghost/2+1)),:,:)+bufferin2x(1:nghost/2,:,:)
---
>                 bufferin1x(1:2,:,:)=curx(mx-2:mx-1,:,:)
>                 bufferin2x(1:2,:,:)=curx(1:2,:,:)
>                 curx(3:4,:,:)=curx(3:4,:,:)+bufferin1x(1:2,:,:)
>                 curx(mx-4:mx-3,:,:)=curx(mx-4:mx-3,:,:)+bufferin2x(1:2,:,:)
>
>                 bufferin1x(1:2,:,:)=cury(mx-2:mx-1,:,:)
>                 bufferin2x(1:2,:,:)=cury(1:2,:,:)
>                 cury(3:4,:,:)=cury(3:4,:,:)+bufferin1x(1:2,:,:)
>                 cury(mx-4:mx-3,:,:)=cury(mx-4:mx-3,:,:)+bufferin2x(1:2,:,:)
>
>                 bufferin1x(1:2,:,:)=curz(mx-2:mx-1,:,:)
>                 bufferin2x(1:2,:,:)=curz(1:2,:,:)
>                 curz(3:4,:,:)=curz(3:4,:,:)+bufferin1x(1:2,:,:)
>                 curz(mx-4:mx-3,:,:)=curz(mx-4:mx-3,:,:)+bufferin2x(1:2,:,:)
1832,1833c1782
<                 count=(j2-j1+1)*(nghost/2+1)
<                   count2=(j2-j1+1)*(nghost/2)
---
>                 count=(j2-j1+1)*2
1837,1838c1786
<                 count=(j2-j1+1)*(k2-k1+1)*(nghost/2+1)
<                   count2=(i2-i1+1)*(k2-k1+1)*(nghost/2)
---
>                 count=(j2-j1+1)*(k2-k1+1)*2
1841,1842c1789,1790
```

```
<                call MPI_SendRecv(curx((mx-nghost/2):mx,j1:j2,k1:k2),count,mpi_read,plusrank &
<                ,plustag,bufferin1x(1:nghost/2+1,j1:j2,k1:k2), count,mpi_read,minusrank,plustag, &
---
>                call MPI_SendRecv(curx(mx-2:mx-1,j1:j2,k1:k2),count,mpi_read,plusrank &
>                ,plustag,bufferin1x(1:2,j1:j2,k1:k2), count,mpi_read,minusrank,plustag, &
1847,1850c1795
<                if(iperiodic) then
<                      curx((nghost/2+1):(nghost),j1:j2,k1:k2)=curx((nghost/2+1):(nghost),j1:j2,k1:k2)&
<                           +bufferin1x(1:nghost/2+1,j1:j2,k1:k2)
<                  endif
---
>                if(iperiodic) curx(3:4,j1:j2,k1:k2)=curx(3:4,j1:j2,k1:k2)+bufferin1x(1:2,j1:j2,k1:k2)
1854,1855c1799,1800
<                call MPI_SendRecv(curx(1:nghost/2,j1:j2,k1:k2),count2,mpi_read,minusrank,minustag, &
<                bufferin2x(1:nghost/2,j1:j2,k1:k2),count2,mpi_read,plusrank,minustag, comm &
---
>                call MPI_SendRecv(curx(1:2,j1:j2,k1:k2),count,mpi_read,minusrank,minustag, &
>                bufferin2x(1:2,j1:j2,k1:k2),count,mpi_read,plusrank,minustag, comm &
1860,1863c1805
<                if(iperiodic) then
<                      curx(mx-(nghost-1):(mx-(nghost/2+1)),j1:j2,k1:k2)=curx(mx-(nghost-1):(mx-(nghost/2+1)),j1:j2,k1:k2)&
<                           +bufferin2x(1:nghost/2,j1:j2,k1:k2)
<                  endif
---
>                if(iperiodic) curx(mx-4:mx-3,j1:j2,k1:k2)=curx(mx-4:mx-3,j1:j2,k1:k2)+bufferin2x(1:2,j1:j2,k1:k2)
1872,1873c1814,1815
<                call MPI_SendRecv(cury((mx-nghost/2):mx,j1:j2,k1:k2),count,mpi_read,plusrank &
<                ,plustag,bufferin1x(1:nghost/2+1,j1:j2,k1:k2), count,mpi_read,minusrank,plustag, &
---
>                call MPI_SendRecv(cury(mx-2:mx-1,j1:j2,k1:k2),count,mpi_read,plusrank &
>                ,plustag,bufferin1x(1:2,j1:j2,k1:k2), count,mpi_read,minusrank,plustag, &
1878,1881c1820,1821
<                if(iperiodic) then
<                      cury((nghost/2+1):(nghost),j1:j2,k1:k2)=cury((nghost/2+1):(nghost),j1:j2,k1:k2) &
<                           +bufferin1x(1:nghost/2+1,j1:j2,k1:k2)
<                  endif
---
>                if(iperiodic) cury(3:4,j1:j2,k1:k2)=cury(3:4,j1:j2,k1:k2)+bufferin1x(1:2,j1:j2,k1:k2)
>
1884,1885c1824,1825
<                call MPI_SendRecv(cury(1:nghost/2,j1:j2,k1:k2),count2,mpi_read,minusrank,minustag, &
<                bufferin2x(1:nghost/2,j1:j2,k1:k2),count2,mpi_read,plusrank,minustag, comm &
---
>                call MPI_SendRecv(cury(1:2,j1:j2,k1:k2),count,mpi_read,minusrank,minustag, &
>                bufferin2x(1:2,j1:j2,k1:k2),count,mpi_read,plusrank,minustag, comm &
1890,1893c1830
<                if(iperiodic) then
<                      cury(mx-(nghost-1):(mx-(nghost/2+1)),j1:j2,k1:k2)=cury(mx-(nghost-1):(mx-(nghost/2+1)),j1:j2,k1:k2) &
<                           +bufferin2x(1:nghost/2,j1:j2,k1:k2)
<                   endif
---
>                if(iperiodic) cury(mx-4:mx-3,j1:j2,k1:k2)=cury(mx-4:mx-3,j1:j2,k1:k2)+bufferin2x(1:2,j1:j2,k1:k2)
1900,1901c1837,1838
<                call MPI_SendRecv(curz((mx-nghost/2):(mx),j1:j2,k1:k2),count,mpi_read,plusrank &
<                ,plustag,bufferin1x(1:nghost/2+1,j1:j2,k1:k2), count,mpi_read,minusrank,plustag, &
---
>                call MPI_SendRecv(curz(mx-2:mx-1,j1:j2,k1:k2),count,mpi_read,plusrank &
>                ,plustag,bufferin1x(1:2,j1:j2,k1:k2), count,mpi_read,minusrank,plustag, &
1906,1909c1843
<                if(iperiodic) then
<                      curz((nghost/2+1):(nghost),j1:j2,k1:k2)=curz((nghost/2+1):(nghost),j1:j2,k1:k2)&
<                           +bufferin1x(1:nghost/2+1,j1:j2,k1:k2)
<                  endif
---
>                if(iperiodic) curz(3:4,j1:j2,k1:k2)=curz(3:4,j1:j2,k1:k2)+bufferin1x(1:2,j1:j2,k1:k2)
1913,1914c1847,1848
<                call MPI_SendRecv(curz(1:nghost/2,j1:j2,k1:k2),count2,mpi_read,minusrank,minustag, &
<                bufferin2x(1:nghost/2,j1:j2,k1:k2),count2,mpi_read,plusrank,minustag, comm &
---
>                call MPI_SendRecv(curz(1:2,j1:j2,k1:k2),count,mpi_read,minusrank,minustag, &
>                bufferin2x(1:2,j1:j2,k1:k2),count,mpi_read,plusrank,minustag, comm &
1919,1922c1853
<                if(iperiodic) then
<                      curz(mx-(nghost-1):(mx-(nghost/2+1)),j1:j2,k1:k2)=curz(mx-(nghost-1):(mx-(nghost/2+1)),j1:j2,k1:k2)&
<                           +bufferin2x(1:nghost/2,j1:j2,k1:k2)
<                  endif
---
>                if(iperiodic) curz(mx-4:mx-3,j1:j2,k1:k2)=curz(mx-4:mx-3,j1:j2,k1:k2)+bufferin2x(1:2,j1:j2,k1:k2)
1933,1946c1864,1877
<                bufferin1y(:,1:nghost/2+1,:)=curx(:,(my-(nghost/2)):(my),:)
<                bufferin2y(:,1:nghost/2,:)=curx(:,1:nghost/2,:)
<                curx(:,(nghost/2+1):(nghost),:)=curx(:,(nghost/2+1):(nghost),:)+bufferin1y(:,1:nghost/2+1,:)
<                curx(:,(my-(nghost-1)):(my-(nghost/2+1)),:)=curx(:,(my-(nghost-1)):(my-(nghost/2+1)),:)&
<                       +bufferin2y(:,1:nghost/2,:)
<
```

```
<                bufferin1y(:,1:nghost/2,+1:)=cury(:,(my-(nghost/2)):(my),:)
<                bufferin2y(:,1:nghost/2,:)=cury(:,1:nghost/2,:)
<                cury(:,(nghost/2+1):(nghost),:)=cury(:,(nghost/2+1):(nghost),:)+bufferin1y(:,1:nghost/2+1,:)
<                cury(:,(my-(nghost-1)):(my-(nghost/2+1)),:)=cury(:,(my-(nghost-1)):(my-(nghost/2+1)),:)&
<                         +bufferin2y(:,1:nghost/2,:)
<
<                bufferin1y(:,1:nghost/2+1,:)=curz(:,(my-(nghost/2)):(my),:)
<                bufferin2y(:,1:nghost/2,:)=curz(:,1:nghost/2,:)
<                curz(:,(nghost/2+1):(nghost),:)=curz(:,(nghost/2+1):(nghost),:)+bufferin1y(:,1:nghost/2+1,:)
<                curz(:,(my-(nghost-1)):(my-(nghost/2+1)),:)=curz(:,(my-(nghost-1)):(my-(nghost/2+1)),:)&
<                        +bufferin2y(:,1:nghost/2,:)
---
>                bufferin1y(:,1:2,:)=curx(:,my-2:my-1,:)
>                bufferin2y(:,1:2,:)=curx(:,1:2,:)
>                curx(:,3:4,:)=curx(:,3:4,:)+bufferin1y(:,1:2,:)
>                curx(:,my-4:my-3,:)=curx(:,my-4:my-3,:)+bufferin2y(:,1:2,:)
>
>                bufferin1y(:,1:2,:)=cury(:,my-2:my-1,:)
>                bufferin2y(:,1:2,:)=cury(:,1:2,:)
>                cury(:,3:4,:)=cury(:,3:4,:)+bufferin1y(:,1:2,:)
>                cury(:,my-4:my-3,:)=cury(:,my-4:my-3,:)+bufferin2y(:,1:2,:)
>
>                bufferin1y(:,1:2,:)=curz(:,my-2:my-1,:)
>                bufferin2y(:,1:2,:)=curz(:,1:2,:)
>                curz(:,3:4,:)=curz(:,3:4,:)+bufferin1y(:,1:2,:)
>                curz(:,my-4:my-3,:)=curz(:,my-4:my-3,:)+bufferin2y(:,1:2,:)
1981,1982c1913
<                count=(i2-i1+1)*(nghost/2+1)
<                  count2=(i2-i1+1)*(nghost/2)
---
>                count=(i2-i1+1)*2
1986,1987c1917
<                count=(i2-i1+1)*(k2-k1+1)*(nghost/2+1)
<                  count2=(i2-i1+1)*(k2-k1+1)*(nghost/2)
---
>                count=(i2-i1+1)*(k2-k1+1)*2
1990,1991c1920,1921
<                call MPI_SendRecv(curx(i1:i2,(my-(nghost/2)):(my),k1:k2),count,mpi_read,rgtrank &
<                ,rgttag,bufferin1y(i1:i2,1:nghost/2+1,k1:k2), count,mpi_read,lftrank,rgttag, &
---
>                call MPI_SendRecv(curx(i1:i2,my-2:my-1,k1:k2),count,mpi_read,rgtrank &
>                ,rgttag,bufferin1y(i1:i2,1:2,k1:k2), count,mpi_read,lftrank,rgttag, &
1997,2000c1927
<                if(iperiodic) then
<                      curx(i1:i2,(nghost/2+1):(nghost),k1:k2)=curx(i1:i2,(nghost/2+1):(nghost),k1:k2)&
<                           +bufferin1y(i1:i2,1:nghost/2+1,k1:k2)
<                  endif
---
>                if(iperiodic) curx(i1:i2,3:4,k1:k2)=curx(i1:i2,3:4,k1:k2)+bufferin1y(i1:i2,1:2,k1:k2)
2004,2005c1931,1932
<                call MPI_SendRecv(curx(i1:i2,1:nghost/2,k1:k2),count2,mpi_read,lftrank,lfttag, &
<                bufferin2y(i1:i2,1:nghost/2,k1:k2),count2,mpi_read,rgtrank,lfttag, comm &
---
>                call MPI_SendRecv(curx(i1:i2,1:2,k1:k2),count,mpi_read,lftrank,lfttag, &
>                bufferin2y(i1:i2,1:2,k1:k2),count,mpi_read,rgtrank,lfttag, comm &
2011,2014c1938
<                if(iperiodic) then
<                      curx(i1:i2,(my-(nghost-1)):(my-(nghost/2+1)),k1:k2)=curx(i1:i2,(my-(nghost-1)):(my-(nghost/2+1)),k1:k2)&
<                           +bufferin2y(i1:i2,1:nghost/2,k1:k2)
<                  endif
---
>                if(iperiodic) curx(i1:i2,my-4:my-3,k1:k2)=curx(i1:i2,my-4:my-3,k1:k2)+bufferin2y(i1:i2,1:2,k1:k2)
2022,2023c1946,1947
<                call MPI_SendRecv(cury(i1:i2,(my-(nghost/2)):(my),k1:k2),count,mpi_read,rgtrank &
<                ,rgttag,bufferin1y(i1:i2,1:nghost/2+1,k1:k2), count,mpi_read,lftrank,rgttag, &
---
>                call MPI_SendRecv(cury(i1:i2,my-2:my-1,k1:k2),count,mpi_read,rgtrank &
>                ,rgttag,bufferin1y(i1:i2,1:2,k1:k2), count,mpi_read,lftrank,rgttag, &
2029,2033c1953,1954
<                if(iperiodic) then
<                      cury(i1:i2,(nghost/2+1):(nghost),k1:k2)=cury(i1:i2,(nghost/2+1):(nghost),k1:k2) &
<                           +bufferin1y(i1:i2,1:nghost/2+1,k1:k2)
<                   endif
<
---
>                if(iperiodic) cury(i1:i2,3:4,k1:k2)=cury(i1:i2,3:4,k1:k2)+bufferin1y(i1:i2,1:2,k1:k2)
>
2037,2038c1958,1959
<                call MPI_SendRecv(cury(i1:i2,1:nghost/2,k1:k2),count2,mpi_read,lftrank,lfttag, &
<                bufferin2y(i1:i2,1:nghost/2,k1:k2),count2,mpi_read,rgtrank,lfttag, comm,status &
---
>                call MPI_SendRecv(cury(i1:i2,1:2,k1:k2),count,mpi_read,lftrank,lfttag, &
>                bufferin2y(i1:i2,1:2,k1:k2),count,mpi_read,rgtrank,lfttag, comm,status &
2044,2047c1965
<                if(iperiodic) then
```

```
<                     cury(i1:i2,(my-(nghost-1)):(my-(nghost/2+1)),k1:k2)=cury(i1:i2,(my-(nghost-1)):(my-(nghost/2+1)),k1:k2)&
<                          +bufferin2y(i1:i2,1:nghost/2,k1:k2)
<                   endif
---
>                 if(iperiodic) cury(i1:i2,my-4:my-3,k1:k2)=cury(i1:i2,my-4:my-3,k1:k2)+bufferin2y(i1:i2,1:2,k1:k2)
2053,2054c1971,1972
<                 call MPI_SendRecv(curz(i1:i2,(my-(nghost/2)):(my),k1:k2),count,mpi_read,rgtrank &
<                 ,rgttag,bufferin1y(i1:i2,1:nghost/2+1,k1:k2),count,mpi_read,lftrank,rgttag &
---
>                 call MPI_SendRecv(curz(i1:i2,my-2:my-1,k1:k2),count,mpi_read,rgtrank &
>                 ,rgttag,bufferin1y(i1:i2,1:2,k1:k2),count,mpi_read,lftrank,rgttag &
2060,2063c1978
<                 if(iperiodic) then
<                       curz(i1:i2,(nghost/2+1):(nghost),k1:k2)=curz(i1:i2,(nghost/2+1):(nghost),k1:k2)&
<                            +bufferin1y(i1:i2,1:nghost/2+1,k1:k2)
<                   endif
---
>                 if(iperiodic) curz(i1:i2,3:4,k1:k2)=curz(i1:i2,3:4,k1:k2)+bufferin1y(i1:i2,1:2,k1:k2)
2069,2070c1984,1985
<                 call MPI_SendRecv(curz(i1:i2,1:nghost/2,k1:k2),count2,mpi_read,lftrank,lfttag, &
<                 bufferin2y(i1:i2,1:nghost/2,k1:k2),count2,mpi_read,rgtrank,lfttag, &
---
>                 call MPI_SendRecv(curz(i1:i2,1:2,k1:k2),count,mpi_read,lftrank,lfttag, &
>                 bufferin2y(i1:i2,1:2,k1:k2),count,mpi_read,rgtrank,lfttag, &
2076,2079c1991
<                 if(iperiodic) then
<                       curz(i1:i2,(my-(nghost-1)):(my-(nghost/2+1)),k1:k2)=curz(i1:i2,(my-(nghost-1)):(my-(nghost/2+1)),k1:k2)&
<                            +bufferin2y(i1:i2,1:nghost/2,k1:k2)
<                   endif
---
>                 if(iperiodic) curz(i1:i2,my-4:my-3,k1:k2)=curz(i1:i2,my-4:my-3,k1:k2)+bufferin2y(i1:i2,1:2,k1:k2)
2101,2102c2013
<                 count=(i2-i1+1)*(j2-j1+1)*(nghostz/2+1)
<                   count2=(i2-i1+1)*(j2-j1+1)*(nghostz/2)
---
>                 count=(i2-i1+1)*(j2-j1+1)*2
2108,2109c2019,2020
<                 call MPI_SendRecv(curx(i1:i2,j1:j2,mz-(nghostz/2):mz),count,mpi_read,uprank,uptag, &
<                 bufferin1(i1:i2,j1:j2,1:nghostz/2+1),count,mpi_read,dwnrank,uptag, comm &
---
>                 call MPI_SendRecv(curx(i1:i2,j1:j2,mz-2:mz-1),count,mpi_read,uprank,uptag, &
>                 bufferin1(i1:i2,j1:j2,1:2),count,mpi_read,dwnrank,uptag, comm &
2114,2117c2025
<                 if(iperiodic) then
<                       curx(i1:i2,j1:j2,(nghostz/2+1):(nghostz))=curx(i1:i2,j1:j2,(nghostz/2+1):(nghostz))&
<                            +bufferin1(i1:i2,j1:j2,1:nghostz/2+1)
<                   endif
---
>                 if(iperiodic)curx(i1:i2,j1:j2,3:4)=curx(i1:i2,j1:j2,3:4)+bufferin1(i1:i2,j1:j2,1:2)
2121,2122c2029,2030
<                 call MPI_SendRecv(curx(i1:i2,j1:j2,1:nghostz/2),count2,mpi_read,dwnrank,dwntag, &
<                 bufferin2(i1:i2,j1:j2,1:nghostz/2),count2,mpi_read,uprank,dwntag, comm,status &
---
>                 call MPI_SendRecv(curx(i1:i2,j1:j2,1:2),count,mpi_read,dwnrank,dwntag, &
>                 bufferin2(i1:i2,j1:j2,1:2),count,mpi_read,uprank,dwntag, comm,status &
2127,2130c2035,2036
<                 if(iperiodic) then
<                       curx(i1:i2,j1:j2,(mz-(nghostz-1)):(mz-(nghostz/2+1)))=curx(i1:i2,j1:j2,&
<                            (mz-(nghostz-1)):(mz-(nghostz/2+1)))+bufferin2(i1:i2,j1:j2,1:nghostz/2)
<                   endif
---
>                 if(iperiodic) curx(i1:i2,j1:j2,mz-4:mz-3)=curx(i1:i2,j1:j2,mz-4:mz-3)+bufferin2(i1:i2,j1:j2,1:2)
>
2135,2136c2041,2042
<                 call MPI_SendRecv(cury(i1:i2,j1:j2,mz-(nghostz/2):mz),count,mpi_read,uprank,uptag, &
<                 bufferin1(i1:i2,j1:j2,1:(nghostz/2+1)), count,mpi_read,dwnrank,uptag, comm,status &
---
>                 call MPI_SendRecv(cury(i1:i2,j1:j2,mz-2:mz-1),count,mpi_read,uprank,uptag, &
>                 bufferin1(i1:i2,j1:j2,1:2), count,mpi_read,dwnrank,uptag, comm,status &
2141,2144c2047
<                 if(iperiodic) then
<                       cury(i1:i2,j1:j2,(nghostz/2+1):(nghostz))=cury(i1:i2,j1:j2,(nghostz/2+1):nghostz) &
<                            +bufferin1(i1:i2,j1:j2,1:(nghostz/2+1))
<                   endif
---
>                 if(iperiodic)cury(i1:i2,j1:j2,3:4)=cury(i1:i2,j1:j2,3:4)+bufferin1(i1:i2,j1:j2,1:2)
2148,2149c2051,2052
<                 call MPI_SendRecv(cury(i1:i2,j1:j2,1:(nghostz/2)),count2,mpi_read,dwnrank,dwntag, &
<                 bufferin2(i1:i2,j1:j2,1:(nghostz/2)),count2,mpi_read,uprank,dwntag, comm,status &
---
>                 call MPI_SendRecv(cury(i1:i2,j1:j2,1:2),count,mpi_read,dwnrank,dwntag, &
>                 bufferin2(i1:i2,j1:j2,1:2),count,mpi_read,uprank,dwntag, comm,status &
2154,2157c2057
<                   if(iperiodic) then
<                      cury(i1:i2,j1:j2,mz-(nghostz-1):mz-(nghostz/2+1))=cury(i1:i2,j1:j2,mz-(nghostz-1):mz-(nghostz/2+1)) &
```

```
<                         +bufferin2(i1:i2,j1:j2,1:nghostz/2)
<                 endif
---
>                 if(iperiodic) cury(i1:i2,j1:j2,mz-4:mz-3)=cury(i1:i2,j1:j2,mz-4:mz-3)+bufferin2(i1:i2,j1:j2,1:2)
2163,2164c2063,2064
<                 call MPI_SendRecv(curz(i1:i2,j1:j2,mz-(nghostz/2):mz),count,mpi_read,uprank,uptag, &
<                 bufferin1(i1:i2,j1:j2,1:(nghostz/2+1)),count,mpi_read,dwnrank,uptag, &
---
>                 call MPI_SendRecv(curz(i1:i2,j1:j2,mz-2:mz-1),count,mpi_read,uprank,uptag, &
>                 bufferin1(i1:i2,j1:j2,1:2),count,mpi_read,dwnrank,uptag, &
2169,2172c2069
<                 if(iperiodic) then
<                       curz(i1:i2,j1:j2,(nghostz/2+1):nghostz)=curz(i1:i2,j1:j2,(nghostz/2+1):nghostz) &
<                            +bufferin1(i1:i2,j1:j2,1:(nghostz/2+1))
<                 endif
---
>                 if(iperiodic)curz(i1:i2,j1:j2,3:4)=curz(i1:i2,j1:j2,3:4)+bufferin1(i1:i2,j1:j2,1:2)
2176,2177c2073,2074
<                 call MPI_SendRecv(curz(i1:i2,j1:j2,1:nghostz/2),count2,mpi_read,dwnrank,dwntag, &
<                 bufferin2(i1:i2,j1:j2,1:nghostz/2),count2,mpi_read,uprank,dwntag, &
---
>                 call MPI_SendRecv(curz(i1:i2,j1:j2,1:2),count,mpi_read,dwnrank,dwntag, &
>                 bufferin2(i1:i2,j1:j2,1:2),count,mpi_read,uprank,dwntag, &
2182,2185c2079
<                 if(iperiodic) then
<                       curz(i1:i2,j1:j2,mz-(nghostz-1):mz-(nghostz/2+1))=curz(i1:i2,j1:j2,mz-(nghostz-1):mz-(nghostz/2+1)) &
<                            +bufferin2(i1:i2,j1:j2,1:nghostz/2)
<                 endif
---
>                 if(iperiodic) curz(i1:i2,j1:j2,mz-4:mz-3)=curz(i1:i2,j1:j2,mz-4:mz-3)+bufferin2(i1:i2,j1:j2,1:2)
```

3.6 apply_filter1_opt()

The indices for `filter1` are changed to extend over physical cells, in the general case `nghost` $\neq 5$.

```
2533,2534c2427,2428
<       istr=nghost/2  +1
<       ifin=mx-1 -(nghost/2)
---
>       istr=2  +1
>       ifin=mx-1 -2
2566,2567c2460,2461
< !                     call copy_layrx1(curx,cury,curz,mx,my,mz,nghost/2,mx-(nghost/2+1),mx-nghost/2,nghost/2+1)
<                       call copy_layrx1_opt(curx,cury,curz,mx,my,mz,nghost/2,mx-(nghost/2+1),mx-nghost/2,nghost/2+1)
---
> !                     call copy_layrx1(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
>                       call copy_layrx1_opt(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
2569,2570c2463,2464
< !                     call copy_layrx2(curx,cury,curz,mx,my,mz,nghost/2,mx-(nghost/2+1),mx-nghost/2,nghost/2+1)
<                       call copy_layrx2_opt(curx,cury,curz,mx,my,mz,nghost/2,mx-(nghost/2+1),mx-nghost/2,nghost/2+1)
---
> !                     call copy_layrx2(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
>                       call copy_layrx2_opt(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
2575c2469
<                 do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                 do k=3,mz-3
2579c2473
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2587c2481
<                 do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                 do k=3,mz-3
2591c2485
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2602c2496
<                 do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                 do k=3,mz-3
2606c2500
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2614c2508
<                 do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                 do k=3,mz-3
```

```
2618c2512
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2626c2520
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2630c2524
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2638c2532
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2642c2536
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2652,2653c2546,2547
< !                     call copy_layry1(curx,cury,curz,mx,my,mz,nghost/2,my-(nghost/2+1),my-(nghost/2),nghost/2+1)
<                       call copy_layry1_opt(curx,cury,curz,mx,my,mz,nghost/2,my-(nghost/2+1),my-(nghost/2),nghost/2+1)
---
> !                     call copy_layry1(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
>                       call copy_layry1_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
2655,2656c2549,2550
< !                     call copy_layry2(curx,cury,curz,mx,my,mz,nghost/2,my-(nghost/2+1),my-(nghost/2),nghost/2+1)
<                       call copy_layry2_opt(curx,cury,curz,mx,my,mz,nghost/2,my-(nghost/2+1),my-(nghost/2),nghost/2+1)
---
> !                     call copy_layry2(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
>                       call copy_layry2_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
2660c2554
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2664c2558
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2672c2566
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2676c2570
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2684c2578
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2688c2582
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2696c2590
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2700c2594
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2708c2602
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2712c2606
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
2720c2614
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
2724c2618
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
```

The eighth and ninth arguments of `copy_layrz1_opt` look like a typo in `apply_filter1_opt()` of `tristan-mp-pu-master` (at least the version we're using: see `2735,2736c2629` directly below). It is only called in 3D when using `filter1` and `periodicz` $= 1$, but potentially a problem. This is updated in the modified version.

```
2735,2736c2629
<
<                               call copy_layrz1_opt(curx,cury,curz,mx,my,mz,nghostz/2,&
<                                                    mz-(nghostz/2+1),mz-(nghostz/2),nghostz/2+1)
---
>                               call copy_layrz1_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
2740c2633
<                              call copy_layrz2_opt(curx,cury,curz,mx,my,mz,(nghostz/2),&
<                                                    mz-(nghostz/2+1),mz-(nghostz/2),(nghostz/2+1))
---
>                              call copy_layrz2_opt(curx,cury,curz,mx,my,mz,2,mz-3,mz-2,3)
2743,2744c2636,2637
<                         do k=(nghostz/2+1),mz-(nghostz/2+1)
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                         do k=3,mz-3
>                                 do j=3,my-3
2751,2752c2644,2645
<                         do k=(nghostz/2+1),mz-(nghostz/2+1)
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                         do k=3,mz-3
>                                 do j=3,my-3
2759,2760c2652,2653
<                         do k=(nghostz/2+1),mz-(nghostz/2+1)
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                         do k=3,mz-3
>                                 do j=3,my-3
2767,2768c2660,2661
<                         do k=(nghostz/2+1),mz-(nghostz/2+1)
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                         do k=3,mz-3
>                                 do j=3,my-3
2775,2776c2668,2669
<                         do k=(nghostz/2+1),mz-(nghostz/2+1)
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                         do k=3,mz-3
>                                 do j=3,my-3
2783,2784c2676,2677
<                         do k=(nghostz/2+1),mz-(nghostz/2+1)
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                         do k=3,mz-3
>                                 do j=3,my-3
2796c2689
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                                 do j=3,my-3
2804c2697
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                                 do j=3,my-3
2812c2705
<                                 do j=(nghost/2+1),my-(nghost/2+1)
---
>                                 do j=3,my-3
```

4 domain.F90

As stated previously, we have tested only cases with periodic boundary conditions. The modifications to `enlarge_domain_with_disk()`, `enlarge_ydomain_with_disk()` should not be used without further testing.

4.1 enlarge_domain_with_disk()

```
214c214
<               buffsize = max(10*int(ppc0*upsamp_e*c*max((mx-nghost)*(my-nghost),1*(mx-nghost)*(mz-nghostz))) &
---
>               buffsize = max(10*int(ppc0*upsamp_e*c*max((mx-5)*(my-5),1*(mx-5)*(mz-5))) &
219c219
```

```
<               buffsize = max(3*int(ppc0*upsamp_e*c*max((mx-nghost),(my-nghost))),60000)
---
>               buffsize = max(3*int(ppc0*upsamp_e*c*max((mx-5),(my-5))),60000)
234,242c234,242
<               write(7)(((bx(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((by(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((bz(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((ex(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((ey(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((ez(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((curx(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((cury(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<               (((curz(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz)
---
>               write(7)(((bx(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((by(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((bz(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((ex(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((ey(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((ez(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((curx(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((cury(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>               (((curz(i,j,k),i=1,mxold-3),j=1,my),k=1,mz)
263,265c263
<       call create_MPI_filter_datatypes(nghost/2+1,mx-(nghost/2+1),&
<                                        nghost/2+1,my-(nghost/2+1),&
<                                        nghostz/2+1,mz-(nghostz/2+1))
---
>       call create_MPI_filter_datatypes(3,mx-3,3,my-3,3,mz-3)
271,272c269,270
<                       ,mz),bufferin1(mx,my,nghostz/2+1),bufferin2(mx,my,nghostz/2),&
<                       bufferin1y(mx,nghost/2+1,mz),bufferin2y(mx,nghost/2,mz))
---
>                       ,mz),bufferin1(mx,my,2),bufferin2(mx,my,2), &
>                       bufferin1y(mx,2,mz),bufferin2y(mx,2,mz))
275c273
<               allocate(bufferin1x(nghost/2+1,my,mz),bufferin2x(nghost/2,my,mz))
---
>               allocate(bufferin1x(2,my,mz),bufferin2x(2,my,mz))
326,334c324,332
<               read(7)(((bx(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((by(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((bz(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((ex(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((ey(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((ez(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((curx(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((cury(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz), &
<                       (((curz(i,j,k),i=1,mxold-(nghost/2+1)),j=1,my),k=1,mz)
---
>               read(7)(((bx(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((by(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((bz(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((ex(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((ey(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((ez(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((curx(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((cury(i,j,k),i=1,mxold-3),j=1,my),k=1,mz), &
>                       (((curz(i,j,k),i=1,mxold-3),j=1,my),k=1,mz)
357c355
<               mx0=sum(mxl(i1:i2)-nghost)+nghost
---
>               mx0=sum(mxl(i1:i2)-5)+5
361c359
<               mxcum=sum(mxl(i1:i2)-nghost)-(mxl(rank+1)-nghost)
---
>               mxcum=sum(mxl(i1:i2)-5)-(mxl(rank+1)-5)
```

4.2 enlarge_ydomain_with_disk()

```
405c403
<               buffsize = max(10*int(ppc0*c*max((mx-nghost)*(my-nghost),1*(mx-nghost)*(mz-nghostz))) &
---
>               buffsize = max(10*int(ppc0*c*max((mx-5)*(my-5),1*(mx-5)*(mz-5))) &
410c408
<               buffsize = max(3*int(ppc0*c*max((mx-nghost),(my-nghost))),60000)
---
>               buffsize = max(3*int(ppc0*c*max((mx-5),(my-5))),60000)
425,433c423,431
<               write(7)(((bx(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((by(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((bz(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((ex(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((ey(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((ez(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
```

```
<               (((curx(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((cury(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<               (((curz(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz)
---
>               write(7)(((bx(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((by(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((bz(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((ex(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((ey(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((ez(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((curx(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((cury(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>               (((curz(i,j,k),i=1,mx),j=1,myold-3),k=1,mz)
457,459c455
<       call create_MPI_filter_datatypes(nghost/2+1,mx-(nghost/2+1),&
<                                        (nghost/2+1),my-(nghost/2+1),&
<                                        (nghostz/2+1),mz-(nghostz/2+1))
---
>       call create_MPI_filter_datatypes(3,mx-3,3,my-3,3,mz-3)
464,465c460,461
<                       ,mz),bufferin1(mx,my,nghostz/2+1),bufferin2(mx,my,nghostz/2), &
<                       bufferin1y(mx,nghost/2+1,mz),bufferin2y(mx,nghost/2,mz))
---
>                       ,mz),bufferin1(mx,my,2),bufferin2(mx,my,2), &
>                       bufferin1y(mx,2,mz),bufferin2y(mx,2,mz))
469c465
<               allocate(bufferin1x(nghost/2+1,my,mz),bufferin2x(nghost/2,my,mz))
---
>               allocate(bufferin1x(2,my,mz),bufferin2x(2,my,mz))
509,517c505,513
<               read(7)(((bx(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((by(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((bz(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((ex(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((ey(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((ez(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((curx(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((cury(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz), &
<                       (((curz(i,j,k),i=1,mx),j=1,myold-(nghost/2+1)),k=1,mz)
---
>               read(7)(((bx(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((by(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((bz(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((ex(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((ey(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((ez(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((curx(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((cury(i,j,k),i=1,mx),j=1,myold-3),k=1,mz), &
>                       (((curz(i,j,k),i=1,mx),j=1,myold-3),k=1,mz)
539c535
<               my0=sum(myl(j1:j2:sizex)-nghost)+nghost
---
>               my0=sum(myl(j1:j2:sizex)-5)+5
543c539
<               mycum=sum(myl(j1:j2:sizex)-nghost)-(myl(j2)-nghost)
---
>               mycum=sum(myl(j1:j2:sizex)-5)-(myl(j2)-5)
```

5 particles.F90

```
69,71d68
<
<         real(sprec) :: three,two,thhalf, nineighth, one, threeq, twoth, half, &
<              third, quart, sixth, negsixth, negone
137,138c134
<         public :: reorder_particles, exchange_particles, zigzag,&
<                         densdecomp_1ord, densdecomp_2ord, densdecomp_3ord, &
---
>         public :: reorder_particles, exchange_particles, zigzag, &
169,171c165
<                       movwinoffset, pi, external_fields, user_part_bcs, &
<                         three,two,thhalf, nineighth, one, threeq, twoth, half, &
<                         third, quart, sixth, negsixth, negone
---
>                       movwinoffset, pi, external_fields, user_part_bcs
```

5.1 read_input_particles()

These are a few constants that are precomputed for use in shape function definitions.

```
239,252d232
<         three=3.
```

```
<         two=2.
<         thhalf=3/2.
<         nineighth=9/8.
<         one=1.
<         threeq=3/4.
<         twoth=2/3.
<         half=1/2.
<         third=1/3.
<         quart=1/4.
<         sixth=1/6.
<         negsixth=-1/6.
<         negone=-1.
<
```

5.2 allocate_particles()

```
309c289
<       buffsize = max(10*int(ppc0*max(upsamp_e,upsamp_i)*c*max((mx-nghost)*(my-nghost),1*(mx-nghost)*(mz-nghostz))),10000)
---
>       buffsize = max(10*int(ppc0*max(upsamp_e,upsamp_i)*c*max((mx-5)*(my-5),1*(mx-5)*(mz-5))),10000)
312c292
<       buffsize = max(3*int(ppc0*max(upsamp_e,upsamp_i)*c*max((mx-nghost),(my-nghost))),1060000)
---
>       buffsize = max(3*int(ppc0*max(upsamp_e,upsamp_i)*c*max((mx-5),(my-5))),1060000)
339,344c319,324
<       x1in=1.*(nghost/2+1)
<       x2in=mx0-1.*(nghost/2)
<       y1in=1.*(nghost/2+1)
<       y2in=my0-1.*(nghost/2)
<       z1in=1.*(nghostz/2+1)
<       z2in=mz0-1.*(nghostz/2)
---
>       x1in=3.
>       x2in=mx0-2.
>       y1in=3.
>       y2in=my0-2.
>       z1in=3.
>       z2in=mz0-2.
```

The following subroutines (`densdecomp_1ord`, `densdecomp_2ord`, `densdecomp_3ord`) are implementations of the Esirkepov current deposit algorithm for 1st, 2nd, and 3rd order shape functions [1]. A different subroutine (`densdecomp_1ord`, `densdecomp_2ord`, `densdecomp_3ord`) is used for each choice of shape function; apart from shape function definition, the routines are the same. These are written for speed, not readability. A more readable (also slow) version of the current deposit is copied below (just the key parts, for 3D). The 2D Eqs. are similar, however the weights `Wx`, `Wy`, `Wz`, as well as the z-current update vis-à-vis `tmpz` admit a reduced form, given in Sec. 5 of [1].

```
! 3d not optimized
do iter=1,6
   DSx(iter)=Sx2(iter)-Sx1(iter)
   DSy(iter)=Sy2(iter)-Sy1(iter)
   DSz(iter)=Sz2(iter)-Sz1(iter)
enddo

do iter2=1,6
   do iter1=1,6
      do iter=1,6
         Wx(iter,iter1,iter2)=DSx(iter )*(Sy1(iter1)*Sz1(iter2)+half*DSy(iter1)*Sz1(iter2) &
              +half*Sy1(iter1)*DSz(iter2)+third*DSy(iter1)*DSz(iter2))
         Wy(iter,iter1,iter2)=DSy(iter1)*(Sx1(iter )*Sz1(iter2)+half*DSx(iter )*Sz1(iter2) &
              +half*Sx1(iter )*DSz(iter2)+third*DSx(iter )*DSz(iter2))
         Wz(iter,iter1,iter2)=DSz(iter2)*(Sx1(iter )*Sy1(iter1)+half*DSx(iter )*Sy1(iter1) &
              +half*Sx1(iter )*DSy(iter1)+third*DSx(iter )*DSy(iter1))
      enddo
   enddo
enddo
   do iter=1,6
      do iter1=1,6
         do iter2=1,6
            if(iter.eq.1) then
               tmpx(iter,iter1,iter2)=q*Wx(iter, iter1, iter2)
            else
               tmpx(iter,iter1,iter2)=tmpx(iter-1,iter1,iter2) &
                    + q*Wx(iter,iter1,iter2)
            endif
         enddo
      enddo
   enddo
   do iter=1,6
      do iter1=1,6
         do iter2=1,6
            if(iter1.eq.1) then
               tmpy(iter,iter1,iter2)=q*Wy(iter,iter1,iter2)
            else
               tmpy(iter,iter1,iter2)=tmpy(iter,iter1-1,iter2) &
                    + q*Wy(iter,iter1,iter2)
            endif
         enddo
      enddo
   enddo
   do iter=1,6
      do iter1=1,6
         do iter2=1,6
            if(iter2.eq.1) then
               tmpz(iter,iter1,iter2)=q*Wz(iter,iter1,iter2)
            else
               tmpz(iter,iter1,iter2)=tmpz(iter,iter1,iter2-1) &
                    + q*Wz(iter,iter1,iter2)
            endif
         enddo
      enddo
   enddo
do iter=1,6
   do iter1=1,6
      do iter2=1,6
         curx(i1-3+iter,j1-3+iter1,k1-3+iter2)=curx(i1-3+iter,j1-3+iter1,k1-3+iter2)+tmpx(iter,iter1,iter2)
         cury(i1-3+iter,j1-3+iter1,k1-3+iter2)=cury(i1-3+iter,j1-3+iter1,k1-3+iter2)+tmpy(iter,iter1,iter2)
         curz(i1-3+iter,j1-3+iter1,k1-3+iter2)=curz(i1-3+iter,j1-3+iter1,k1-3+iter2)+tmpz(iter,iter1,iter2)
      enddo
   enddo
enddo
```

5.3 densdecomp_1ord()

```
672,1362d650
< !                                          subroutine densdecomp()
< !
< ! Charge (current) deposition on the grid, Esirkepov algorithm
< !
< !-------------------------------------------------------------------------------
<
< subroutine densdecomp_1ord(x2,y2,z2,x1,y1,z1,in)
<
<   implicit none
<
<   ! dummy variables
<
<   logical :: in
<   real(sprec), intent(in) :: x1,x2
<   real(sprec), intent(in) :: y1,y2,z1,z2
<   real(sprec) :: dx1,dy1,dx2,dy2,dz1,dz2,deltaz
<
<   ! local variables
<   real(sprec),DIMENSION(6)::Sx1,Sy1,Sx2,Sy2,Sz1,Sz2
<   real(sprec)::curx_add,curx_add_prev,curz_add
<   real(sprec),DIMENSION(6)::cury_adds,cury_add_prevs
<   real(sprec),DIMENSION(6,6)::curz_adds,curz_add_prevs
<   integer::i1,i2,j1,j2,k1,k2,iter,iter1,iter2, &
<        iterx1min,iterx1max,itery1min,itery1max,iterz1min,iterz1max,&
<        iterx2min,iterx2max,itery2min,itery2max,iterz2min,iterz2max,&
<        iterxmin,iterxmax,iterymin,iterymax,iterzmin,iterzmax,&
<        i,j,shifti,shiftj,shiftk,l2
<
<   i1=aint(x1)
<   i2=aint(x2)
<   j1=aint(y1)
<   j2=aint(y2)
<   k1=aint(z1)
<   k2=aint(z2)
<   shifti=aint(x2)-aint(x1)
<   shiftj=aint(y2)-aint(y1)
<   shiftk=aint(z2)-aint(z1)
<   dx1=x1-aint(x1)
<   dy1=y1-aint(y1)
<   dx2=x2-aint(x2)
<   dy2=y2-aint(y2)
<   dz1=z1-aint(z1)
<   dz2=z2-aint(z2)
<   deltaz=z2-z1
<
<   Sx1(:)=0.
<   Sx2(:)=0.
<   Sy1(:)=0.
<   Sy2(:)=0.
<
< #ifndef twoD
<   Sz1(:)=0.
<   Sz2(:)=0.
<   curz_adds=0.
<   curz_add_prevs=0.
< #else
<   k1=1
<   k2=1
<   shiftk=0
<   curz_add=0.
< #endif
<
<
< !---------------------------------------
< ! 0 order form factor
< !---------------------------------------
<   Sx1(3)=1.-dx1
<   Sx1(4)=dx1
<   iterx1min = 3
<   iterx1max = 4
<   Sx2(3+shifti)=1.-dx2
<   Sx2(4+shifti)=dx2
<   iterx2min=3+shifti
<   iterx2max=4+shifti
<
<   Sy1(3)=1.-dy1
<   Sy1(4)=dy1
<   itery1min = 3
<   itery1max = 4
<   Sy2(3+shiftj)=1.-dy2
<   Sy2(4+shiftj)=dy2
<   itery2min=3+shiftj
```

```
<   itery2max=4+shiftj
<
<   iterxmin=min(iterx1min,iterx2min)
<   iterxmax=max(iterx1max,iterx2max)
<   iterymin=min(itery1min,itery2min)
<   iterymax=max(itery1max,itery2max)
<
< #ifndef twoD
<   Sz1(3)=1.-dz1
<   Sz1(4)=dz1
<   iterz1min = 3
<   iterz1max = 4
<   Sz2(3+shiftk)=1.-dz2
<   Sz2(4+shiftk)=dz2
<   iterz2min=3+shiftk
<   iterz2max=4+shiftk
<
<   iterzmin=min(iterz1min,iterz2min)
<   iterzmax=max(iterz1max,iterz2max)
< #endif
<
< #ifndef twoD
<   do iter2=iterzmin,iterzmax
<      do iter1=iterymin,iterymax
<         do iter=iterxmin,iterxmax
<
<            l2=(i1-3+iter)+(j1-3+iter1-1)*iy+(k1-3+iter2-1)*iz
<
<            !if(l2.lt.1 .or. l2.gt.lot) then
<            !   print *,"lot=",lot,"l2=",l2
<            !   print *,"i1, j1, k1",i1,j1,k1
<            !   print *,"ix,iy,iz",ix,iy,iz
<            !   print *,"iter,iter1,iter2",iter,iter1,iter2
<            !   print *,"index err"
<            !endif
<
<            curx_add=q*((Sx2(iter )-Sx1(iter ))&
<                 *(Sy1(iter1)*Sz1(iter2)+half*(Sy2(iter1)-Sy1(iter1))*Sz1(iter2)&
<                 +half*Sy1(iter1)*(Sz2(iter2)-Sz1(iter2))&
<                 +third*(Sy2(iter1)-Sy1(iter1))*(Sz2(iter2)-Sz1(iter2))&
<            ))+curx_add_prev
<            cury_adds(iter)=q*((Sy2(iter1)-Sy1(iter1))&
<                 *(Sx1(iter )*Sz1(iter2)+half*(Sx2(iter )-Sx1(iter ))*Sz1(iter2)&
<                 +half*Sx1(iter )*(Sz2(iter2)-Sz1(iter2))&
<                 +third*(Sx2(iter )-Sx1(iter ))*(Sz2(iter2)-Sz1(iter2))&
<            ))+cury_add_prevs(iter)
<            curz_adds(iter,iter1)=q*((Sz2(iter2)-Sz1(iter2))&
<                 *(Sx1(iter )*Sy1(iter1)+half*(Sx2(iter )-Sx1(iter ))*Sy1(iter1)&
<                 +half*Sx1(iter )*(Sy2(iter1)-Sy1(iter1))&
<                 +third*(Sx2(iter )-Sx1(iter ))*(Sy2(iter1)-Sy1(iter1))&
<            ))+curz_add_prevs(iter,iter1)
<
<            curx(l2,1,1)=curx(l2,1,1)+curx_add
<            cury(l2,1,1)=cury(l2,1,1)+cury_adds(iter)
<            curz(l2,1,1)=curz(l2,1,1)+curz_adds(iter,iter1)
<
<            curx_add_prev = curx_add
<
<         enddo
<         curx_add=0.
<         cury_add_prevs=cury_adds
<         cury_adds=0.
<      enddo
<      curz_add_prevs=curz_adds
<      curz_adds=0.
<   enddo
< #else
<   ! 2d optimized
<      do iter1=min(itery1min,itery2min),max(itery1max,itery2max)
<         do iter=min(iterx1min,iterx2min),max(iterx1max,iterx2max)
<
<            l2=(i1-3+iter)+(j1-3+iter1-1)*iy
<
<            !if(l2.lt.1 .or. l2.gt.lot) then
<            !   print *,"index err"
<            !   print *,"lot=",lot,"l2=",l2
<            !   print *,"i1, j1",i1,j1
<            !   print *,"ix,iy",ix,iy
<            !   print *,"iter,iter1",iter,iter1
<            !endif
<
<            curx_add=q*((Sx2(iter)-Sx1(iter))*(Sy1(iter1)+half*(Sy2(iter1)-Sy1(iter1))))+curx_add_prev
<            cury_adds(iter)=q*((Sy2(iter1)-Sy1(iter1))*(Sx1(iter )+half*(Sx2(iter)-Sx1(iter))))+cury_add_prevs(iter)
<            curz_add=-1*q*deltaz*(Sx1(iter )*Sy1(iter1)+half*(Sx2(iter)-Sx1(iter))*Sy1(iter1) &
<                 +half*Sx1(iter)*(Sy2(iter1)-Sy1(iter1))+third*(Sx2(iter)-Sx1(iter))*(Sy2(iter1)-Sy1(iter1)))
```

```
<
<            curx(l2,1,k1)=curx(l2,1,k1)+curx_add
<            cury(l2,1,k1)=cury(l2,1,k1)+cury_adds(iter)
<            curz(l2,1,k1)=curz(l2,1,k1)+curz_add
<
<            curx_add_prev=curx_add
<         enddo
<         curx_add=0.
<         cury_add_prevs=cury_adds
<         cury_adds=0.
<      enddo
< #endif
<
<      in=.true.
<
< end subroutine densdecomp_1ord
```

5.4 densdecomp_2ord()

```
< !-----------------------------------------------------------------------------
< !                                          subroutine densdecomp()
< !
< ! Charge (current) deposition on the grid, Esirkepov algorithm
< !
< !-----------------------------------------------------------------------------
<
< subroutine densdecomp_2ord(x2,y2,z2,x1,y1,z1,in)
<
<   implicit none
<
<   ! dummy variables
<   logical :: in
<   real(sprec), intent(in) :: x1,x2
<   real(sprec), intent(in) :: y1,y2,z1,z2
<   real(sprec) :: dx1,dy1,dx2,dy2,dz1,dz2,deltaz
<
<   ! local variables
<   real(sprec),DIMENSION(6)::Sx1,Sy1,Sx2,Sy2,Sz1,Sz2
<   real(sprec)::curx_add,curx_add_prev,curz_add
<   real(sprec),DIMENSION(6)::cury_adds,cury_add_prevs
<   real(sprec),DIMENSION(6,6)::curz_adds,curz_add_prevs
<   integer::i1,i2,j1,j2,k1,k2,iter,iter1,iter2, &
<        iterx1min,iterx1max,itery1min,itery1max,iterz1min,iterz1max,&
<        iterx2min,iterx2max,itery2min,itery2max,iterz2min,iterz2max,&
<        iterxmin,iterxmax,iterymin,iterymax,iterzmin,iterzmax,&
<        i,j,shifti,shiftj,shiftk,l2
<
<   i1=aint(x1)
<   i2=aint(x2)
<   j1=aint(y1)
<   j2=aint(y2)
<   k1=aint(z1)
<   k2=aint(z2)
<   shifti=aint(x2)-aint(x1)
<   shiftj=aint(y2)-aint(y1)
<   shiftk=aint(z2)-aint(z1)
<   dx1=x1-aint(x1)
<   dy1=y1-aint(y1)
<   dx2=x2-aint(x2)
<   dy2=y2-aint(y2)
<   dz1=z1-aint(z1)
<   dz2=z2-aint(z2)
<   deltaz=z2-z1
<
<   Sx1(:)=0.
<   Sx2(:)=0.
<   Sy1(:)=0.
<   Sy2(:)=0.
<
<   curx_add=0.
<   cury_adds=0.
<   curx_add_prev=0.
<   cury_add_prevs=0.
<
< #ifndef twoD
<   Sz1(:)=0.
<   Sz2(:)=0.
<   curz_adds=0.
<   curz_add_prevs=0.
< #else
<   k1=1
<   k2=1
<   shiftk=0
<   curz_add=0.
< #endif
```

```
<
<
< !---------------------
< ! 2 order form factor
< !---------------------
<   if (dx1.le.half) then
<      Sx1(2)=half*(dx1*dx1-dx1+quart)
<      Sx1(4)=Sx1(2)+dx1
<      Sx1(3)=one-Sx1(4)-Sx1(2)
<      iterx1min=2
<      iterx1max=4
<   elseif (dx1.gt.half) then
<      Sx1(3)=nineighth - thhalf*dx1 + half*dx1*dx1
<      Sx1(5)=Sx1(3)-one+dx1
<      Sx1(4)=one-Sx1(5)-Sx1(3)
<      iterx1min=3
<      iterx1max=5
<   endif
<   if (dy1.le.half) then
<      Sy1(2)=half*(dy1*dy1-dy1+quart)
<      Sy1(4)=Sy1(2)+dy1
<      Sy1(3)=one-Sy1(4)-Sy1(2)
<      itery1min=2
<      itery1max=4
<   elseif (dy1.gt.half) then
<      Sy1(3)=nineighth - thhalf*dy1 + half*dy1*dy1
<      Sy1(5)=Sy1(3)-one+dy1
<      Sy1(4)=one-Sy1(5)-Sy1(3)
<      itery1min=3
<      itery1max=5
<   endif
< #ifndef twoD
<   if (dz1.le.half) then
<      Sz1(2)=half*(dz1*dz1-dz1+quart)
<      Sz1(4)=Sz1(2)+dz1
<      Sz1(3)=one-Sz1(4)-Sz1(2)
<      iterz1min=2
<      iterz1max=4
<   elseif (dz1.gt.half) then
<      Sz1(3)=nineighth - thhalf*dz1 + half*dz1*dz1
<      Sz1(5)=Sz1(3)-one+dz1
<      Sz1(4)=one-Sz1(5)-Sz1(3)
<      iterz1min=3
<      iterz1max=5
<   endif
< #endif
<
<   if (dx2.le.half) then
<      Sx2(2+shifti)=half*(dx2*dx2-dx2+quart)
<      Sx2(4+shifti)=Sx2(2+shifti)+dx2
<      Sx2(3+shifti)=one-Sx2(4+shifti)-Sx2(2+shifti)
<      iterx2min=2+shifti
<      iterx2max=4+shifti
<   elseif (dx2.gt.half) then
<      Sx2(3+shifti)=nineighth-thhalf*dx2+half*dx2*dx2
<      Sx2(5+shifti)=Sx2(3+shifti)-one+dx2
<      Sx2(4+shifti)=one-Sx2(5+shifti)-Sx2(3+shifti)
<      iterx2min=3+shifti
<      iterx2max=5+shifti
<   endif
<   if(dy2.le.half) then
<      Sy2(2+shiftj)=half*(dy2*dy2-dy2+quart)
<      Sy2(4+shiftj)=Sy2(2+shiftj)+dy2
<      Sy2(3+shiftj)=one-Sy2(4+shiftj)-Sy2(2+shiftj)
<      itery2min=2+shiftj
<      itery2max=4+shiftj
<   elseif(dy2.gt.half) then
<      Sy2(3+shiftj)=nineighth-thhalf*dy2+half*dy2*dy2
<      Sy2(5+shiftj)=Sy2(3+shiftj)-one+dy2
<      Sy2(4+shiftj)=one-Sy2(5+shiftj)-Sy2(3+shiftj)
<      itery2min=3+shiftj
<      itery2max=5+shiftj
<   endif
<
<   iterxmin=min(iterx1min,iterx2min)
<   iterxmax=max(iterx1max,iterx2max)
<   iterymin=min(itery1min,itery2min)
<   iterymax=max(itery1max,itery2max)
<
< #ifndef twoD
<   if(dz2.le.half) then
<      Sz2(2+shiftk)=half*(dz2*dz2-dz2+quart)
<      Sz2(4+shiftk)=Sz2(2+shiftk)+dz2
<      Sz2(3+shiftk)=one-Sz2(4+shiftk)-Sz2(2+shiftk)
<      iterz2min=2+shiftk
```

```
<      iterz2max=4+shiftk
<   elseif(dz2.gt.half) then
<      Sz2(3+shiftk)=nineighth-thhalf*dz2+half*dz2*dz2
<      Sz2(5+shiftk)=Sz2(3+shiftk)-one+dz2
<      Sz2(4+shiftk)=one-Sz2(5+shiftk)-Sz2(3+shiftk)
<      iterz2min=3+shiftk
<      iterz2max=5+shiftk
<   endif
<
<   iterzmin=min(iterz1min,iterz2min)
<   iterzmax=max(iterz1max,iterz2max)
<
< #endif
<
<
< #ifndef twoD
<   do iter2=iterzmin,iterzmax
<      do iter1=iterymin,iterymax
<         do iter=iterxmin,iterxmax
<
<            l2=(i1-3+iter)+(j1-3+iter1-1)*iy+(k1-3+iter2-1)*iz
<
<            !if(l2.lt.1 .or. l2.gt.lot) then
<            !   print *,"lot=",lot,"l2=",l2
<            !   print *,"i1, j1, k1",i1,j1,k1
<            !   print *,"ix,iy,iz",ix,iy,iz
<            !   print *,"iter,iter1,iter2",iter,iter1,iter2
<            !   print *,"index err"
<            !endif
<
<            curx_add=q*((Sx2(iter )-Sx1(iter ))&
<                 *(Sy1(iter1)*Sz1(iter2)+half*(Sy2(iter1)-Sy1(iter1))*Sz1(iter2)&
<                 +half*Sy1(iter1)*(Sz2(iter2)-Sz1(iter2))&
<                 +third*(Sy2(iter1)-Sy1(iter1))*(Sz2(iter2)-Sz1(iter2))&
<            ))+curx_add_prev
<            cury_adds(iter)=q*((Sy2(iter1)-Sy1(iter1))&
<                 *(Sx1(iter )*Sz1(iter2)+half*(Sx2(iter )-Sx1(iter ))*Sz1(iter2)&
<                 +half*Sx1(iter )*(Sz2(iter2)-Sz1(iter2))&
<                 +third*(Sx2(iter )-Sx1(iter ))*(Sz2(iter2)-Sz1(iter2))&
<            ))+cury_add_prevs(iter)
<            curz_adds(iter,iter1)=q*((Sz2(iter2)-Sz1(iter2))&
<                 *(Sx1(iter )*Sy1(iter1)+half*(Sx2(iter )-Sx1(iter ))*Sy1(iter1)&
<                 +half*Sx1(iter )*(Sy2(iter1)-Sy1(iter1))&
<                 +third*(Sx2(iter )-Sx1(iter ))*(Sy2(iter1)-Sy1(iter1))&
<            ))+curz_add_prevs(iter,iter1)
<
<            curx(l2,1,1)=curx(l2,1,1)+curx_add
<            cury(l2,1,1)=cury(l2,1,1)+cury_adds(iter)
<            curz(l2,1,1)=curz(l2,1,1)+curz_adds(iter,iter1)
<
<            curx_add_prev = curx_add
<
<         enddo
<         curx_add=0.
<         cury_add_prevs=cury_adds
<         cury_adds=0.
<      enddo
<      curz_add_prevs=curz_adds
<      curz_adds=0.
<   enddo
< #else
<   ! 2d optimized
<      do iter1=min(itery1min,itery2min),max(itery1max,itery2max)
<         do iter=min(iterx1min,iterx2min),max(iterx1max,iterx2max)
<
<            l2=(i1-3+iter)+(j1-3+iter1-1)*iy
<
<            !if(l2.lt.1 .or. l2.gt.lot) then
<            !   print *,"index err"
<            !   print *,"lot=",lot,"l2=",l2
<            !   print *,"i1, j1",i1,j1
<            !   print *,"ix,iy",ix,iy
<            !   print *,"iter,iter1",iter,iter1
<            !endif
<
<            curx_add=q*((Sx2(iter)-Sx1(iter))*(Sy1(iter1)+half*(Sy2(iter1)-Sy1(iter1))))+curx_add_prev
<            cury_adds(iter)=q*((Sy2(iter1)-Sy1(iter1))*(Sx1(iter )+half*(Sx2(iter)-Sx1(iter))))+cury_add_prevs(iter)
<            curz_add=-1*q*deltaz*(Sx1(iter )*Sy1(iter1)+half*(Sx2(iter)-Sx1(iter))*Sy1(iter1) &
<                 +half*Sx1(iter)*(Sy2(iter1)-Sy1(iter1))+third*(Sx2(iter)-Sx1(iter))*(Sy2(iter1)-Sy1(iter1)))
<
<            curx(l2,1,k1)=curx(l2,1,k1)+curx_add
<            cury(l2,1,k1)=cury(l2,1,k1)+cury_adds(iter)
<            curz(l2,1,k1)=curz(l2,1,k1)+curz_add
<
<            curx_add_prev=curx_add
```

```
<         enddo
<         curx_add=0.
<         cury_add_prevs=cury_adds
<         cury_adds=0.
<       enddo
< #endif
<
<       in=.true.
<
< end subroutine densdecomp_2ord
```

5.5 densdecomp_3ord()

```
< !-----------------------------------------------------------------------------
< !                                          subroutine densdecomp()
< !
< ! Charge (current) deposition on the grid, Esirkepov algorithm
< !
< !-----------------------------------------------------------------------------
<
< subroutine densdecomp_3ord(x2,y2,z2,x1,y1,z1,in)
<
<   implicit none
<
<   ! dummy variables
<   logical :: in
<   real(sprec), intent(in) :: x1,x2
<   real(sprec), intent(in) :: y1,y2,z1,z2
<   real(sprec) :: dx1,dy1,dx2,dy2,dz1,dz2,deltaz
<
<   ! local variables
<   real(sprec),DIMENSION(6)::Sx1,Sy1,Sx2,Sy2,Sz1,Sz2
<   real(sprec)::curx_add,curx_add_prev,curz_add
<   real(sprec),DIMENSION(6)::cury_adds,cury_add_prevs
<   real(sprec),DIMENSION(6,6)::curz_adds,curz_add_prevs
<   integer::i1,i2,j1,j2,k1,k2,iter,iter1,iter2, &
<        iterx1min,iterx1max,itery1min,itery1max,iterz1min,iterz1max,&
<        iterx2min,iterx2max,itery2min,itery2max,iterz2min,iterz2max,&
<        iterxmin,iterxmax,iterymin,iterymax,iterzmin,iterzmax,&
<        i,j,shifti,shiftj,shiftk,l2
<
<   i1=aint(x1)
<   i2=aint(x2)
<   j1=aint(y1)
<   j2=aint(y2)
<   k1=aint(z1)
<   k2=aint(z2)
<   shifti=aint(x2)-aint(x1)
<   shiftj=aint(y2)-aint(y1)
<   shiftk=aint(z2)-aint(z1)
<   dx1=x1-aint(x1)
<   dy1=y1-aint(y1)
<   dx2=x2-aint(x2)
<   dy2=y2-aint(y2)
<   dz1=z1-aint(z1)
<   dz2=z2-aint(z2)
<   deltaz=z2-z1
<
<   Sx1(:)=0.
<   Sx2(:)=0.
<   Sy1(:)=0.
<   Sy2(:)=0.
<
<   curx_add=0.
<   cury_adds=0.
<   curx_add_prev=0.
<   cury_add_prevs=0.
<
< #ifndef twoD
<   Sz1(:)=0.
<   Sz2(:)=0.
<   curz_adds=0.
<   curz_add_prevs=0.
< #else
<   k1=1
<   k2=1
<   shiftk=0
<   curz_add=0.
< #endif
<
< !----------------------
< ! 3 order form factor
< !----------------------
<     if (dx1.le.half) then
<         Sx1(2)= negsixth*(dx1-one)*(dx1-one)*(dx1-one) !6
```

```
<         Sx1(3)= twoth+half*(dx1-two)*dx1*dx1 !5
<         Sx1(5)= sixth *dx1*dx1*dx1 !3
<         Sx1(4)= one-Sx1(5)-Sx1(3)-Sx1(2)!sixth*(one+three*dx1*(one+dx1-dx1*dx1)) !7
<         iterx1min = 2
<         iterx1max = 5
<     elseif (dx1.gt.half) then
<         Sx1(5)=sixth*dx1*dx1*dx1 !3
<         Sx1(4)=twoth+half*(negone-dx1)*(one-dx1)*(one-dx1) !7
<         Sx1(2)=sixth*(one-dx1)*(one-dx1)*(one-dx1) !6
<         Sx1(3)=one-Sx1(5)-Sx1(4)-Sx1(2)!Sx1(3)=sixth*(one+three*(one-dx1)*(two-(one-dx1)*(one-dx1)-dx1)) !10
<         iterx1min = 2
<         iterx1max = 5
<      endif
<      if (dy1.le.half) then
<         Sy1(2)= negsixth*(dy1-one)*(dy1-one)*(dy1-one) !6
<         Sy1(3)= twoth+half*(dy1-two)*dy1*dy1 !5
<         Sy1(5)= sixth *dy1*dy1*dy1 !3
<         Sy1(4)= one-Sy1(5)-Sy1(3)-Sy1(2)!sixth*(one+three*dy1*(one+dy1-dy1*dy1)) !7
<         itery1min = 2
<         itery1max = 5
<     elseif (dy1.gt.half) then
<         Sy1(5)=sixth*dy1*dy1*dy1 !3
<         Sy1(4)=twoth+half*(negone-dy1)*(one-dy1)*(one-dy1) !7
<         Sy1(2)=sixth*(one-dy1)*(one-dy1)*(one-dy1) !6
<         Sy1(3)=one-Sy1(5)-Sy1(4)-Sy1(2)!Sy1(3)=sixth*(one+three*(one-dy1)*(two-(one-dy1)*(one-dy1)-dy1)) !10
<         itery1min = 2
<         itery1max = 5
<      endif
< #ifndef twoD
<      if (dz1.le.half) then
<         Sz1(2)= negsixth*(dz1-one)*(dz1-one)*(dz1-one) !6
<         Sz1(3)= twoth+half*(dz1-two)*dz1*dz1 !5
<         Sz1(5)= sixth *dz1*dz1*dz1 !3
<         Sz1(4)= one-Sz1(5)-Sz1(3)-Sz1(2)!sixth*(one+three*dz1*(one+dz1-dz1*dz1)) !7
<         iterz1min = 2
<         iterz1max = 5
<     elseif (dz1.gt.half) then
<         Sz1(5)=sixth*dz1*dz1*dz1 !3
<         Sz1(4)=twoth+half*(negone-dz1)*(one-dz1)*(one-dz1) !7
<         Sz1(2)=sixth*(one-dz1)*(one-dz1)*(one-dz1) !6
<         Sz1(3)=one-Sz1(5)-Sz1(4)-Sz1(2)!Sz1(3)=sixth*(one+three*(one-dz1)*(two-(one-dz1)*(one-dz1)-dz1)) !10
<         iterz1min = 2
<         iterz1max = 5
<      endif
< #endif
<      if (dx2.le.half) then
<         Sx2(2+shifti)= negsixth*(dx2-one)*(dx2-one)*(dx2-one) !6
<         Sx2(3+shifti)= twoth+half*(dx2-two)*dx2*dx2 !5
<         Sx2(5+shifti)= sixth *dx2*dx2*dx2 !3
<         Sx2(4+shifti)= one-Sx2(5+shifti)-Sx2(3+shifti)-Sx2(2+shifti)!sixth*(one+three*dx2*(one+dx2-dx2*dx2)) !7
<         iterx2min = 2+shifti
<         iterx2max = 5+shifti
<     elseif (dx2.gt.half) then
<         Sx2(5+shifti)=sixth*dx2*dx2*dx2 !3
<         Sx2(4+shifti)=twoth+half*(negone-dx2)*(one-dx2)*(one-dx2) !7
<         Sx2(2+shifti)=sixth*(one-dx2)*(one-dx2)*(one-dx2) !6
<         Sx2(3+shifti)=one-Sx2(5+shifti)-Sx2(4+shifti)-Sx2(2+shifti)
<         iterx2min = 2+shifti
<         iterx2max = 5+shifti
<      endif
<      if (dy2.le.half) then
<         Sy2(2+shiftj)= negsixth*(dy2-one)*(dy2-one)*(dy2-one) !6
<         Sy2(3+shiftj)= twoth+half*(dy2-two)*dy2*dy2 !5
<         Sy2(5+shiftj)= sixth *dy2*dy2*dy2 !3
<         Sy2(4+shiftj)= one-Sy2(5+shiftj)-Sy2(3+shiftj)-Sy2(2+shiftj)
<         itery2min = 2+shiftj
<         itery2max = 5+shiftj
<     elseif (dy2.gt.half) then
<         Sy2(5+shiftj)=sixth*dy2*dy2*dy2 !3
<         Sy2(4+shiftj)=twoth+half*(negone-dy2)*(one-dy2)*(one-dy2) !7
<         Sy2(2+shiftj)=sixth*(one-dy2)*(one-dy2)*(one-dy2) !6
<         Sy2(3+shiftj)=one-Sy2(5+shiftj)-Sy2(4+shiftj)-Sy2(2+shiftj)
<         itery2min = 2+shiftj
<         itery2max = 5+shiftj
<      endif
<
<      iterxmin=min(iterx1min,iterx2min)
<      iterxmax=max(iterx1max,iterx2max)
<      iterymin=min(itery1min,itery2min)
<      iterymax=max(itery1max,itery2max)
<
< #ifndef twoD
<      if (dz2.le.half) then
<         Sz2(2+shiftk)= negsixth*(dz2-one)*(dz2-one)*(dz2-one) !6
<         Sz2(3+shiftk)= twoth+half*(dz2-two)*dz2*dz2 !5
```

```
<         Sz2(5+shiftk)= sixth *dz2*dz2*dz2 !3
<         Sz2(4+shiftk)= one-Sz2(5+shiftk)-Sz2(3+shiftk)-Sz2(2+shiftk)
<         iterz2min = 2+shiftk
<         iterz2max = 5+shiftk
<     elseif (dz2.gt.half) then
<         Sz2(5+shiftk)=sixth*dz2*dz2*dz2 !3
<         Sz2(4+shiftk)=twoth+half*(negone-dz2)*(one-dz2)*(one-dz2) !7
<         Sz2(2+shiftk)=sixth*(one-dz2)*(one-dz2)*(one-dz2) !6
<         Sz2(3+shiftk)=one-Sz2(5+shiftk)-Sz2(4+shiftk)-Sz2(2+shiftk)
<         iterz2min = 2+shiftk
<         iterz2max = 5+shiftk
<      endif
<
<      iterzmin=min(iterz1min,iterz2min)
<      iterzmax=max(iterz1max,iterz2max)
< #endif
<
< #ifndef twoD
<   do iter2=iterzmin,iterzmax
<      do iter1=iterymin,iterymax
<         do iter=iterxmin,iterxmax
<            l2=(i1-3+iter)+(j1-3+iter1-1)*iy+(k1-3+iter2-1)*iz
<
<            curx_add=q*((Sx2(iter )-Sx1(iter ))&
<                 *(Sy1(iter1)*Sz1(iter2)+half*(Sy2(iter1)-Sy1(iter1))*Sz1(iter2)&
<                 +half*Sy1(iter1)*(Sz2(iter2)-Sz1(iter2))&
<                 +third*(Sy2(iter1)-Sy1(iter1))*(Sz2(iter2)-Sz1(iter2))&
<            ))+curx_add_prev
<            cury_adds(iter)=q*((Sy2(iter1)-Sy1(iter1))&
<                 *(Sx1(iter )*Sz1(iter2)+half*(Sx2(iter )-Sx1(iter ))*Sz1(iter2)&
<                 +half*Sx1(iter )*(Sz2(iter2)-Sz1(iter2))&
<                 +third*(Sx2(iter )-Sx1(iter ))*(Sz2(iter2)-Sz1(iter2))&
<            ))+cury_add_prevs(iter)
<            curz_adds(iter,iter1)=q*((Sz2(iter2)-Sz1(iter2))&
<                 *(Sx1(iter )*Sy1(iter1)+half*(Sx2(iter )-Sx1(iter ))*Sy1(iter1)&
<                 +half*Sx1(iter )*(Sy2(iter1)-Sy1(iter1))&
<                 +third*(Sx2(iter )-Sx1(iter ))*(Sy2(iter1)-Sy1(iter1))&
<            ))+curz_add_prevs(iter,iter1)
<
<            curx(l2,1,1)=curx(l2,1,1)+curx_add
<            cury(l2,1,1)=cury(l2,1,1)+cury_adds(iter)
<            curz(l2,1,1)=curz(l2,1,1)+curz_adds(iter,iter1)
<
<            curx_add_prev = curx_add
<
<         enddo
<         curx_add=0.
<         cury_add_prevs=cury_adds
<         cury_adds=0.
<      enddo
<      curz_add_prevs=curz_adds
<      curz_adds=0.
<   enddo
<
< #else
<   ! 2d optimized
<      do iter1=min(itery1min,itery2min),max(itery1max,itery2max)
<         do iter=min(iterx1min,iterx2min),max(iterx1max,iterx2max)
<
<            l2=(i1-3+iter)+(j1-3+iter1-1)*iy
<
<            curx_add=q*((Sx2(iter)-Sx1(iter))*(Sy1(iter1)+half*(Sy2(iter1)-Sy1(iter1))))+curx_add_prev
<            cury_adds(iter)=q*((Sy2(iter1)-Sy1(iter1))*(Sx1(iter )+half*(Sx2(iter)-Sx1(iter))))+cury_add_prevs(iter)
<            curz_add=-1*q*deltaz*(Sx1(iter )*Sy1(iter1)+half*(Sx2(iter)-Sx1(iter))*Sy1(iter1) &
<                 +half*Sx1(iter)*(Sy2(iter1)-Sy1(iter1))+third*(Sx2(iter)-Sx1(iter))*(Sy2(iter1)-Sy1(iter1)))
<
<            curx(l2,1,k1)=curx(l2,1,k1)+curx_add
<            cury(l2,1,k1)=cury(l2,1,k1)+cury_adds(iter)
<            curz(l2,1,k1)=curz(l2,1,k1)+curz_add
<
<            curx_add_prev=curx_add
<         enddo
<         curx_add=0.
<         cury_add_prevs=cury_adds
<         cury_adds=0.
<      enddo
< #endif
<
<      in=.true.
<
< end subroutine densdecomp_3ord
```

5.6 inject_others()

```
1370c726
```

```
<               perz=sign(.5*(mz-1.*nghostz),p(ions)%z-1.*(nghostz/2+1))+sign(.5*(mz-1.*nghostz),p(ions)%z-mz +1.*(nghostz/2))
---
>               perz=sign(.5*(mz-5.),p(ions)%z-3.)+sign(.5*(mz-5.),p(ions)%z-mz +2.)
1380c736
<                       perz=-(mzl(k1+1)-1.*nghostz)
---
>                       perz=-(mzl(k1+1)-5.)
1426c782
<       perz=sign(.5*(mz-1.*nghostz),p(ions)%z-1.*(nghostz/2+1))+sign(.5*(mz-1.*nghostz),p(ions)%z-mz +1.*(nghostz/2))
---
>       perz=sign(.5*(mz-5.),p(ions)%z-3.)+sign(.5*(mz-5.),p(ions)%z-mz +2.)
1435c791
<                       perz=-(mzl(k1+1)-1.*nghostz)
---
>                       perz=-(mzl(k1+1)-5.)
1482c838
<       perz=sign(.5*(mz-1.*nghostz),p(maxhlf+lecs)%z-1.*(nghostz/2+1))&
<                        +sign(.5*(mz-1.*nghostz),p(maxhlf+lecs)%z- mz +1.*(nghostz/2))
---
>       perz=sign(.5*(mz-5.),p(maxhlf+lecs)%z-3.)+sign(.5*(mz-5.),p(maxhlf+lecs)%z- mz +2.)
1489c845
<                       perz=-(mzl(k1+1)-1.*nghostz)
---
>                       perz=-(mzl(k1+1)-5.)
1538c894
<       perz=sign(.5*(mz-1.*nghostz),p(maxhlf+lecs)%z-1.*(nghostz/2+1))&
<                       +sign(.5*(mz-1.*nghostz),p(maxhlf+lecs)%z- mz +1.*(nghostz/2))
---
>       perz=sign(.5*(mz-5.),p(maxhlf+lecs)%z-3.)+sign(.5*(mz-5.),p(maxhlf+lecs)%z- mz +2.)
1545c901
<                       perz=-(mzl(k1+1)-1.*nghostz)
---
>                       perz=-(mzl(k1+1)-5.)
1596c952
<               pery=sign(.5*(my-1.*nghost),p(ions)%y-1.*(nghost/2+1))+sign(.5*(my-1.*nghost),p(ions)%y-my+1.*(nghost/2))
---
>               pery=sign(.5*(my-5.),p(ions)%y-3.)+sign(.5*(my-5.),p(ions)%y-my+2.)
1601c957
<                       pery=-(myl(j1+1)-1.*nghost)
---
>                       pery=-(myl(j1+1)-5.)
1641c997
<               pery=sign(.5*(my-1.*nghost),p(ions)%y-1.*(nghost/2+1))+sign(.5*(my-1.*nghost),p(ions)%y-my+1.*(nghost/2))
---
>               pery=sign(.5*(my-5.),p(ions)%y-3.)+sign(.5*(my-5.),p(ions)%y-my+2.)
1646c1002
<                       pery=-(myl(j1+1)-1.*nghost)
---
>                       pery=-(myl(j1+1)-5.)
1687c1043
<               pery=sign(.5*(my-1.*nghost),p(maxhlf+lecs)%y-1.*(nghost/2+1))&
<                      +sign(.5*(my-1.*nghost),p(maxhlf+lecs)%y-my+1.*(nghost/2))
---
>               pery=sign(.5*(my-5.),p(maxhlf+lecs)%y-3.)+sign(.5*(my-5.),p(maxhlf+lecs)%y-my+2.)
1692c1048
<                       pery=-(myl(j1+1)-1.*nghost)
---
>                       pery=-(myl(j1+1)-5.)
1733c1089
<               pery=sign(.5*(my-1.*nghost),p(maxhlf+lecs)%y-1.*(nghost/2+1))&
<                      +sign(.5*(my-1.*nghost),p(maxhlf+lecs)%y-my+1.*(nghost/2))
---
>               pery=sign(.5*(my-5.),p(maxhlf+lecs)%y-3.)+sign(.5*(my-5.),p(maxhlf+lecs)%y-my+2.)
1738c1094
<                       pery=-(myl(j1+1)-1.*nghost)
---
>                       pery=-(myl(j1+1)-5.)
```

5.7 init_maxw_table()

This next change is unrelated to current deposit, but is a small improvement to the PDF used for initialization of the particles. The arrays `pdf_table_e` and `pdf_table_i` are set to be of dimension `pdf_sz+1`, then elements from 2 to `pdf_sz+1` are computed as the prefix sum of `func`. Without this change, `pdf_table_e` and `pdf_table_i` are not large enough to contain all elements in the cumulative sum of `func`, so one element is skipped over (i.e., `sum(func(1:1))`). This leads to a sharp feature between bins 1 and 2 of the PDFs, but is removed via the modification below.

```
2086,2087c1444,1445
<       do i=1,pdf_sz
<               pdf_table(i+1)=sum(func(1:i))
---
>       do i=2,pdf_sz
>               pdf_table(i)=sum(func(1:i))
```

5.8 inject_plasma_region()

```
2495,2496c1853
<       real, dimension(pdf_sz) :: gamma_table_i, gamma_table_e
<         real, dimension(pdf_sz+1) :: pdf_table_i, pdf_table_e
---
>       real, dimension(pdf_sz) :: pdf_table_i, gamma_table_i, pdf_table_e, gamma_table_e
2549,2550c1906,1907
<       iglob_min=(nghost/2+1)+mxcum
<       iglob_max=(mx-(nghost/2))+mxcum
---
>       iglob_min=3+mxcum
>       iglob_max=(mx-2)+mxcum
2552,2553c1909,1910
<       minx=1.*(nghost/2+1)
<       maxx=1.*(nghost/2+1)
---
>       minx=3.
>       maxx=3.
2563,2566c1920,1923
<       if(inject_minx<iglob_min) minx=1.*(nghost/2+1)
<       if(inject_maxx<iglob_min) maxx=1.*(nghost/2+1)
<       if(inject_minx>=iglob_max) minx=mx-(nghost/2)
<       if(inject_maxx>=iglob_max) maxx=mx-(nghost/2)
---
>       if(inject_minx<iglob_min) minx=3.
>       if(inject_maxx<iglob_min) maxx=3.
>       if(inject_minx>=iglob_max) minx=mx-2
>       if(inject_maxx>=iglob_max) maxx=mx-2
2577,2578c1934,1935
<       jglob_min=(nghost/2+1)+mycum !global extent of the y boundaries on this processor
<       jglob_max=(my-(nghost/2))+mycum
---
>       jglob_min=3+mycum !global extent of the y boundaries on this processor
>       jglob_max=(my-2)+mycum
2580,2581c1937,1938
<       miny=1.*(nghost/2+1)
<       maxy=1.*(nghost/2+1)
---
>       miny=3.
>       maxy=3.
2595,2598c1952,1955
<       if(inject_miny<jglob_min) miny=1.*(nghost/2+1)
<       if(inject_maxy<jglob_min) maxy=1.*(nghost/2+1)
<       if(inject_miny>=jglob_max) miny=my-nghost/2
<       if(inject_maxy>=jglob_max) maxy=my-nghost/2
---
>       if(inject_miny<jglob_min) miny=3.
>       if(inject_maxy<jglob_min) maxy=3.
>       if(inject_miny>=jglob_max) miny=my-2
>       if(inject_maxy>=jglob_max) maxy=my-2
2619,2620c1976,1977
<       maxz=1.*(nghostz/2+1)+1.
<       minz=1.*(nghostz/2+1)
---
>       maxz=4.
>       minz=3.
2626,2627c1983,1984
<       kglob_min=(nghostz/2+1)+mzcum !global extent of the y boundaries on this processor
<       kglob_max=(mz-(nghostz/2))+mzcum
---
>       kglob_min=3+mzcum !global extent of the y boundaries on this processor
>       kglob_max=(mz-2)+mzcum
2629,2630c1986,1987
<       minz=1.*(nghostz/2+1)
<       maxz=1.*(nghostz/2+1)
---
>       minz=3.
>       maxz=3.
2640,2643c1997,2000
<       if(inject_minz<kglob_min) minz=1.*(nghostz/2+1)
<       if(inject_maxz<kglob_min) maxz=1.*(nghostz/2+1)
<       if(inject_minz>=kglob_max) minz=mz-1.*(nghostz/2)
<       if(inject_maxz>=kglob_max) maxz=mz-1.*(nghostz/2)
---
>       if(inject_minz<kglob_min) minz=3.
>       if(inject_maxz<kglob_min) maxz=3.
>       if(inject_minz>=kglob_max) minz=mz-2.
>       if(inject_maxz>=kglob_max) maxz=mz-2.
2737,2739c2089,2091
<                       values(1)=(p(ions)%x-1.*(nghost/2+1))/c_omp
<                       values(2)=((p(ions)%y+modulo(rank,sizey)*(myall-nghost)) -1.*(nghost/2+1))/c_omp
<                       values(3)=((p(ions)%z+(rank/sizey)*(mzall-nghostz))-1.*(nghostz/2+1))/c_omp
---
>                       values(1)=(p(ions)%x-3.)/c_omp
```

```
>                       values(2)=((p(ions)%y+modulo(rank,sizey)*(myall-5)) -3.)/c_omp
>                       values(3)=((p(ions)%z+(rank/sizey)*(mzall-5))-3.)/c_omp
2760,2762c2113,2115
<                       values(1)=(p(ions)%x-1.*(nghost/2+1))/c_omp
<                       values(2)=((p(ions)%y+modulo(rank,sizey)*(myall-nghost)) -1.*(nghost/2+1))/c_omp
<                       values(3)=((p(ions)%z+(rank/sizey)*(mzall-nghostz))-1.*(nghostz/2+1))/c_omp
---
>                       values(1)=(p(ions)%x-3.)/c_omp
>                       values(2)=((p(ions)%y+modulo(rank,sizey)*(myall-5)) -3.)/c_omp
>                       values(3)=((p(ions)%z+(rank/sizey)*(mzall-5))-3.)/c_omp
```

6 particles_movedeposit.F90

6.1 particles_movedeposit()

A mover routine, consistent with the current deposit, is called (`mover` for zigzag, `mover_1ord` for 1st order density decomposition, `mover_2ord` for 2nd order, `mover_3ord` for 3rd). The routines `mover_1ord`, `mover_2ord`, and `mover_3ord` differ only by shape function definition. For the density decomposition mover routines, fields are computed as the sum of the particle shape function multiplied by the field at (primal) gridpoints. In 3D, the fields at gridpoints are computed via linear interpolation of the fields defined on the Yee lattice. These interpolated arrays are denoted by `ex_p, ey_p, ez_p, bx_p, by_p, bz_p`. The 2D case is a similar operation, but uses shifted particle shape functions (to align with gridpoints on the Yee lattice) and fields defined on the Yee lattice. This difference between 2D and 3D implementations (i.e. for the 2D implementation, the use of particle shape functions defined at Yee lattice points in conjunction with the EM fields defined on the Yee lattice, and in 3D the use of shape functions defined at primal gridpoints in conjunction with EM fields defined at primal gridpoints) is not chosen for any fundamental reason, it is just the state they were left in after optimization of the 2D and 3D mover routines. The 2D mover routine could be changed to match the 3D version.

```
69,85c69,71
< #ifdef zzag
<                       call mover(1,ions,qmi)                ! ions
<                       call mover(1+maxhlf,lecs+maxhlf,qme)   ! electrons
< #endif
< #ifdef dd1
<                       call mover_1ord(1,ions,qmi)                   ! ions
<                       call mover_1ord(1+maxhlf,lecs+maxhlf,qme)   ! electrons
< #endif
< #ifdef dd2
<                       call mover_2ord(1,ions,qmi)                   ! ions
<                       call mover_2ord(1+maxhlf,lecs+maxhlf,qme)   ! electrons
< #endif
< #ifdef dd3
<                       call mover_3ord(1,ions,qmi)                   ! ions
<                       call mover_3ord(1+maxhlf,lecs+maxhlf,qme)   ! electrons
< #endif
<
---
>               call mover(1,ions,qmi)                  ! ions
>               call mover(1+maxhlf,lecs+maxhlf,qme)    ! electrons
>
```

6.2 mover_1ord()

```
350,1274d340
< !                                           subroutine mover_1ord()
< !
< !
< !
< !-------------------------------------------------------------------------------
<
< subroutine mover_1ord(n1,n2,qm)
<
<       implicit none
<
<       ! dummy variables
<
<       integer, intent(in) :: n2, n1
<       real(sprec), intent(in) :: qm
<
<       ! local variables
<
<         integer :: iter1,iter2,iter3,iter1min,iter1max,iter2min,iter2max,iter3min,iter3max
<
<
<         real, DIMENSION(6) :: Sxp,Syp,Sxd,Syd,Szp,Szd
<       integer :: npr,l,n,ip,jp,kp,id,jd,kd
<         integer :: lpp,lpd, ldp, ldd
<         integer :: lppp
<       real dxp,dyp,dzp,dxd,dyd,dzd,f,g,ex0,ey0,ez0,bx0,by0,bz0
<
```

```
<       real u0,v0,w0,u1,v1,w1, cinv, g1, corrqm
<       real qm0
<         logical :: cond1, cond2, cond3, cond4
< !, xglob, yglob, gammawall, betawall, gamma, walloc
< !     real vdriftx, vdrifty, gdrift, decrem
<
< !     integer ::yes_gammacut
< !     real gammacut
< #ifdef vay
<       real ustar,sig,tx,ty,tz,vx0,vy0,vz0
< #endif
<         real bx_ext, by_ext, bz_ext, ex_ext, ey_ext, ez_ext
<         real chival,threed
<         real exsc,eysc,ezsc,bxsc,bysc,bzsc,v3x0,v3y0,v3z0
<         real up,vp,wp,gampinv,ppinv,gamp,dragx,dragy,dragz,lfx,lfy,lfz,edotv,lsq
<         real bx00,by00,bz00,dragcoeff,dragthresh,reductionfactor
<         integer redLL
<         real gammathinv
<
<         real,dimension(mx,my,mz)::ex_p, ey_p, ez_p, bx_p, by_p, bz_p
<
<         ! primal grids
<         ex_p=0.5*(ex+cshift(ex,-1,dim=1))
<         ey_p=0.5*(ey+cshift(ey,-1,dim=2))
<         ez_p=0.5*(ez+cshift(ez,-1,dim=3))
<         bx_p=0.5*(bx+cshift(bx,-1,dim=2))
<         by_p=0.5*(by+cshift(by,-1,dim=1))
<         bz_p=0.5*(0.5*(bz+cshift(bz,-1,dim=1))&
<              +0.5*(cshift(bz,-1,dim=2)+cshift(cshift(bz,-1,dim=2),-1,dim=1)))
<
<
< !user stripe cooling
<  redLL=1 ! if 1 then use reduced Landau Lifshits, otherwise Hededal (apart from 1/beta^2)
<  dragthresh=1e-1 ! maximum differential change in momentum
<  gammathinv=1./(1.+3.*delgam)
< !end user stripe cooling
<
< !     real pcosth, pphi, psinth, v0t, ut1, vt1, wt1, gam
< !     real ptx, pty, ptz
<
<       cinv=1./c
<       qm0=qm
<
<       do n=n1,n2
<               npr=n
<
<               ip=aint(p(npr)%x)
<               dxp=p(npr)%x-ip
<               jp=aint(p(npr)%y)
<               dyp=p(npr)%y-jp
<               kp=aint(p(npr)%z)
<                 Sxp(:)=0.
<                 Syp(:)=0.
<
<                 id=aint(p(npr)%x-half)
<               dxd=p(npr)%x-half-id
<               jd=aint(p(npr)%y-half)
<               dyd=p(npr)%y-half-jd
<               kd=aint(p(npr)%z-half)
<                 Sxd(:)=0.
<                 Syd(:)=0.
<
< #ifndef twoD
<                 dzp=p(npr)%z-kp
<                 Szp(:)=0.
<                 dzd=(p(npr)%z-half)-kd
<                 Szd(:)=0.
< #endif
<
<
<   !0 order form factor
<   Sxp(3)=1.-dxp
<   Sxp(4)=dxp
<   Sxd(3)=1.-dxd
<   Sxd(4)=dxd
<   iter1min=3
<   iter1max=4
<
<   Syp(3)=1.-dyp
<   Syp(4)=dyp
<   Syd(3)=1.-dyd
<   Syd(4)=dyd
<   iter2min=3
<   iter2max=4
<
```

```
< #ifndef twoD
<   Szp(3)=1.-dzp
<   Szp(4)=dzp
<   Szd(3)=1.-dzd
<   Szd(4)=dzd
<   iter3min=3
<   iter3max=4
< #endif
<
<   ex0=0.
<   ey0=0.
<   ez0=0.
<   bx0=0.
<   by0=0.
<   bz0=0.
<
<
< #ifndef twoD
<   do iter3=iter3min,iter3max
<      do iter2=iter2min,iter2max
<            !lppp=(ip-3+iter1)+iy*(jp-3+iter2-1)+iz*(kp-3+iter3-1)
<            !if (lppp.ge.1 .and. lppp.le.lot) then
<            !    print *,"out of range"
<            !endif
<               ex0=ex0+sum(ex_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               ey0=ey0+sum(ey_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               ez0=ez0+sum(ez_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               bx0=bx0+sum(bx_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               by0=by0+sum(by_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               bz0=bz0+sum(bz_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<      enddo
<   enddo
< #else
<   do iter2=iter2min,iter2max
<      do iter1=iter1min,iter1max
<         lpp=(ip-3+iter1)+iy*(jp-3+iter2-1)
<         lpd=(ip-3+iter1)+iy*(jd-3+iter2-1)
<         ldp=(id-3+iter1)+iy*(jp-3+iter2-1)
<         ldd=(id-3+iter1)+iy*(jd-3+iter2-1)
<
<         ex0=ex0+ex(ldp,1,1)*Sxd(iter1)*Syp(iter2)
<         ey0=ey0+ey(lpd,1,1)*Sxp(iter1)*Syd(iter2)
<         ez0=ez0+ez(lpp,1,1)*Sxp(iter1)*Syp(iter2)
<         bx0=bx0+bx(lpd,1,1)*Sxp(iter1)*Syd(iter2)
<         by0=by0+by(ldp,1,1)*Sxd(iter1)*Syp(iter2)
<         bz0=bz0+bz(ldd,1,1)*Sxd(iter1)*Syd(iter2)
<      enddo
<   enddo
< #endif
<
<   ex0=0.5*ex0*qm
<   ey0=0.5*ey0*qm
<   ez0=0.5*ez0*qm
<   bx0=0.5*bx0*qm*cinv
<   by0=0.5*by0*qm*cinv
<   bz0=0.5*bz0*qm*cinv
<
<   if(external_fields) then
<
<      call get_external_fields(real(p(npr)%x,sprec),p(npr)%y,&
<           p(npr)%z,ex_ext,ey_ext,ez_ext,bx_ext,by_ext,bz_ext)
<
<      bx0=bx0+bx_ext*0.5*qm*cinv
<      by0=by0+by_ext*0.5*qm*cinv
<      bz0=bz0+bz_ext*0.5*qm*cinv
<      ex0=ex0+ex_ext*0.5*qm
<      ey0=ey0+ey_ext*0.5*qm
<      ez0=ez0+ez_ext*0.5*qm
<
<   endif
<
<   !to be used later for synchrotron
<   bx00=bx0
<   by00=by0
<   bz00=bz0
<
<   !   First half electric acceleration, with relativity's gamma:
< #ifdef vay
<   !Use Vay 2008 particle mover
<   g=1./sqrt(1.+p(npr)%u**2+p(npr)%v**2+p(npr)%w**2) !reciprocal of the Lorentz factor
<   vx0=c*p(npr)%u*g !3-velocity of the particle
<   vy0=c*p(npr)%v*g
<   vz0=c*p(npr)%w*g
<
<   u1=c*p(npr)%u+2.*ex0+vy0*bz0-vz0*by0 !uprime, taking into account
```

```
<   !that cinv is already incorporated within B
<   v1=c*p(npr)%v+2.*ey0+vz0*bx0-vx0*bz0
<   w1=c*p(npr)%w+2.*ez0+vx0*by0-vy0*bx0
<
<   !Lorentz factor for uprime
<
<   ustar=cinv*(u1*bx0+v1*by0+w1*bz0)
<   sig=cinv*cinv*(c**2+u1**2+v1**2+w1**2)-(bx0**2+by0**2+bz0**2)
<   g=1./sqrt(0.5*(sig+sqrt(sig**2+4.*(bx0**2+by0**2+bz0**2+ustar**2))))
<   tx=bx0*g
<   ty=by0*g
<   tz=bz0*g
<   f=1./(1.+tx**2+ty**2+tz**2)
<
<   u0=f*(u1+(u1*tx+v1*ty+w1*tz)*tx+v1*tz-w1*ty)
<   v0=f*(v1+(u1*tx+v1*ty+w1*tz)*ty+w1*tx-u1*tz)
<   w0=f*(w1+(u1*tx+v1*ty+w1*tz)*tz+u1*ty-v1*tx)
< #else
<   !Use Boris algorithm
<   !   First half electric acceleration, with Lorentz gamma:
<   u0=c*p(npr)%u+ex0
<   v0=c*p(npr)%v+ey0
<   w0=c*p(npr)%w+ez0
<   !   First half magnetic rotation, with Lorentz gamma:
<   g=c/sqrt(c**2+u0**2+v0**2+w0**2)
<   bx0=g*bx0
<   by0=g*by0
<   bz0=g*bz0
<
<   f=2./(1.+bx0*bx0+by0*by0+bz0*bz0)
<   u1=(u0+v0*bz0-w0*by0)*f
<   v1=(v0+w0*bx0-u0*bz0)*f
<   w1=(w0+u0*by0-v0*bx0)*f
<   !   Second half mag. rot'n &   el. acc'n:
<   u0=u0+v1*bz0-w1*by0+ex0
<   v0=v0+w1*bx0-u1*bz0+ey0
<   w0=w0+u1*by0-v1*bx0+ez0
< #endif
<
<   !user stripe
<   !following tamburrini, cooling is implemented at this point,
<   !after the Lorentz push
<
<   !   Get normalized 4-velocity:
<   p(npr)%u=u0*cinv
<   p(npr)%v=v0*cinv
<   p(npr)%w=w0*cinv
<
<   ! stripe user
<   ! freeze particles, to verify the generation of the correct fields
<   ! (the three lines above should also be commented out)
<   !                    p(npr)%u=p(npr)%u
<   !                     p(npr)%v=p(npr)%v
<   !                     p(npr)%w=p(npr)%w
<   ! end stripe user
<
<   !   Position advance:
<   g=c/sqrt(c**2+u0**2+v0**2+w0**2)
<   p(npr)%x=p(npr)%x + p(npr)%u*g*c
<   p(npr)%y=p(npr)%y + p(npr)%v*g*c
<   p(npr)%z=p(npr)%z + p(npr)%w*g*c
<
< enddo
<
< end subroutine mover_1ord
```

6.3 mover_2ord()

```
< !-------------------------------------------------------------------------------
< !                                          subroutine mover_2ord()
< !
< !
< !
< !-------------------------------------------------------------------------------
<
< subroutine mover_2ord(n1,n2,qm)
<
<       implicit none
<
<       ! dummy variables
<
<       integer, intent(in) :: n2, n1
<       real(sprec), intent(in) :: qm
<
<       ! local variables
```

```
<
<         integer :: iter1,iter2,iter3,iter1min,iter1max,iter2min,iter2max,iter3min,iter3max
<
<
<         real, DIMENSION(6) :: Sxp,Syp,Sxd,Syd,Szp,Szd
<       integer :: npr,l,n,ip,jp,kp,id,jd,kd
<         integer :: lpp,lpd, ldp, ldd
<         integer :: lppp
<       real dxp,dyp,dzp,dxd,dyd,dzd,f,g,ex0,ey0,ez0,bx0,by0,bz0
<
<       real u0,v0,w0,u1,v1,w1, cinv, g1, corrqm
<       real qm0
<         logical :: cond1, cond2, cond3, cond4
< !, xglob, yglob, gammawall, betawall, gamma, walloc
< !     real vdriftx, vdrifty, gdrift, decrem
<
< !     integer ::yes_gammacut
< !     real gammacut
< #ifdef vay
<       real ustar,sig,tx,ty,tz,vx0,vy0,vz0
< #endif
<         real bx_ext, by_ext, bz_ext, ex_ext, ey_ext, ez_ext
<         real chival,threed
<         real exsc,eysc,ezsc,bxsc,bysc,bzsc,v3x0,v3y0,v3z0
<         real up,vp,wp,gampinv,ppinv,gamp,dragx,dragy,dragz,lfx,lfy,lfz,edotv,lsq
<         real bx00,by00,bz00,dragcoeff,dragthresh,reductionfactor
<         integer redLL
<         real gammathinv
<
<         real,dimension(mx,my,mz)::ex_p, ey_p, ez_p, bx_p, by_p, bz_p
<
<         ! primal grids
<         ex_p=0.5*(ex+cshift(ex,-1,dim=1))
<         ey_p=0.5*(ey+cshift(ey,-1,dim=2))
<         ez_p=0.5*(ez+cshift(ez,-1,dim=3))
<         bx_p=0.5*(bx+cshift(bx,-1,dim=2))
<         by_p=0.5*(by+cshift(by,-1,dim=1))
<         bz_p=0.5*(0.5*(bz+cshift(bz,-1,dim=1))&
<              +0.5*(cshift(bz,-1,dim=2)+cshift(cshift(bz,-1,dim=2),-1,dim=1)))
<         !bz_p=0.5*(bz_p + cshift(bz_p,-1,dim=3))
<
<
< !user stripe cooling
<  redLL=1 ! if 1 then use reduced Landau Lifshits, otherwise Hededal (apart from 1/beta^2)
<  dragthresh=1e-1 ! maximum differential change in momentum
<  gammathinv=1./(1.+3.*delgam)
< !end user stripe cooling
<
< !     real pcosth, pphi, psinth, v0t, ut1, vt1, wt1, gam
< !     real ptx, pty, ptz
<
<       cinv=1./c
<       qm0=qm
<
<       do n=n1,n2
<               npr=n
<
<               ip=aint(p(npr)%x)
<               dxp=p(npr)%x-ip
<               jp=aint(p(npr)%y)
<               dyp=p(npr)%y-jp
<               kp=aint(p(npr)%z)
<                 Sxp(:)=0.
<                 Syp(:)=0.
<
<                 id=aint(p(npr)%x-half)
<               dxd=p(npr)%x-half-id
<               jd=aint(p(npr)%y-half)
<               dyd=p(npr)%y-half-jd
<               kd=aint(p(npr)%z-half)
<                 Sxd(:)=0.
<                 Syd(:)=0.
<
< #ifndef twoD
<                 dzp=p(npr)%z-kp
<                 Szp(:)=0.
<                 dzd=(p(npr)%z-half)-kd
<                 Szd(:)=0.
< #endif
<
<                 if (dxp.le.half) then
<                    Sxp(2) = half * (dxp*dxp-dxp+quart)
<                    Sxp(4) = Sxp(2)+dxp
<                    Sxp(3) = one-Sxp(4)-Sxp(2)
<                    iter1min=2
```

```
<                   iter1max=4
<                elseif(dxp.gt.half) then
<                   Sxp(3) = nineighth - thhalf*dxp + half*dxp*dxp
<                   Sxp(5) = Sxp(3)-one+dxp
<                   Sxp(4) = one-Sxp(5)-Sxp(3)
<                   iter1min=3
<                   iter1max=5
<                endif
<                if (dyp.le.half) then
<                   Syp(2) = half * (dyp*dyp-dyp+quart)
<                   Syp(4) = Syp(2)+dyp
<                   Syp(3) = one-Syp(4)-Syp(2)
<                   iter2min=2
<                   iter2max=4
<                elseif(dyp.gt.half) then
<                   Syp(3) = nineighth - thhalf*dyp + half*dyp*dyp
<                   Syp(5) = Syp(3)-one+dyp
<                   Syp(4) = one-Syp(5)-Syp(3)
<                   iter2min=3
<                   iter2max=5
<                endif
< #ifndef twoD
<                if (dzp.le.half) then
<                   Szp(2) = half * (dzp*dzp-dzp+quart)
<                   Szp(4) = Szp(2)+dzp
<                   Szp(3) = one-Szp(4)-Szp(2)
<                   iter3min=2
<                   iter3max=4
<                elseif(dzp.gt.half) then
<                   Szp(3) = nineighth - thhalf*dzp + half*dzp*dzp
<                   Szp(5) = Szp(3)-one+dzp
<                   Szp(4) = one-Szp(5)-Szp(3)
<                   iter3min=3
<                   iter3max=5
<                endif
< #endif
<                if (dxd.le.half) then
<                   Sxd(2) = half * (dxd*dxd-dxd+quart)
<                   Sxd(4) = Sxd(2)+dxd
<                   Sxd(3) = one-Sxd(4)-Sxd(2)
<                   iter1min=2
<                   iter1max=4
<                elseif(dxd.gt.half) then
<                   Sxd(3) = nineighth - thhalf*dxd + half*dxd*dxd
<                   Sxd(5) = Sxd(3)-one+dxd
<                   Sxd(4) = one-Sxd(5)-Sxd(3)
<                   iter1min=3
<                   iter1max=5
<                endif
<                if (dyd.le.half) then
<                   Syd(2) = half * (dyd*dyd-dyd+quart)
<                   Syd(4) = Syd(2)+dyd
<                   Syd(3) = one-Syd(4)-Syd(2)
<                   iter2min=2
<                   iter2max=4
<                elseif(dyd.gt.half) then
<                   Syd(3) = nineighth - thhalf*dyd + half*dyd*dyd
<                   Syd(5) = Syd(3)-one+dyd
<                   Syd(4) = one-Syd(5)-Syd(3)
<                   iter2min=3
<                   iter2max=5
<                endif
< #ifndef twoD
<                if (dzd.le.half) then
<                   Szd(2) = half * (dzd*dzd-dzd+quart)
<                   Szd(4) = Szd(2)+dzd
<                   Szd(3) = one-Szd(4)-Szd(2)
<                   iter3min=2
<                   iter3max=4
<                elseif(dzd.gt.half) then
<                   Szd(3) = nineighth - thhalf*dzd + half*dzd*dzd
<                   Szd(5) = Szd(3)-one+dzd
<                   Szd(4) = one-Szd(5)-Szd(3)
<                   iter3min=3
<                   iter3max=5
<                endif
< #endif
<
<   ex0=0.
<   ey0=0.
<   ez0=0.
<   bx0=0.
<   by0=0.
<   bz0=0.
<
```

```
<
< #ifndef twoD
<   do iter3=iter3min,iter3max
<      do iter2=iter2min,iter2max
<            !lppp=(ip-3+iter1)+iy*(jp-3+iter2-1)+iz*(kp-3+iter3-1)
<            !if (lppp.ge.1 .and. lppp.le.lot) then
<            !    print *,"out of range"
<            !endif
<               ex0=ex0+sum(ex_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               ey0=ey0+sum(ey_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               ez0=ez0+sum(ez_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               bx0=bx0+sum(bx_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               by0=by0+sum(by_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<               bz0=bz0+sum(bz_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<         !endif
<      enddo
<   enddo
< #else
<   do iter2=iter2min,iter2max
<      do iter1=iter1min,iter1max
<         lpp=(ip-3+iter1)+iy*(jp-3+iter2-1)
<         lpd=(ip-3+iter1)+iy*(jd-3+iter2-1)
<         ldp=(id-3+iter1)+iy*(jp-3+iter2-1)
<         ldd=(id-3+iter1)+iy*(jd-3+iter2-1)
<
<         ex0=ex0+ex(ldp,1,1)*Sxd(iter1)*Syp(iter2)
<         ey0=ey0+ey(lpd,1,1)*Sxp(iter1)*Syd(iter2)
<         ez0=ez0+ez(lpp,1,1)*Sxp(iter1)*Syp(iter2)
<         bx0=bx0+bx(lpd,1,1)*Sxp(iter1)*Syd(iter2)
<         by0=by0+by(ldp,1,1)*Sxd(iter1)*Syp(iter2)
<         bz0=bz0+bz(ldd,1,1)*Sxd(iter1)*Syd(iter2)
<      enddo
<   enddo
< #endif
<
<   ex0=0.5*ex0*qm
<   ey0=0.5*ey0*qm
<   ez0=0.5*ez0*qm
<   bx0=0.5*bx0*qm*cinv
<   by0=0.5*by0*qm*cinv
<   bz0=0.5*bz0*qm*cinv
<
<   if(external_fields) then
<
<      call get_external_fields(real(p(npr)%x,sprec),p(npr)%y,&
<           p(npr)%z,ex_ext,ey_ext,ez_ext,bx_ext,by_ext,bz_ext)
<
<      bx0=bx0+bx_ext*0.5*qm*cinv
<      by0=by0+by_ext*0.5*qm*cinv
<      bz0=bz0+bz_ext*0.5*qm*cinv
<      ex0=ex0+ex_ext*0.5*qm
<      ey0=ey0+ey_ext*0.5*qm
<      ez0=ez0+ez_ext*0.5*qm
<
<   endif
<
<   !to be used later for synchrotron
<   bx00=bx0
<   by00=by0
<   bz00=bz0
<
<   !   First half electric acceleration, with relativity's gamma:
< #ifdef vay
<   !Use Vay 2008 particle mover
<   g=1./sqrt(1.+p(npr)%u**2+p(npr)%v**2+p(npr)%w**2) !reciprocal of the Lorentz factor
<   vx0=c*p(npr)%u*g !3-velocity of the particle
<   vy0=c*p(npr)%v*g
<   vz0=c*p(npr)%w*g
<
<   u1=c*p(npr)%u+2.*ex0+vy0*bz0-vz0*by0 !uprime, taking into account
<   !that cinv is already incorporated within B
<   v1=c*p(npr)%v+2.*ey0+vz0*bx0-vx0*bz0
<   w1=c*p(npr)%w+2.*ez0+vx0*by0-vy0*bx0
<
<   !Lorentz factor for uprime
<
<   ustar=cinv*(u1*bx0+v1*by0+w1*bz0)
<   sig=cinv*cinv*(c**2+u1**2+v1**2+w1**2)-(bx0**2+by0**2+bz0**2)
<   g=1./sqrt(0.5*(sig+sqrt(sig**2+4.*(bx0**2+by0**2+bz0**2+ustar**2))))
<   tx=bx0*g
<   ty=by0*g
<   tz=bz0*g
<   f=1./(1.+tx**2+ty**2+tz**2)
<
<   u0=f*(u1+(u1*tx+v1*ty+w1*tz)*tx+v1*tz-w1*ty)
```

```
<   v0=f*(v1+(u1*tx+v1*ty+w1*tz)*ty+w1*tx-u1*tz)
<   w0=f*(w1+(u1*tx+v1*ty+w1*tz)*tz+u1*ty-v1*tx)
< #else
<   !Use Boris algorithm
<   !   First half electric acceleration, with Lorentz gamma:
<   u0=c*p(npr)%u+ex0
<   v0=c*p(npr)%v+ey0
<   w0=c*p(npr)%w+ez0
<   !   First half magnetic rotation, with Lorentz gamma:
<   g=c/sqrt(c**2+u0**2+v0**2+w0**2)
<   bx0=g*bx0
<   by0=g*by0
<   bz0=g*bz0
<
<   f=2./(1.+bx0*bx0+by0*by0+bz0*bz0)
<   u1=(u0+v0*bz0-w0*by0)*f
<   v1=(v0+w0*bx0-u0*bz0)*f
<   w1=(w0+u0*by0-v0*bx0)*f
<   !   Second half mag. rot'n &   el. acc'n:
<   u0=u0+v1*bz0-w1*by0+ex0
<   v0=v0+w1*bx0-u1*bz0+ey0
<   w0=w0+u1*by0-v1*bx0+ez0
< #endif
<
<   !user stripe
<   !following tamburrini, cooling is implemented at this point,
<   !after the Lorentz push
<
<   !   Get normalized 4-velocity:
<   p(npr)%u=u0*cinv
<   p(npr)%v=v0*cinv
<   p(npr)%w=w0*cinv
<
<   ! stripe user
<   ! freeze particles, to verify the generation of the correct fields
<   ! (the three lines above should also be commented out)
<   !                    p(npr)%u=p(npr)%u
<   !                     p(npr)%v=p(npr)%v
<   !                     p(npr)%w=p(npr)%w
<   ! end stripe user
<
<   !   Position advance:
<   g=c/sqrt(c**2+u0**2+v0**2+w0**2)
<   p(npr)%x=p(npr)%x + p(npr)%u*g*c
<   p(npr)%y=p(npr)%y + p(npr)%v*g*c
<   p(npr)%z=p(npr)%z + p(npr)%w*g*c
<
< enddo
<
< end subroutine mover_2ord
<
```

6.4 mover_3ord()

```
< !-------------------------------------------------------------------------------
< !                                          subroutine mover_3ord()
< !
< !
< !-------------------------------------------------------------------------------
<
< subroutine mover_3ord(n1,n2,qm)
<
<       implicit none
<
<       ! dummy variables
<
<       integer, intent(in) :: n2, n1
<       real(sprec), intent(in) :: qm
<
<       ! local variables
<
<         integer :: iter1,iter2,iter3,iter1min,iter1max,iter2min,iter2max,iter3min,iter3max
<
<
<         real, DIMENSION(6) :: Sxp,Syp,Sxd,Syd,Szp,Szd
<       integer :: npr,l,n,ip,jp,kp,id,jd,kd
<         integer :: lpp,lpd, ldp, ldd
<         integer :: lppp
<       real dxp,dyp,dzp,dxd,dyd,dzd,f,g,ex0,ey0,ez0,bx0,by0,bz0
<
<       real u0,v0,w0,u1,v1,w1, cinv, g1, corrqm
<       real qm0
<         logical :: cond1, cond2, cond3, cond4
< !, xglob, yglob, gammawall, betawall, gamma, walloc
```

```
< !     real vdriftx, vdrifty, gdrift, decrem
<
< !     integer ::yes_gammacut
< !     real gammacut
< #ifdef vay
<       real ustar,sig,tx,ty,tz,vx0,vy0,vz0
< #endif
<         real bx_ext, by_ext, bz_ext, ex_ext, ey_ext, ez_ext
<         real chival,threed
<         real exsc,eysc,ezsc,bxsc,bysc,bzsc,v3x0,v3y0,v3z0
<         real up,vp,wp,gampinv,ppinv,gamp,dragx,dragy,dragz,lfx,lfy,lfz,edotv,lsq
<         real bx00,by00,bz00,dragcoeff,dragthresh,reductionfactor
<         integer redLL
<         real gammathinv
<
<         real,dimension(mx,my,mz)::ex_p, ey_p, ez_p, bx_p, by_p, bz_p
<
<         ! primal grids
<         ex_p=0.5*(ex+cshift(ex,-1,dim=1))
<         ey_p=0.5*(ey+cshift(ey,-1,dim=2))
<         ez_p=0.5*(ez+cshift(ez,-1,dim=3))
<         bx_p=0.5*(bx+cshift(bx,-1,dim=2))
<         by_p=0.5*(by+cshift(by,-1,dim=1))
<         bz_p=0.5*(0.5*(bz+cshift(bz,-1,dim=1))&
<              +0.5*(cshift(bz,-1,dim=2)+cshift(cshift(bz,-1,dim=2),-1,dim=1)))
<
<
< !user stripe cooling
<  redLL=1 ! if 1 then use reduced Landau Lifshits, otherwise Hededal (apart from 1/beta^2)
<  dragthresh=1e-1 ! maximum differential change in momentum
<  gammathinv=1./(1.+3.*delgam)
< !end user stripe cooling
<
< !     real pcosth, pphi, psinth, v0t, ut1, vt1, wt1, gam
< !     real ptx, pty, ptz
<
<       cinv=1./c
<       qm0=qm
<
<       do n=n1,n2
<               npr=n
<
<               ip=aint(p(npr)%x)
<               dxp=p(npr)%x-ip
<               jp=aint(p(npr)%y)
<               dyp=p(npr)%y-jp
<               kp=aint(p(npr)%z)
<                 Sxp(:)=0.
<                 Syp(:)=0.
<
<                 id=aint(p(npr)%x-half)
<               dxd=p(npr)%x-half-id
<               jd=aint(p(npr)%y-half)
<               dyd=p(npr)%y-half-jd
<               kd=aint(p(npr)%z-half)
<                 Sxd(:)=0.
<                 Syd(:)=0.
<
< #ifndef twoD
<                 dzp=p(npr)%z-kp
<                 Szp(:)=0.
<                 dzd=(p(npr)%z-half)-kd
<                 Szd(:)=0.
< #endif
<
< !--------------------
< !3 order form factor
< !--------------------
<     if (dxp.le.half) then
<         Sxp(2)= negsixth*(dxp-one)*(dxp-one)*(dxp-one)
<         Sxp(3)= twoth+half*(dxp-two)*dxp*dxp
<         Sxp(5)= sixth *dxp*dxp*dxp
<         Sxp(4)= one-Sxp(5)-Sxp(3)-Sxp(2)
<         iter1min = 2
<         iter1max = 5
<     elseif (dxp.gt.half) then
<         Sxp(5)=sixth*dxp*dxp*dxp
<         Sxp(4)=twoth+half*(negone-dxp)*(one-dxp)*(one-dxp)
<         Sxp(2)=sixth*(one-dxp)*(one-dxp)*(one-dxp)
<         Sxp(3)=one-Sxp(5)-Sxp(4)-Sxp(2)
<         iter1min = 2
<         iter1max = 5
<      endif
<      if (dyp.le.half) then
<         Syp(2)= negsixth*(dyp-one)*(dyp-one)*(dyp-one)
```

```
<         Syp(3)= twoth+half*(dyp-two)*dyp*dyp
<         Syp(5)= sixth *dyp*dyp*dyp
<         Syp(4)= one-Syp(5)-Syp(3)-Syp(2)
<         iter2min = 2
<         iter2max = 5
<     elseif (dyp.gt.half) then
<         Syp(5)=sixth*dyp*dyp*dyp
<         Syp(4)=twoth+half*(negone-dyp)*(one-dyp)*(one-dyp)
<         Syp(2)=sixth*(one-dyp)*(one-dyp)*(one-dyp)
<         Syp(3)=one-Syp(5)-Syp(4)-Syp(2)
<         iter2min = 2
<         iter2max = 5
<      endif
< #ifndef twoD
<      if (dzp.le.half) then
<         Szp(2)= negsixth*(dzp-one)*(dzp-one)*(dzp-one)
<         Szp(3)= twoth+half*(dzp-two)*dzp*dzp
<         Szp(5)= sixth *dzp*dzp*dzp
<         Szp(4)= one-Szp(5)-Szp(3)-Szp(2)
<         iter3min = 2
<         iter3max = 5
<     elseif (dzp.gt.half) then
<         Szp(5)=sixth*dzp*dzp*dzp
<         Szp(4)=twoth+half*(negone-dzp)*(one-dzp)*(one-dzp)
<         Szp(2)=sixth*(one-dzp)*(one-dzp)*(one-dzp)
<         Szp(3)=one-Szp(5)-Szp(4)-Szp(2)
<         iter3min = 2
<         iter3max = 5
<      endif
< #endif
<      if (dxd.le.half) then
<         Sxd(2)= negsixth*(dxd-one)*(dxd-one)*(dxd-one)
<         Sxd(3)= twoth+half*(dxd-two)*dxd*dxd
<         Sxd(5)= sixth *dxd*dxd*dxd
<         Sxd(4)= one-Sxd(5)-Sxd(3)-Sxd(2)
<         iter1min = 2
<         iter1max = 5
<     elseif (dxd.gt.half) then
<         Sxd(5)=sixth*dxd*dxd*dxd
<         Sxd(4)=twoth+half*(negone-dxd)*(one-dxd)*(one-dxd)
<         Sxd(2)=sixth*(one-dxd)*(one-dxd)*(one-dxd)
<         Sxd(3)=one-Sxd(5)-Sxd(4)-Sxd(2)
<         iter1min = 2
<         iter1max = 5
<      endif
<      if (dyd.le.half) then
<         Syd(2)= negsixth*(dyd-one)*(dyd-one)*(dyd-one)
<         Syd(3)= twoth+half*(dyd-two)*dyd*dyd
<         Syd(5)= sixth *dyd*dyd*dyd
<         Syd(4)= one-Syd(5)-Syd(3)-Syd(2)
<         iter2min = 2
<         iter2max = 5
<     elseif (dyd.gt.half) then
<         Syd(5)=sixth*dyd*dyd*dyd
<         Syd(4)=twoth+half*(negone-dyd)*(one-dyd)*(one-dyd)
<         Syd(2)=sixth*(one-dyd)*(one-dyd)*(one-dyd)
<         Syd(3)=one-Syd(5)-Syd(4)-Syd(2)
<         iter2min = 2
<         iter2max = 5
<      endif
< #ifndef twoD
<      if (dzd.le.half) then
<         Szd(2)= negsixth*(dzd-one)*(dzd-one)*(dzd-one)
<         Szd(3)= twoth+half*(dzd-two)*dzd*dzd
<         Szd(5)= sixth *dzd*dzd*dzd
<         Szd(4)= one-Szd(5)-Szd(3)-Szd(2)
<         iter3min = 2
<         iter3max = 5
<     elseif (dzd.gt.half) then
<         Szd(5)=sixth*dzd*dzd*dzd
<         Szd(4)=twoth+half*(negone-dzd)*(one-dzd)*(one-dzd)
<         Szd(2)=sixth*(one-dzd)*(one-dzd)*(one-dzd)
<         Szd(3)=one-Szd(5)-Szd(4)-Szd(2)
<         iter3min = 2
<         iter3max = 5
<      endif
< #endif
<
<   ex0=0.
<   ey0=0.
<   ez0=0.
<   bx0=0.
<   by0=0.
<   bz0=0.
<
```

```
< #ifndef twoD
<   do iter3=iter3min,iter3max
<      do iter2=iter2min,iter2max
<           !lppp=(ip-3+iter1)+iy*(jp-3+iter2-1)+iz*(kp-3+iter3-1)
<           !if (lppp.ge.1 .and. lppp.le.lot) then
<           !    print *,"out of range"
<           !endif
<              ex0=ex0+sum(ex_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<              ey0=ey0+sum(ey_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<              ez0=ez0+sum(ez_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<              bx0=bx0+sum(bx_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<              by0=by0+sum(by_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<              bz0=bz0+sum(bz_p(ip-3+iter1min:ip-3+iter1max,jp-3+iter2,kp-3+iter3)*Sxp(iter1min:iter1max))*Syp(iter2)*Szp(iter3)
<
<      enddo
<   enddo
< #else
<   do iter2=iter2min,iter2max
<      do iter1=iter1min,iter1max
<         lpp=(ip-3+iter1)+iy*(jp-3+iter2-1)
<         lpd=(ip-3+iter1)+iy*(jd-3+iter2-1)
<         ldp=(id-3+iter1)+iy*(jp-3+iter2-1)
<         ldd=(id-3+iter1)+iy*(jd-3+iter2-1)
<
<         ex0=ex0+ex(ldp,1,1)*Sxd(iter1)*Syp(iter2)
<         ey0=ey0+ey(lpd,1,1)*Sxp(iter1)*Syd(iter2)
<         ez0=ez0+ez(lpp,1,1)*Sxp(iter1)*Syp(iter2)
<         bx0=bx0+bx(lpd,1,1)*Sxp(iter1)*Syd(iter2)
<         by0=by0+by(ldp,1,1)*Sxd(iter1)*Syp(iter2)
<         bz0=bz0+bz(ldd,1,1)*Sxd(iter1)*Syd(iter2)
<      enddo
<   enddo
< #endif
<
<   ex0=0.5*ex0*qm
<   ey0=0.5*ey0*qm
<   ez0=0.5*ez0*qm
<   bx0=0.5*bx0*qm*cinv
<   by0=0.5*by0*qm*cinv
<   bz0=0.5*bz0*qm*cinv
<
<   if(external_fields) then
<
<      call get_external_fields(real(p(npr)%x,sprec),p(npr)%y,&
<           p(npr)%z,ex_ext,ey_ext,ez_ext,bx_ext,by_ext,bz_ext)
<
<      bx0=bx0+bx_ext*0.5*qm*cinv
<      by0=by0+by_ext*0.5*qm*cinv
<      bz0=bz0+bz_ext*0.5*qm*cinv
<      ex0=ex0+ex_ext*0.5*qm
<      ey0=ey0+ey_ext*0.5*qm
<      ez0=ez0+ez_ext*0.5*qm
<
<   endif
<
<   !to be used later for synchrotron
<   bx00=bx0
<   by00=by0
<   bz00=bz0
<
<   !   First half electric acceleration, with relativity's gamma:
< #ifdef vay
<   !Use Vay 2008 particle mover
<   g=1./sqrt(1.+p(npr)%u**2+p(npr)%v**2+p(npr)%w**2) !reciprocal of the Lorentz factor
<   vx0=c*p(npr)%u*g !3-velocity of the particle
<   vy0=c*p(npr)%v*g
<   vz0=c*p(npr)%w*g
<
<   u1=c*p(npr)%u+2.*ex0+vy0*bz0-vz0*by0 !uprime, taking into account
<   !that cinv is already incorporated within B
<   v1=c*p(npr)%v+2.*ey0+vz0*bx0-vx0*bz0
<   w1=c*p(npr)%w+2.*ez0+vx0*by0-vy0*bx0
<
<   !Lorentz factor for uprime
<
<   ustar=cinv*(u1*bx0+v1*by0+w1*bz0)
<   sig=cinv*cinv*(c**2+u1**2+v1**2+w1**2)-(bx0**2+by0**2+bz0**2)
<   g=1./sqrt(0.5*(sig+sqrt(sig**2+4.*(bx0**2+by0**2+bz0**2+ustar**2))))
<   tx=bx0*g
<   ty=by0*g
<   tz=bz0*g
<   f=1./(1.+tx**2+ty**2+tz**2)
<
<   u0=f*(u1+(u1*tx+v1*ty+w1*tz)*tx+v1*tz-w1*ty)
<   v0=f*(v1+(u1*tx+v1*ty+w1*tz)*ty+w1*tx-u1*tz)
```

```
<   w0=f*(w1+(u1*tx+v1*ty+w1*tz)*tz+u1*ty-v1*tx)
< #else
<   !Use Boris algorithm
<   !   First half electric acceleration, with Lorentz gamma:
<   u0=c*p(npr)%u+ex0
<   v0=c*p(npr)%v+ey0
<   w0=c*p(npr)%w+ez0
<   !   First half magnetic rotation, with Lorentz gamma:
<   g=c/sqrt(c**2+u0**2+v0**2+w0**2)
<   bx0=g*bx0
<   by0=g*by0
<   bz0=g*bz0
<
<   f=2./(1.+bx0*bx0+by0*by0+bz0*bz0)
<   u1=(u0+v0*bz0-w0*by0)*f
<   v1=(v0+w0*bx0-u0*bz0)*f
<   w1=(w0+u0*by0-v0*bx0)*f
<   !   Second half mag. rot'n &   el. acc'n:
<   u0=u0+v1*bz0-w1*by0+ex0
<   v0=v0+w1*bx0-u1*bz0+ey0
<   w0=w0+u1*by0-v1*bx0+ez0
< #endif
<
<   !user stripe
<   !following tamburrini, cooling is implemented at this point,
<   !after the Lorentz push
<
<   !   Get normalized 4-velocity:
<   p(npr)%u=u0*cinv
<   p(npr)%v=v0*cinv
<   p(npr)%w=w0*cinv
<
<   ! stripe user
<   ! freeze particles, to verify the generation of the correct fields
<   ! (the three lines above should also be commented out)
<   !                    p(npr)%u=p(npr)%u
<   !                     p(npr)%v=p(npr)%v
<   !                     p(npr)%w=p(npr)%w
<   ! end stripe user
<
<   !   Position advance:
<   g=c/sqrt(c**2+u0**2+v0**2+w0**2)
<   p(npr)%x=p(npr)%x + p(npr)%u*g*c
<   p(npr)%y=p(npr)%y + p(npr)%v*g*c
<   p(npr)%z=p(npr)%z + p(npr)%w*g*c
<
< enddo
<
< end subroutine mover_3ord
```

6.5 deposit_particles()

```
1358,1361c424,427
<         maxx=mx-1.*(nghost/2)
<         minx=1.*(nghost/2+1)
<         maxy=my-1.*(nghost/2)
<         miny=1.*(nghost/2+1)
---
>         maxx=mx-2.
>         minx=3.
>         maxy=my-2.
>         miny=3.
1367,1368c433,434
<         maxz=mz-1.*(nghostz/2)
<         minz=1.*(nghostz/2+1)
---
>         maxz=mz-2.
>         minz=3.
1370,1371c436,437
<         minz=1.*(nghostz/2+1)
<         maxz=1.*(nghostz/2+1)+1
---
>         minz=3.
>         maxz= 6.-2.
```

The modifications are not tested with `#define cleancur` option used in the `tristan-mp-pu-master` version.

```
1377c443
< !#define cleancur
---
> #define cleancur
1379c445
```

```
< !#ifdef cleancur
---
> #ifdef cleancur
1382c448
< !#endif
---
> #endif
1391,1404c457,458
< #ifdef zzag
<                  call zigzag(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
< #ifdef dd1
<                  call densdecomp_1ord(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
< #ifdef dd2
<                  call densdecomp_2ord(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
< #ifdef dd3
<                call densdecomp_3ord(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
<
< !#ifdef findNaN
---
> !             call zigzag(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
> #ifdef findNaN
1406,1416c460,470
< !     xt=p(n1)%x  !hack -- this is needed for printout only
< !     yt=p(n1)%y
< !     zt=p(n1)%z
< !     if(xt<0 .or. yt<0 .or. zt<0 .or. isnan(xt*yt*zt) .or. xt>mx-1.or.yt>my-1 ) then !.or.zt>mz-1) then
< !         write(*,'(A,I7,A,3(I6,A),6(F8.4,A),3(I6,A))') "prb in ion dep, ind, proc, lap, rank: ",p(n1)%ind," ",p(n1)%proc," ",&
< !         lap," ",rank," x=",p(n1)%x," y=",p(n1)%y," z=",p(n1)%z," u=",p(n1)%u," v=",&
< !            p(n1)%v," w=",p(n1)%w, " mx,y,z=",mx," ",my," ",mz
<
< !         x0=5.
< !         y0=5.
< !         z0=5.
---
>       xt=p(n1)%x  !hack -- this is needed for printout only
>       yt=p(n1)%y
>       zt=p(n1)%z
>       if(xt<0 .or. yt<0 .or. zt<0 .or. isnan(xt*yt*zt) .or. xt>mx-1.or.yt>my-1 ) then !.or.zt>mz-1) then
>           write(*,'(A,I7,A,3(I6,A),6(F8.4,A),3(I6,A))') "prb in ion dep, ind, proc, lap, rank: ",p(n1)%ind," ",p(n1)%proc," ",&
>           lap," ",rank," x=",p(n1)%x," y=",p(n1)%y," z=",p(n1)%z," u=",p(n1)%u," v=",&
>             p(n1)%v," w=",p(n1)%w, " mx,y,z=",mx," ",my," ",mz
>
>           x0=5.
>           y0=5.
>           z0=5.
1424,1425c478,479
< !     endif
< !#endif
---
>       endif
> #endif
```

The `tristan-mp-pu-master` version implements the zigzag current deposit in the subroutine `deposit_particles()`, but external to the `zigzag` subroutine defined in `particles.F90`(meaning, `call zigzag(···)` is commented out, and the full routine is copied afterward, with a few additional parts and slightly different notation). For the modified code, the external copy of zigzag is commented out, and is instead called with the `zigzag` subroutine.

```
1429,1452c483,506
<         ! x1sp=x0
<         ! x2sp=p(n1)%x
<         ! y1sp=y0
<         ! y2sp=p(n1)%y
<
<         ! z1=z0
<         ! z2=p(n1)%z
<
<         ! i1=int(x1sp)
<         ! i2=int(x2sp)
<         ! j1=int(y1sp)
<         ! j2=int(y2sp)
<         ! k1=int(z1)
<         ! k2=int(z2)
<
<
<       ! xr=min(real(min(i1,i2)+1),max(real(max(i1,i2)),.5*(x1sp+x2sp)))
<       ! yr=min(real(min(j1,j2)+1),max(real(max(j1,j2)),.5*(y1sp+y2sp)))
<       ! zr=min(real(min(k1,k2)+1),max(real(max(k1,k2)),.5*(z1+z2)))
<
```

```
< ! #ifdef twoD
< !             k1=1
< !             k2=1
< ! #endif
---
>         x1sp=x0
>         x2sp=p(n1)%x
>         y1sp=y0
>         y2sp=p(n1)%y
>
>         z1=z0
>         z2=p(n1)%z
>
>         i1=int(x1sp)
>         i2=int(x2sp)
>         j1=int(y1sp)
>         j2=int(y2sp)
>         k1=int(z1)
>         k2=int(z2)
>
>
>       xr=min(real(min(i1,i2)+1),max(real(max(i1,i2)),.5*(x1sp+x2sp)))
>       yr=min(real(min(j1,j2)+1),max(real(max(j1,j2)),.5*(y1sp+y2sp)))
>       zr=min(real(min(k1,k2)+1),max(real(max(k1,k2)),.5*(z1+z2)))
>
> #ifdef twoD
>               k1=1
>               k2=1
> #endif
1456,1543c510,597
<       ! Fx1=-q*(xr-x1sp)
<       ! Fy1=-q*(yr-y1sp)
<       ! Fz1=-q*(zr-z1)
<
<       ! Wx1=.5*(x1sp+xr)-i1
<       ! Wy1=.5*(y1sp+yr)-j1
<
< ! #ifndef twoD
< !         Wz1=.5*(z1+zr)-k1
< ! #endif
<
< !     Wx2=.5*(x2sp+xr)-i2
< !     Wy2=.5*(y2sp+yr)-j2
<
< ! #ifndef twoD
< !             Wz2=.5*(z2+zr)-k2
< ! #endif
<
<       ! Fx2=-q*(x2sp-xr)
<       ! Fy2=-q*(y2sp-yr)
<       ! Fz2=-q*(z2-zr)
<
< ! #ifdef twoD
< !             Wz1=0
< !             Wz2=0
< ! #endif
<
<   ! onemWx1=1.-Wx1
<   ! onemWx2=1.-Wx2
<   ! onemWy1=1.-Wy1
<   ! onemWy2=1.-Wy2
<   ! onemWz1=1.-Wz1
<   ! onemWz2=1.-Wz2
<
< !   i1p1=i1+1
< !   i2p1=i2+1
< !   j1p1=j1+1
< !   j2p1=j2+1
< ! #ifndef twoD
< !   k1p1=k1+1
< !   k2p1=k2+1
< ! #endif
<
< ! #ifdef cleancur
<
< !   curx(i1,j1,k1)=curx(i1,j1,k1)+Fx1 * onemWy1 * onemWz1
< !   curx(i1,j1p1,k1)=curx(i1,j1p1,k1)+Fx1 * Wy1 * onemWz1
< ! #ifndef twoD
< !   curx(i1,j1,  k1p1)= curx(i1,j1,  k1p1)+Fx1 * onemWy1 * Wz1
< !   curx(i1,j1p1,k1p1)= curx(i1,j1p1,k1p1)+Fx1 *  Wy1    * Wz1
< ! #endif
<
< !  curx(i2,j2,k2)=curx(i2,j2,k2)+Fx2 * onemWy2 * onemWz2
< !   curx(i2,j2p1,k2)=curx(i2,j2p1,k2)+Fx2 * Wy2 * onemWz2
< ! #ifndef twoD
```

```
< !   curx(i2,j2,  k2p1)= curx(i2,j2,  k2p1)+Fx2 * onemWy2* Wz2
< !   curx(i2,j2p1,k2p1)= curx(i2,j2p1,k2p1)+Fx2 *  Wy2    * Wz2
< ! #endif
<
< !   cury(i1,j1,k1)=cury(i1,j1,k1)+Fy1 * onemWx1 * onemWz1
< !   cury(i1p1,j1,k1)=cury(i1p1,j1,k1)+Fy1 * Wx1 * onemWz1
< ! #ifndef twoD
< !   cury(i1  ,j1,k1p1)= cury(i1  ,j1,k1p1)+Fy1 * onemWx1 * Wz1
< !   cury(i1p1,j1,k1p1)= cury(i1p1,j1,k1p1)+Fy1 *  Wx1    * Wz1
< ! #endif
< !   cury(i2,j2,k2)=cury(i2,j2,k2)+Fy2 * onemWx2 * onemWz2
< !   cury(i2p1,j2,k2)=cury(i2p1,j2,k2)+Fy2 *  Wx2 * onemWz2
< ! #ifndef twoD
< !   cury(i2  ,j2,k2p1)= cury(i2  ,j2,k2p1)+Fy2 * onemWx2 * Wz2
< !   cury(i2p1,j2,k2p1)= cury(i2p1,j2,k2p1)+Fy2 *  Wx2    * Wz2
< ! #endif
<
<   ! curz(i1,j1,k1)=curz(i1,j1,k1)+ Fz1 * onemWx1 * onemWy1
<   ! curz(i1p1,j1,k1)=curz(i1p1,j1,k1)+Fz1 * Wx1 * onemWy1
<   ! curz(i1,j1p1,k1)=curz(i1,j1p1,k1)+Fz1 * onemWx1 * Wy1
<   ! curz(i1p1,j1p1,k1)=curz(i1p1,j1p1,k1)+Fz1 * Wx1 * Wy1
<
<   ! curz(i2,j2,k2)=curz(i2,j2,k2)+ Fz2 * onemWx2 * onemWy2
<   ! curz(i2p1,j2,k2)=curz(i2p1,j2,k2)+ Fz2 * Wx2 * onemWy2
<   ! curz(i2,j2p1,k2)=curz(i2,j2p1,k2)+ Fz2 * onemWx2 * Wy2
<   ! curz(i2p1,j2p1,k2)=curz(i2p1,j2p1,k2)+ Fz2 * Wx2 * Wy2
<
< !#endif !cleancur
< enddo !m,n=1,ions
<
< !#define dontSKIP
< !#ifdef dontSKIP
< !        pind(1:ions)=0
---
>       Fx1=-q*(xr-x1sp)
>       Fy1=-q*(yr-y1sp)
>       Fz1=-q*(zr-z1)
>
>       Wx1=.5*(x1sp+xr)-i1
>       Wy1=.5*(y1sp+yr)-j1
>
> #ifndef twoD
>         Wz1=.5*(z1+zr)-k1
> #endif
>
>       Wx2=.5*(x2sp+xr)-i2
>       Wy2=.5*(y2sp+yr)-j2
>
> #ifndef twoD
>               Wz2=.5*(z2+zr)-k2
> #endif
>
>       Fx2=-q*(x2sp-xr)
>       Fy2=-q*(y2sp-yr)
>       Fz2=-q*(z2-zr)
>
> #ifdef twoD
>               Wz1=0
>               Wz2=0
> #endif
>
>   onemWx1=1.-Wx1
>   onemWx2=1.-Wx2
>   onemWy1=1.-Wy1
>   onemWy2=1.-Wy2
>   onemWz1=1.-Wz1
>   onemWz2=1.-Wz2
>
>   i1p1=i1+1
>   i2p1=i2+1
>   j1p1=j1+1
>   j2p1=j2+1
> #ifndef twoD
>   k1p1=k1+1
>   k2p1=k2+1
> #endif
>
> #ifdef cleancur
>
>   curx(i1,j1,k1)=curx(i1,j1,k1)+Fx1 * onemWy1 * onemWz1
>   curx(i1,j1p1,k1)=curx(i1,j1p1,k1)+Fx1 * Wy1 * onemWz1
> #ifndef twoD
>   curx(i1,j1,  k1p1)= curx(i1,j1,  k1p1)+Fx1 * onemWy1 * Wz1
>   curx(i1,j1p1,k1p1)= curx(i1,j1p1,k1p1)+Fx1 *  Wy1    * Wz1
> #endif
```

```
>
>    curx(i2,j2,k2)=curx(i2,j2,k2)+Fx2 * onemWy2 * onemWz2
>    curx(i2,j2p1,k2)=curx(i2,j2p1,k2)+Fx2 * Wy2 * onemWz2
> #ifndef twoD
>    curx(i2,j2,  k2p1)= curx(i2,j2,  k2p1)+Fx2 * onemWy2* Wz2
>    curx(i2,j2p1,k2p1)= curx(i2,j2p1,k2p1)+Fx2 *  Wy2    * Wz2
> #endif
>
>    cury(i1,j1,k1)=cury(i1,j1,k1)+Fy1 * onemWx1 * onemWz1
>    cury(i1p1,j1,k1)=cury(i1p1,j1,k1)+Fy1 * Wx1 * onemWz1
> #ifndef twoD
>    cury(i1  ,j1,k1p1)= cury(i1  ,j1,k1p1)+Fy1 * onemWx1 * Wz1
>    cury(i1p1,j1,k1p1)= cury(i1p1,j1,k1p1)+Fy1 *  Wx1    * Wz1
> #endif
>    cury(i2,j2,k2)=cury(i2,j2,k2)+Fy2 * onemWx2 * onemWz2
>    cury(i2p1,j2,k2)=cury(i2p1,j2,k2)+Fy2 *  Wx2 * onemWz2
> #ifndef twoD
>    cury(i2  ,j2,k2p1)= cury(i2  ,j2,k2p1)+Fy2 * onemWx2 * Wz2
>    cury(i2p1,j2,k2p1)= cury(i2p1,j2,k2p1)+Fy2 *  Wx2    * Wz2
> #endif
>
>    curz(i1,j1,k1)=curz(i1,j1,k1)+ Fz1 * onemWx1 * onemWy1
>    curz(i1p1,j1,k1)=curz(i1p1,j1,k1)+Fz1 * Wx1 * onemWy1
>    curz(i1,j1p1,k1)=curz(i1,j1p1,k1)+Fz1 * onemWx1 * Wy1
>    curz(i1p1,j1p1,k1)=curz(i1p1,j1p1,k1)+Fz1 * Wx1 * Wy1
>
>    curz(i2,j2,k2)=curz(i2,j2,k2)+ Fz2 * onemWx2 * onemWy2
>    curz(i2p1,j2,k2)=curz(i2p1,j2,k2)+ Fz2 * Wx2 * onemWy2
>    curz(i2,j2p1,k2)=curz(i2,j2p1,k2)+ Fz2 * onemWx2 * Wy2
>    curz(i2p1,j2p1,k2)=curz(i2p1,j2p1,k2)+ Fz2 * Wx2 * Wy2
>
> #endif !cleancur
>         enddo !m,n=1,ions
>
> #define dontSKIP
> #ifdef dontSKIP
>         pind(1:ions)=0
1570,1572c624,626
<         if(periodicz.eq.0 .and. in) then
<            in=(p(n1)%z+mzcum .gt. z1in) .and. (p(n1)%z+mzcum .lt. z2in)
<         endif
---
>               if(periodicz.eq.0 .and. in) then
>                     in=(p(n1)%z+mzcum .gt. z1in) .and. (p(n1)%z+mzcum .lt. z2in)
>               endif
1576,1614c630,668
<         if(.not. in) then                          !imp
<            perx=0                                  !imp
<            pery=0                                  !imp
<            perz=0                                  !imp
<         endif                                      !imp
<
<         ! to send to another processor
<         if(perx .ne. 0 .and. in .and. sizex .ne. 1) then !imp
<            in=.false.                                !imp
<            pery=0                                      !imp
<            perz=0                                      !imp
<         endif                                          !imp
<
<         ! assume non-uniform mx
<         if(perx .lt. 0 .and. sizex .ne. 1) then
<            ! minus rank (-x direction)
<            i1=(rank/sizex)*sizex + modulo(rank-1,sizex)
<            perx=-(mxl(i1+1)-1.*nghost)
<         endif
<
<         p(n1)%x=p(n1)%x-perx                       !imp
<
<         ! to send to another processor
<         if(pery .ne. 0 .and. in .and. sizey .ne. 1) then   !imp
<            in=.false.                                      !imp
<            perx=0                                          !imp
<            perz=0                                          !imp
<         endif                                              !imp
<
<         ! assume non-uniform my
<         if(pery .lt. 0 .and. sizey .ne. 1) then                                                  !imp
<            ! left rank        (-y direction)                                                    !imp
<            !                   j1=modulo((rank/sizex - 1),sizey)*sizex + modulo(rank,sizex)    !imp
<            j1=modulo(rank/sizex - 1,sizey)*sizex+rank/(sizex*sizey)*(sizex*sizey) &             !imp
<                 + modulo(rank,sizex)                                                           !imp
<            pery=-(myl(j1+1)-1.*nghost)                                                         !imp
<         endif
<
<         p(n1)%y=p(n1)%y-pery
```

```
---
>               if(.not. in) then
>                       perx=0
>                       pery=0
>                       perz=0
>               endif
>
>               ! to send to another processor
>               if(perx .ne. 0 .and. in .and. sizex .ne. 1) then
>                       in=.false.
>                       pery=0
>                       perz=0
>               endif
>
>               ! assume non-uniform mx
>               if(perx .lt. 0 .and. sizex .ne. 1) then
>                       ! minus rank (-x direction)
>                       i1=(rank/sizex)*sizex + modulo(rank-1,sizex)
>                       perx=-(mxl(i1+1)-5.)
>               endif
>
>               p(n1)%x=p(n1)%x-perx
>
>               ! to send to another processor
>               if(pery .ne. 0 .and. in .and. sizey .ne. 1) then
>                       in=.false.
>                       perx=0
>                       perz=0
>               endif
>
>               ! assume non-uniform my
>               if(pery .lt. 0 .and. sizey .ne. 1) then
>                       ! left rank     (-y direction)
> !                     j1=modulo((rank/sizex - 1),sizey)*sizex + modulo(rank,sizex)
>                       j1=modulo(rank/sizex - 1,sizey)*sizex+rank/(sizex*sizey)*(sizex*sizey) &
>                               + modulo(rank,sizex)
>                       pery=-(myl(j1+1)-5.)
>               endif
>
>               p(n1)%y=p(n1)%y-pery
1618,1629c672,683
<         if(perz .ne. 0 .and. in ) then
<            in=.false.
<            perx=0
<            pery=0
<         endif
<         ! assume non-uniform mz
<         if(perz .lt. 0) then
<            ! down rank        (-z direction)
<            k1=modulo(rank/(sizex*sizey) - 1,sizez)*(sizex*sizey) + &
<                 modulo(rank,sizex*sizey)
<            perz=-(mzl(k1+1)-1.*nghostz)
<         endif
---
>               if(perz .ne. 0 .and. in ) then
>                       in=.false.
>                       perx=0
>                       pery=0
>               endif
>               ! assume non-uniform mz
>               if(perz .lt. 0) then
>                       ! down rank     (-z direction)
>                       k1=modulo(rank/(sizex*sizey) - 1,sizez)*(sizex*sizey) + &
>                             modulo(rank,sizex*sizey)
>                       perz=-(mzl(k1+1)-5.)
>               endif

1705c759
< !#endif !dontSKIP
---
> #endif !dontSKIP
1728c782
< !#ifdef findNaN
---
> #ifdef findNaN
1730,1740c784,794
< !     xt=p(n1)%x  !hack -- this is needed for printout only
< !     yt=p(n1)%y
< !     zt=p(n1)%z
< !     if(xt<0 .or. yt<0 .or. zt<0 .or. isnan(xt*yt*zt) .or. xt>mx-1.or.yt>my-1 ) then !.or.zt>mz-1) then
< !         write(*,'(A,I7,A,3(I6,A),6(F8.4,A),3(I6,A))') "prb in lec dep, ind, proc, lap, rank: ",p(n1)%ind," ",p(n1)%proc," ",&
< !         lap," ",rank," x=",p(n1)%x," y=",p(n1)%y," z=",p(n1)%z," u=",p(n1)%u," v=",&
< !            p(n1)%v," w=",p(n1)%w, " mx,y,z=",mx," ",my," ",mz
<
< !         x0=5.
```

```
< !         y0=5.
< !         z0=5.
---
>       xt=p(n1)%x  !hack -- this is needed for printout only
>       yt=p(n1)%y
>       zt=p(n1)%z
>       if(xt<0 .or. yt<0 .or. zt<0 .or. isnan(xt*yt*zt) .or. xt>mx-1.or.yt>my-1 ) then !.or.zt>mz-1) then
>           write(*,'(A,I7,A,3(I6,A),6(F8.4,A),3(I6,A))') "prb in lec dep, ind, proc, lap, rank: ",p(n1)%ind," ",p(n1)%proc," ",&
>           lap," ",rank," x=",p(n1)%x," y=",p(n1)%y," z=",p(n1)%z," u=",p(n1)%u," v=",&
>              p(n1)%v," w=",p(n1)%w, " mx,y,z=",mx," ",my," ",mz
>
>           x0=5.
>           y0=5.
>           z0=5.
1748,1761c802
< !     endif
< !#endif
<
< #ifdef zzag
<                call zigzag(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
< #ifdef dd1
<                  call densdecomp_1ord(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
< #ifdef dd2
<                  call densdecomp_2ord(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
< #endif
< #ifdef dd3
<                call densdecomp_3ord(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
---
>       endif
1763a805,807
>
> !             call zigzag(p(n1)%x,p(n1)%y,p(n1)%z,x0,y0,z0,in)
>
1765,1788c809,832
< !         x1sp=x0
< !         x2sp=p(n1)%x
< !         y1sp=y0
< !         y2sp=p(n1)%y
<
< !         z1=z0
< !         z2=p(n1)%z
<
< !         i1=int(x1sp)
< !         i2=int(x2sp)
< !         j1=int(y1sp)
< !         j2=int(y2sp)
< !         k1=int(z1)
< !         k2=int(z2)
<
<
< !     xr=min(real(min(i1,i2)+1),max(real(max(i1,i2)),.5*(x1sp+x2sp)))
< !     yr=min(real(min(j1,j2)+1),max(real(max(j1,j2)),.5*(y1sp+y2sp)))
< !     zr=min(real(min(k1,k2)+1),max(real(max(k1,k2)),.5*(z1+z2)))
<
< !#ifdef twoD
< !             k1=1
< !             k2=1
< !#endif
---
>         x1sp=x0
>         x2sp=p(n1)%x
>         y1sp=y0
>         y2sp=p(n1)%y
>
>         z1=z0
>         z2=p(n1)%z
>
>         i1=int(x1sp)
>         i2=int(x2sp)
>         j1=int(y1sp)
>         j2=int(y2sp)
>         k1=int(z1)
>         k2=int(z2)
>
>
>       xr=min(real(min(i1,i2)+1),max(real(max(i1,i2)),.5*(x1sp+x2sp)))
>       yr=min(real(min(j1,j2)+1),max(real(max(j1,j2)),.5*(y1sp+y2sp)))
>       zr=min(real(min(k1,k2)+1),max(real(max(k1,k2)),.5*(z1+z2)))
>
> #ifdef twoD
>               k1=1
>               k2=1
> #endif
```

```
1792,1833c836,877
< !      Fx1=-q*(xr-x1sp)
< !      Fy1=-q*(yr-y1sp)
< !      Fz1=-q*(zr-z1)
<
< !      Wx1=.5*(x1sp+xr)-i1
< !      Wy1=.5*(y1sp+yr)-j1
<
< !#ifndef twoD
< !         Wz1=.5*(z1+zr)-k1
< !#endif
<
< !      Wx2=.5*(x2sp+xr)-i2
< !      Wy2=.5*(y2sp+yr)-j2
<
< !#ifndef twoD
< !              Wz2=.5*(z2+zr)-k2
< !#endif
<
< !      Fx2=-q*(x2sp-xr)
< !      Fy2=-q*(y2sp-yr)
< !      Fz2=-q*(z2-zr)
<
< !#ifdef twoD
< !              Wz1=0
< !              Wz2=0
< !#endif
<
<    ! onemWx1=1.-Wx1
<    ! onemWx2=1.-Wx2
<    ! onemWy1=1.-Wy1
<    ! onemWy2=1.-Wy2
<    ! onemWz1=1.-Wz1
<    ! onemWz2=1.-Wz2
<
< !   i1p1=i1+1
< !   i2p1=i2+1
< !   j1p1=j1+1
< !   j2p1=j2+1
< ! #ifndef twoD
< !   k1p1=k1+1
< !   k2p1=k2+1
< ! #endif
---
>        Fx1=-q*(xr-x1sp)
>        Fy1=-q*(yr-y1sp)
>        Fz1=-q*(zr-z1)
>
>        Wx1=.5*(x1sp+xr)-i1
>        Wy1=.5*(y1sp+yr)-j1
>
> #ifndef twoD
>          Wz1=.5*(z1+zr)-k1
> #endif
>
>        Wx2=.5*(x2sp+xr)-i2
>        Wy2=.5*(y2sp+yr)-j2
>
> #ifndef twoD
>                Wz2=.5*(z2+zr)-k2
> #endif
>
>        Fx2=-q*(x2sp-xr)
>        Fy2=-q*(y2sp-yr)
>        Fz2=-q*(z2-zr)
>
> #ifdef twoD
>                Wz1=0
>                Wz2=0
> #endif
>
>    onemWx1=1.-Wx1
>    onemWx2=1.-Wx2
>    onemWy1=1.-Wy1
>    onemWy2=1.-Wy2
>    onemWz1=1.-Wz1
>    onemWz2=1.-Wz2
>
>    i1p1=i1+1
>    i2p1=i2+1
>    j1p1=j1+1
>    j2p1=j2+1
> #ifndef twoD
>    k1p1=k1+1
>    k2p1=k2+1
```

```
> #endif
1835c879
< !#ifdef cleancur
---
> #ifdef cleancur
1840,1873c884,917
< !  curx(i1,j1,k1)=curx(i1,j1,k1)+Fx1 * onemWy1 * onemWz1
< !  curx(i1,j1p1,k1)=curx(i1,j1p1,k1)+Fx1 * Wy1 * onemWz1
< !#ifndef twoD
< !  curx(i1,j1,  k1p1)= curx(i1,j1,  k1p1)+Fx1 * onemWy1 * Wz1
< !  curx(i1,j1p1,k1p1)= curx(i1,j1p1,k1p1)+Fx1 *  Wy1    * Wz1
< !#endif
<
< !  curx(i2,j2,k2)=curx(i2,j2,k2)+Fx2 * onemWy2 * onemWz2
< !  curx(i2,j2p1,k2)=curx(i2,j2p1,k2)+Fx2 * Wy2 * onemWz2
< !#ifndef twoD
< !  curx(i2,j2,  k2p1)= curx(i2,j2,  k2p1)+Fx2 * onemWy2* Wz2
< !  curx(i2,j2p1,k2p1)= curx(i2,j2p1,k2p1)+Fx2 *  Wy2    * Wz2
< !#endif
<
< !  cury(i1,j1,k1)=cury(i1,j1,k1)+Fy1 * onemWx1 * onemWz1
< !  cury(i1p1,j1,k1)=cury(i1p1,j1,k1)+Fy1 * Wx1 * onemWz1
< !#ifndef twoD
< !  cury(i1  ,j1,k1p1)= cury(i1  ,j1,k1p1)+Fy1 * onemWx1 * Wz1
< !  cury(i1p1,j1,k1p1)= cury(i1p1,j1,k1p1)+Fy1 *  Wx1    * Wz1
< !#endif
< !  cury(i2,j2,k2)=cury(i2,j2,k2)+Fy2 * onemWx2 * onemWz2
< !  cury(i2p1,j2,k2)=cury(i2p1,j2,k2)+Fy2 *  Wx2    * onemWz2
< !#ifndef twoD
< !  cury(i2  ,j2,k2p1)= cury(i2  ,j2,k2p1)+Fy2 * onemWx2 * Wz2
< !  cury(i2p1,j2,k2p1)= cury(i2p1,j2,k2p1)+Fy2 *  Wx2    * Wz2
< !#endif
< !  curz(i1,j1,k1)=curz(i1,j1,k1)+ Fz1 * onemWx1 * onemWy1
< !  curz(i1p1,j1,k1)=curz(i1p1,j1,k1)+Fz1 * Wx1 * onemWy1
< !  curz(i1,j1p1,k1)=curz(i1,j1p1,k1)+Fz1 * onemWx1 * Wy1
< !  curz(i1p1,j1p1,k1)=curz(i1p1,j1p1,k1)+Fz1 * Wx1 * Wy1
< !  curz(i2,j2,k2)=curz(i2,j2,k2)+ Fz2 * onemWx2 * onemWy2
< !  curz(i2p1,j2,k2)=curz(i2p1,j2,k2)+ Fz2 * Wx2 * onemWy2
< !  curz(i2,j2p1,k2)=curz(i2,j2p1,k2)+ Fz2 * onemWx2 * Wy2
< !  curz(i2p1,j2p1,k2)=curz(i2p1,j2p1,k2)+ Fz2 * Wx2 * Wy2
---
>    curx(i1,j1,k1)=curx(i1,j1,k1)+Fx1 * onemWy1 * onemWz1
>    curx(i1,j1p1,k1)=curx(i1,j1p1,k1)+Fx1 * Wy1 * onemWz1
> #ifndef twoD
>    curx(i1,j1,  k1p1)= curx(i1,j1,  k1p1)+Fx1 * onemWy1 * Wz1
>    curx(i1,j1p1,k1p1)= curx(i1,j1p1,k1p1)+Fx1 *  Wy1    * Wz1
> #endif
>
>    curx(i2,j2,k2)=curx(i2,j2,k2)+Fx2 * onemWy2 * onemWz2
>    curx(i2,j2p1,k2)=curx(i2,j2p1,k2)+Fx2 * Wy2 * onemWz2
> #ifndef twoD
>    curx(i2,j2,  k2p1)= curx(i2,j2,  k2p1)+Fx2 * onemWy2* Wz2
>    curx(i2,j2p1,k2p1)= curx(i2,j2p1,k2p1)+Fx2 *  Wy2    * Wz2
> #endif
>
>    cury(i1,j1,k1)=cury(i1,j1,k1)+Fy1 * onemWx1 * onemWz1
>    cury(i1p1,j1,k1)=cury(i1p1,j1,k1)+Fy1 * Wx1 * onemWz1
> #ifndef twoD
>    cury(i1  ,j1,k1p1)= cury(i1  ,j1,k1p1)+Fy1 * onemWx1 * Wz1
>    cury(i1p1,j1,k1p1)= cury(i1p1,j1,k1p1)+Fy1 *  Wx1    * Wz1
> #endif
>    cury(i2,j2,k2)=cury(i2,j2,k2)+Fy2 * onemWx2 * onemWz2
>    cury(i2p1,j2,k2)=cury(i2p1,j2,k2)+Fy2 *  Wx2    * onemWz2
> #ifndef twoD
>    cury(i2  ,j2,k2p1)= cury(i2  ,j2,k2p1)+Fy2 * onemWx2 * Wz2
>    cury(i2p1,j2,k2p1)= cury(i2p1,j2,k2p1)+Fy2 *  Wx2    * Wz2
> #endif
>    curz(i1,j1,k1)=curz(i1,j1,k1)+ Fz1 * onemWx1 * onemWy1
>    curz(i1p1,j1,k1)=curz(i1p1,j1,k1)+Fz1 * Wx1 * onemWy1
>    curz(i1,j1p1,k1)=curz(i1,j1p1,k1)+Fz1 * onemWx1 * Wy1
>    curz(i1p1,j1p1,k1)=curz(i1p1,j1p1,k1)+Fz1 * Wx1 * Wy1
>    curz(i2,j2,k2)=curz(i2,j2,k2)+ Fz2 * onemWx2 * onemWy2
>    curz(i2p1,j2,k2)=curz(i2p1,j2,k2)+ Fz2 * Wx2 * onemWy2
>    curz(i2,j2p1,k2)=curz(i2,j2p1,k2)+ Fz2 * onemWx2 * Wy2
>    curz(i2p1,j2p1,k2)=curz(i2p1,j2p1,k2)+ Fz2 * Wx2 * Wy2
1876c920
< !#endif !cleancur
---
> #endif !cleancur
1878,1879c922,923
< !107 continue
< enddo !n=1,lecs
---
> 107 continue
>    enddo !n=1,lecs
```

```
1935c979
<                       perx=-(mxl(i1+1)-1.*nghost)
---
>                       perx=-(mxl(i1+1)-5.)
1953c996
<                       pery=-(myl(j1+1)-1.*nghost)
---
>                       pery=-(myl(j1+1)-5.)
1956c999
<               p(n1)%y=p(n1)%y-pery
---
>               p(n1)%y=p(n1)%y-pery
1970c1013
<                       perz=-(mzl(k1+1)-1.*nghostz)
---
>                       perz=-(mzl(k1+1)-5.)
2022d1064
<
2032d1073
<           !print *, "discarding", pind(n), n, p(n)%x, p(n)%y, p(n)%z, lecs, lecs0, lap, rank
```

7 filter.F90

7.1 apply_filter1_opt()

```
29,30c29,30
<       istr=nghost/2+1
<       ifin=mx-(nghost/2+1)
---
>       istr=2  +1
>       ifin=mx-1 -2
73,74c73
<                       !call copy_layrx1_opt(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
<                         call copy_layrx1_opt(curx,cury,curz,mx,my,mz,nghost/2,mx-(nghost/2+1),mx-(nghost/2),(nghost/2+1))
---
>                       call copy_layrx1_opt(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
77c76
<                       call copy_layrx2_opt(curx,cury,curz,mx,my,mz,nghost/2,mx-(nghost/2+1),mx-(nghost/2),(nghost/2+1))
---
>                       call copy_layrx2_opt(curx,cury,curz,mx,my,mz,2,mx-3,mx-2,3)
82,83c81
<                       !call copy_layry1_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
<                         call copy_layry1_opt(curx,cury,curz,mx,my,mz,nghost/2,my-(nghost/2+1),my-nghost/2,(nghost/2+1))
---
>                       call copy_layry1_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
86,87c84
<                       !call copy_layry2_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
<                         call copy_layry2_opt(curx,cury,curz,mx,my,mz,nghost/2,my-(nghost/2+1),my-nghost/2,(nghost/2+1))
---
>                       call copy_layry2_opt(curx,cury,curz,mx,my,mz,2,my-3,my-2,3)
92,93c89
<                               !call copy_layrz1_opt(curx,cury,curz,mx,my,mz,2,mz-3,mz-2,3)
<                                 call copy_layrz1_opt(curx,cury,curz,mx,my,mz,nghostz/2,mz-(nghostz/2+1),mz-nghostz/2,nghostz/2+1)
---
>                                call copy_layrz1_opt(curx,cury,curz,mx,my,mz,2,mz-3,mz-2,3)
97c93
<                               call copy_layrz2_opt(curx,cury,curz,mx,my,mz,nghostz/2,mz-(nghostz/2+1),mz-nghostz/2,nghostz/2+1)
---
>                               call copy_layrz2_opt(curx,cury,curz,mx,my,mz,2,mz-3,mz-2,3)
102c98
<               do k=nghostz/2+1,mz-(nghostz/2+1)
---
>               do k=3,mz-3
106c102
<               do j=nghost/2+1,my-(nghost/2+1)
---
>               do j=3,my-3
128c124
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
132c128
<                       do j=(nghost/2+1),my-(nghost/2+1)
---
>                       do j=3,my-3
141c137
<               do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>               do k=3,mz-3
145c141
<               do j=(nghost/2+1),my-(nghost/2+1)
---
>               do j=3,my-3
168c164
```

```
<                do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                do k=3,mz-3
172c168
<                        do j=nghost/2+1,my-(nghost/2+1)
---
>                        do j=3,my-3
182c178
<                do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                do k=3,mz-3
186c182
<                do j=nghost/2+1,my-nghost/2+1
---
>                do j=3,my-3
207c203
<                do k=(nghostz/2+1),mz-(nghostz/2+1)
---
>                do k=3,mz-3
211c207
<                        do j=nghost/2+1,my-nghost/2+1
---
>                        do j=3,my-3
```

8 optimized_filters.F90

The option `Dfilter2` is undocumented: ghost cell modifications are untested with filter2 (also filter3).

8.1 apply_filter2_opt()

```
23,24c23,24
<       istr=nghost/2+1!3
<       ifin=mx-(nghost/2+1)!-3
---
>       istr=3
>       ifin=mx-3
38,39c38,39
<       jstr=nghost/2+1
<       jfin=my-(nghost/2+1)
---
>       jstr=3
>       jfin=my-3
43,44c43,44
<       kstr=nghostz/2+1
<       kfin=mz-(nghostz/2+1) !2,mz-1 !3,mz-3
---
>       kstr=3
>       kfin=mz-3 !2,mz-1 !3,mz-3
```

8.2 apply_filter2_copy_opt()

```
248,249c248,249
<       istr=nghost/2+1
<       ifin=mx-(nghost/2+1)
---
>       istr=3
>       ifin=mx-3
263,264c263,264
<       jstr=nghost/2+1
<       jfin=my-(nghost/2+1)
---
>       jstr=3
>       jfin=my-3
268,269c268,269
<       kstr=nghostz/2+1
<       kfin=mz-(nghostz/2+1) !2,mz-1 !3,mz-3
---
>       kstr=3
>       kfin=mz-3 !2,mz-1 !3,mz-3
```

8.3 filter_x()

```
524c524
<                               if(ifin .ne. mx-(nghost/2+1)) then
---
>                               if(ifin .ne. mx-3) then
```

8.4 chunk_filter()

```
1003,1004c1003,1004
<       istr=nghost/2+1
<       ifin=mx-(nghost/2+1)
---
>       istr=3
>       ifin=mx-3
1018,1019c1018,1019
<       jstr=nghost/2+1
<       jfin=my-(nghost/2+1)
---
>       jstr=3
>       jfin=my-3
1023,1024c1023,1024
<       kstr=nghostz/2+1
<       kfin=mz-(nghostz/2+1) !2,mz-1 !3,mz-3
---
>       kstr=3
>       kfin=mz-3 !2,mz-1 !3,mz-3
1036,1042c1036,1042
<       num_x_chunks = (mx-nghost)/x_chunk_len
<       num_y_chunks = (my-nghost)/y_chunk_len
<       num_z_chunks = (mz-nghostz)/z_chunk_len
<
<       last_x_size = (mx-nghost) - num_x_chunks*x_chunk_len
<       last_y_size = (my-nghost) - num_y_chunks*y_chunk_len
<       last_z_size = (mz-nghostz) - num_z_chunks*z_chunk_len
---
>       num_x_chunks = (mx-5)/x_chunk_len
>       num_y_chunks = (my-5)/y_chunk_len
>       num_z_chunks = (mz-5)/z_chunk_len
>
>       last_x_size = (mx-5) - num_x_chunks*x_chunk_len
>       last_y_size = (my-5) - num_y_chunks*y_chunk_len
>       last_z_size = (mz-5) - num_z_chunks*z_chunk_len
1077,1080c1077,1079
<                                       =xghost(:ntimes,(nghost/2+1)+(j-1)&
<                                         *y_chunk_len:(nghost/2+1)+ j*y_chunk_len-1,&
<                                         (nghostz/2+1)+(k-1)&
<                                         *z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       =xghost(:ntimes,3+(j-1)*y_chunk_len:3+ &
>                                       j*y_chunk_len-1,3+(k-1)*z_chunk_len:3+ &
>                                       k*z_chunk_len-1)
1084,1087c1083,1086
<                                       =cur((nghost/2+1)+(i-1)&
<                                         *x_chunk_len-ntimes:(nghost/2+1)+(i-1)*x_chunk_len-1,&
<                                         (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                         (nghost/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       =cur(3+(i-1)*x_chunk_len-ntimes:3+ &
>                                       (i-1)*x_chunk_len-1,3+(j-1)*y_chunk_len:3+ &
>                                       j*y_chunk_len-1,3+(k-1)*z_chunk_len:3+ &
>                                       k*z_chunk_len-1)
1092,1097c1091
<                                       chunk(ntimes+x_chunk_len+1:,&
<                                               ntimes+1:ntimes+y_chunk_len,&
<                                               ntimes+1:ntimes+z_chunk_len) &
<                                               =xghost(ntimes+1:,&
<                                               (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                               (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       chunk(ntimes+x_chunk_len+1:,ntimes+1:ntimes+y_chunk_len,ntimes+1:ntimes+z_chunk_len) ...
1099,1104c1093
<                                       chunk(ntimes+x_chunk_len+1:,&
<                                               ntimes+1:ntimes+y_chunk_len,&
<                                               ntimes+1:ntimes+z_chunk_len)&
<                                               =cur((nghost/2+1)+i*x_chunk_len:(nghost/2+1)+i*x_chunk_len+ntimes-1,&
<                                               (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                               (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       chunk(ntimes+x_chunk_len+1:,ntimes+1:ntimes+y_chunk_len,ntimes+1:ntimes+z_chunk_len) ...
1109,1112c1098
<                                       chunk(ntimes+1:ntimes+x_chunk_len,:ntimes,&
<                                               ntimes+1:ntimes+z_chunk_len)&
<                                               =yghost((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                               :ntimes,(nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,:ntimes,ntimes+1:ntimes+z_chunk_len) ...
1114,1118c1100
<                                       chunk(ntimes+1:ntimes+x_chunk_len,:ntimes,&
<                                               ntimes+1:ntimes+z_chunk_len)&
<                                               =cur((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                               (nghost/2+1)+(j-1)*y_chunk_len-ntimes:(nghost/2+1)+(j-1)*y_chunk_len-1,&
<                                               (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
```

```
>                                       chunk(ntimes+1:ntimes+x_chunk_len,:ntimes,ntimes+1:ntimes+z_chunk_len) ...
1123,1128c1105
<                                       chunk(ntimes+1:ntimes+x_chunk_len,&
<                                             ntimes+y_chunk_len+1:,&
<                                             ntimes+1:ntimes+z_chunk_len)&
<                                             =yghost((nghost/2+1)+(i-1)&
<                                             *x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,ntimes+1:,&
<                                             (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+y_chunk_len+1:,ntimes+1:ntimes+z_chunk_len) ...
1130,1134c1107
<                                       chunk(ntimes+1:ntimes+x_chunk_len,&
<                                             ntimes+y_chunk_len+1:,ntimes+1:ntimes+z_chunk_len)&
<                                             =cur((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)&
<                                             +i*x_chunk_len-1,(nghost/2+1)+j*y_chunk_len:(nghost/2+1)+j*y_chunk_len+ntimes-1,&
<                                             (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+y_chunk_len+1:,ntimes+1:ntimes+z_chunk_len) ...
1140,1143c1113
<                                       chunk(ntimes+1:ntimes+x_chunk_len,&
<                                         ntimes+1:ntimes+y_chunk_len,:ntimes)&
<                                         =zghost((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                         (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,:ntimes)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,:ntimes) ...
1145,1148c1115
<                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,:ntimes)&
<                                         =cur((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                         (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                         (nghostz/2+1)+(k-1)*z_chunk_len-ntimes:(nghostz/2+1)+(k-1)*z_chunk_len-1)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,:ntimes) ...
1153,1155c1120
<                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+z_chunk_len+1:) &
<                                         =zghost((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                         (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,ntimes+1:)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+z_chunk_len+1:) ...
1157,1160c1122
<                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+z_chunk_len+1:)&
<                                         =cur((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                         (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                         (nghostz/2+1)+k*z_chunk_len:(nghostz/2+1)+k*z_chunk_len+ntimes-1)
---
>                                       chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+z_chunk_len+1:) ...
1165,1168c1127
<                                 chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+1:ntimes+z_chunk_len)&
<                                   =cur((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                   (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                   (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)
---
>                                 chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+1:ntimes+z_chunk_len) ...
1171,1172c1130
<                                 call chunk_filter_x(chunk,2,ntimes+x_chunk_len+ntimes-1,&
<                                                     1,ntimes+y_chunk_len+ntimes,1,ntimes+z_chunk_len+ntimes)
---
>                                 call chunk_filter_x(chunk,2,ntimes+x_chunk_len+ntimes-1, ...
1175,1176c1133
<                                 call chunk_filter_y(chunk,1,ntimes+x_chunk_len+ntimes,2,&
<                                                     ntimes+y_chunk_len+ntimes-1,1,ntimes+z_chunk_len+ntimes)
---
>                                 call chunk_filter_y(chunk,1,ntimes+x_chunk_len+ntimes,2, ...
1179,1180c1136
<                                 call chunk_filter_z(chunk,1,ntimes+x_chunk_len+ntimes,&
<                                                     1,ntimes+y_chunk_len+ntimes,2,ntimes+z_chunk_len+ntimes-1)
---
>                                 call chunk_filter_z(chunk,1,ntimes+x_chunk_len+ntimes,1,ntimes+y_chunk_len+ntimes, ...
1184,1187c1140
<                                 temp((nghost/2+1)+(i-1)*x_chunk_len:(nghost/2+1)+i*x_chunk_len-1,&
<                                        (nghost/2+1)+(j-1)*y_chunk_len:(nghost/2+1)+j*y_chunk_len-1,&
<                                        (nghostz/2+1)+(k-1)*z_chunk_len:(nghostz/2+1)+k*z_chunk_len-1)&
<                                        =chunk(ntimes+1:ntimes+x_chunk_len,ntimes+1:ntimes+y_chunk_len,ntimes+1:ntimes+z_chunk_len)
---
>                                 temp(3+(i-1)*x_chunk_len:3+i*x_chunk_len-1,3+(j-1)*y_chunk_len:3+j*y_chunk_len-1,3+(k-1)* ...
1192,1193c1145
<       cur((nghost/2+1):mx-(nghost/2+1),(nghost/2+1):my-(nghost/2+1),(nghost/2+1):mz-(nghostz/2+1))&
<            =temp((nghost/2+1):mx-(nghost/2+1),(nghost/2+1):my-(nghost/2+1),(nghostz/2+1):mz-(nghostz/2+1))
---
>       cur(3:mx-3,3:my-3,3:mz-3)=temp(3:mx-3,3:my-3,3:mz-3)
```

9 output.F90

9.1 save_spectrum_2d()

```
836,839c835,838
<               mxmin=int((mx0-nghost)*.5)-1200+(nghost/2+1)
<               mxmax=int((mx0-nghost)*.5)+400+(nghost/2+1)
<               mymin=(nghost/2+1)
<               mymax=my0-(nghost/2)
---
>               mxmin=int((mx0-5)*.5)-1200+3
>               mxmax=int((mx0-5)*.5)+400+3
>               mymin=3
>               mymax=my0-2
```

9.2 output_tot()

```
2648,2649c2647,2648
<       kstart=(nghostz/2+1)
<       kfinish=mz-(nghostz/2+1)
---
>       kstart=3
>       kfinish=mz-3
2669,2670c2668,2669
<       jstart=nghost/2+1
<       jfinish=my-(nghost/2+1)
---
>       jstart=3
>       jfinish=my-3
2686,2687c2685,2686
<       istart=nghost/2+1
<       ifinish=mx-(nghost/2+1)
---
>       istart=3
>       ifinish=mx-3
3297c3296
<       !goto 127
---
> !     goto 127
3546,3549c3545,3546
<             if(varname.eq.'gammae') then
<                temporary_vec(1:strd_lecs)=sqrt(1. &
<                     +(p(lecs_str_ind)%u**2+p(lecs_str_ind)%v**2+p(lecs_str_ind)%w**2))
<             endif
---
>             if(varname.eq.'gammae') temporary_vec(1:strd_lecs)=sqrt(1. &
>               +(p(lecs_str_ind)%u**2+p(lecs_str_ind)%v**2+p(lecs_str_ind)%w**2))
```

9.3 output_hug()

```
4006,4007c4003,4004
<       kstart=(nghostz/2+1)
<       kfinish=mz-(nghostz/2+1)
---
>       kstart=3
>       kfinish=mz-3
4011c4008
< 1110  if(modulo(kstart+(rank/sizey)*(mzall-nghostz)-1*0,istep) .eq. 0) goto 1120
---
> 1110  if(modulo(kstart+(rank/sizey)*(mzall-5)-1*0,istep) .eq. 0) goto 1120
4016c4013
< 1130  if(modulo(kfinish+(rank/sizey)*(mzall-nghostz)-1*0,istep) .eq. 0) goto 1140
---
> 1130  if(modulo(kfinish+(rank/sizey)*(mzall-5)-1*0,istep) .eq. 0) goto 1140
4028,4029c4025,4026
<       jstart=(nghost/2+1)
<       jfinish=my-(nghost/2+1)
---
>       jstart=3
>       jfinish=my-3
4033c4030
< 1210  if(modulo(jstart+modulo(rank,sizey)*(myall-nghost)-1*0,istep) .eq. 0) goto 1220
---
> 1210  if(modulo(jstart+modulo(rank,sizey)*(myall-5)-1*0,istep) .eq. 0) goto 1220
4038c4035
< 1230  if(modulo(jfinish+modulo(rank,sizey)*(myall-nghost)-1*0,istep) .eq. 0)  goto 1240
---
> 1230  if(modulo(jfinish+modulo(rank,sizey)*(myall-5)-1*0,istep) .eq. 0)  goto 1240
```

10 tristanmainloop.F90

Definitely keep `outcorner` $= 1$, to avoid a numerical artifact. Without this, when partitioning the domain with `sizex` and `sizey`, particles in corner cells can be removed from the simulation. With only one call of `exchange_current` and `inject_particles`, corner particles still end up in a ghost cell; these particles are determined to have `in` $=$ `false`, and are removed from the simulation; some wave features can continue propagating in place of the removed particle. Keeping `outcorner` $= 1$ ensures that even the particles in corner cells eventually end up in physical cells.

10.1 mainloop()

Record output at the beginning of the loop:

```
108,110c103
<
<             call Diagnostics()
<
---
>
114,119c107,108
<                 call pre_bc_b()
<
```

The call to `bc_e1()` is present before `advance_Bhalfstep()` in case the high order field advance is used (the call to `bc_b1()` is redundant, and probably safe to remove). `bc_e1()` is also called so that the `mover_1ord`, `mover_2ord`, `mover_3ord` routines see the correct field values in ghost cells. Without this call, a particle in a ghost cell would 'see' a zero field, only because fields had not been copied from the other side of the domain into the ghost cells. This leads to large error in the field interpolation, for particles whose stencils overlap with ghost cells.

```
< !             print *, "advance b half" !hack
<                 call bc_b1()!USER
<                 call bc_e1()!USER
<               call advance_Bhalfstep()
---
>               ! Advance the Magnetic field half time step
> !             print *, "prebc" !hack
120a110,113
>               call pre_bc_b()
> !             print *, "advance b half" !hack
>               call advance_Bhalfstep()
>
```

`bc_b1()` is called after the field advance so that the `mover_1ord`, `mover_2ord`, `mover_3ord` routines see correct field values in ghost cells.

```
139,141c133,134
<               call advance_Bhalfstep()
<                 call bc_b1()!USER
<
---
>               call advance_Bhalfstep()
>
```

These calls the `bc_b1`, `bc_e1` are leftover from testing. Should be safe to remove.

```
180,183c177,178
<
<                 call bc_e1()
<                 call bc_b1()
<                 call deposit_particles()
---
>               call deposit_particles()
>
```

Ghost cell modifications are untested with `conduct_bc()`.

```
243c238,241
<                !call conduct_bc()
---
>
>                call conduct_bc()
>
>                call bc_e1()
245d242
<                !call bc_e1()
293c291
<                !call Diagnostics()
---
>                call Diagnostics()
```

11 user_weibel.F90

```
265,266c265,266
<        xinject=1.*(nghost/2+1)
<        xinject2=(mx0-1.*(nghost/2))
---
>        xinject=3.
>        xinject2=(mx0-2.)
279,280c279,280
<        y1=1.*(nghost/2+1) !in global coordinates
<        y2=my0-1.*(nghost/2)
---
>        y1=3. !in global coordinates
>        y2=my0-2.
301,302c301,302
<        y1=1.*(nghost/2+1) !in global coordinates
<        y2=my0-1.*(nghost/2)
---
>        y1=3. !in global coordinates
>        y2=my0-2.
```

12 Unmodified routines

- aux.F90
- communications.F90

  Use of `dynamic_domain` will require updates to work with the ghost cell modifications (but we are willing to do it).

- dynamic_domain.F90
- fparser.F90
- globaldata.F90
- initialize.F90
- inputparserf.F90
- mpidummy.F90
- overload.F90
- restart.F90
- tristan.F90
- selectprt.F90
- systemf.F90

References

[1] Esirkepov, T. Z., 2001, Computer Physics Communications, 185, 708.