```
/*--------------------------------------------------------------------*/
/* Skeleton 2-1/2D Electromagnetic OpenMP PIC code */
/* written by Viktor K. Decyk, UCLA */
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <sys/time.h>
#include "mbpush2.h"
#include "omplib.h"

void dtimer(double *time, struct timeval *itime, int icntrl);

int main(int argc, char *argv[]) {
   int indx =   9, indy =   9;
   int npx =  3072, npy =   3072;
   int ndim = 3;
   float tend = 10.0, dt = 0.04, qme = -1.0;
   float vtx = 1.0, vty = 1.0, vx0 = 0.0, vy0 = 0.0;
   float vtz = 1.0, vz0 = 0.0;
   float ax = .912871, ay = .912871, ci = 0.1;
/* idimp = dimension of phase space = 5 */
/* relativity = (no,yes) = (0,1) = relativity is used */
   int idimp = 5, ipbc = 1, relativity = 1;
   float wke = 0.0, we = 0.0, wf = 0.0, wm = 0.0, wt = 0.0;
/* sorting tiles, should be less than or equal to 32 */
   int mx = 16, my = 16;
/* fraction of extra particles needed for particle management */
   float xtras = 0.2;
/* declare scalars for standard code */
   int j;
   int np, nx, ny, nxh, nyh, nxe, nye, nxeh, nxyh, nxhy;
   int mx1, my1, mxy1, ntime, nloop, isign;
   float qbme, affp, dth;

/* declare scalars for OpenMP code */
   int nppmx, nppmx0, ntmax, npbmx, irc;
   int nvp;

/* declare arrays for standard code */
   float *part = NULL;
   float *qe = NULL, *cue = NULL, *fxyze = NULL, *bxyze = NULL;
   float complex *exyz = NULL, *bxyz = NULL;
   float complex *ffc = NULL, *sct = NULL;
   int *mixup = NULL;

/* declare arrays for OpenMP (tiled) code */
   float *ppart = NULL, *ppbuff = NULL;
   int *kpic = NULL, *ncl = NULL, *ihole = NULL;

/* declare and initialize timing data */
   float time;
   struct timeval itime;
   float tdpost = 0.0, tguard = 0.0, tfft = 0.0, tfield = 0.0;
   float tdjpost = 0.0, tpush = 0.0, tsort = 0.0;
```

```
   double dtime;

   irc = 0;
/* nvp = number of shared memory nodes  (0=default) */
   nvp = 0;
/* printf("enter number of nodes:\n"); */
/* scanf("%i",&nvp);                   */
/* initialize for shared memory parallel processing */
   cinit_omp(nvp);

/* initialize scalars for standard code */
   np = npx*npy; nx = 1L<<indx; ny = 1L<<indy; nxh = nx/2; nyh = ny/2;
   nxe = nx + 2; nye = ny + 1; nxeh = nxe/2;
   nxyh = (nx > ny ? nx : ny)/2; nxhy = nxh > ny ? nxh : ny;
   mx1 = (nx - 1)/mx + 1; my1 = (ny - 1)/my + 1; mxy1 = mx1*my1;
   nloop = tend/dt + .0001; ntime = 0;
   qbme = qme;
   affp = (float) (nx*ny)/(float ) np;
   dth = 0.0;

/* allocate and initialize data for standard code */
   part = (float *) malloc(idimp*np*sizeof(float));
   qe = (float *) malloc(nxe*nye*sizeof(float));
   fxyze = (float *) malloc(ndim*nxe*nye*sizeof(float));
   cue = (float *) malloc(ndim*nxe*nye*sizeof(float));
   bxyze = (float *) malloc(ndim*nxe*nye*sizeof(float));
   exyz = (float complex *) malloc(ndim*nxeh*nye*sizeof(float complex));
   bxyz = (float complex *) malloc(ndim*nxeh*nye*sizeof(float complex));
   ffc = (float complex *) malloc(nxh*nyh*sizeof(float complex));
   mixup = (int *) malloc(nxhy*sizeof(int));
   sct = (float complex *) malloc(nxyh*sizeof(float complex));
   kpic = (int *) malloc(mxy1*sizeof(int));

/* prepare fft tables */
   cwfft2rinit(mixup,sct,indx,indy,nxhy,nxyh);
/* calculate form factors */
   isign = 0;
   cmpois23((float complex *)qe,(float complex *)fxyze,isign,ffc,ax,ay,
            affp,&we,nx,ny,nxeh,nye,nxh,nyh);
/* initialize electrons */
   cdistr2h(part,vtx,vty,vtz,vx0,vy0,vz0,npx,npy,idimp,np,nx,ny,ipbc);

/* initialize transverse electromagnetic fields */
   for (j = 0; j < ndim*nxeh*nye; j++) {
      exyz[j] = 0.0 + 0.0*_Complex_I;
      bxyz[j] = 0.0 + 0.0*_Complex_I;
   }

/* find number of particles in each of mx, my tiles: updates kpic, nppmx */
   cdblkp2l(part,kpic,&nppmx,idimp,np,mx,my,mx1,mxy1,&irc);
   if (irc != 0) {
      printf("cdblkp2l error, irc=%d\n",irc);
      exit(1);
   }
```

```
/* allocate vector particle data */
   nppmx0 = (1.0 + xtras)*nppmx;
   ntmax = xtras*nppmx;
   npbmx = xtras*nppmx;
   ppart = (float *) malloc(idimp*nppmx0*mxy1*sizeof(float));
   ppbuff = (float *) malloc(idimp*npbmx*mxy1*sizeof(float));
   ncl = (int *) malloc(8*mxy1*sizeof(int));
   ihole = (int *) malloc(2*(ntmax+1)*mxy1*sizeof(int));
/* copy ordered particle data for OpenMP */
   cppmovin2l(part,ppart,kpic,nppmx0,idimp,np,mx,my,mx1,mxy1,&irc);
   if (irc != 0) {
      printf("cppmovin2l overflow error, irc=%d\n",irc);
      exit(1);
   }
/* sanity check */
   cppcheck2l(ppart,kpic,idimp,nppmx0,nx,ny,mx,my,mx1,my1,&irc);
   if (irc != 0) {
      printf("%d,cppcheck2l error: irc=%d\n",ntime,irc);
      exit(1);
   }

   if (dt > 0.45*ci) {
      printf("Warning: Courant condition may be exceeded!\n");
   }

/* * * * start main iteration loop * * * */

L500: if (nloop <= ntime)
         goto L2000;
/*    printf("ntime = %i\n",ntime); */

/* deposit current with OpenMP: updates ppart, cue, ncl, ihole, irc */
      dtimer(&dtime,&itime,-1);
      for (j = 0; j < ndim*nxe*nye; j++) {
         cue[j] = 0.0;
      }
      if (relativity==1) {
         cgrjppostf2l(ppart,cue,kpic,ncl,ihole,qme,dth,ci,nppmx0,
                      idimp,nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax,&irc);
      }
      else {
         cgjppostf2l(ppart,cue,kpic,ncl,ihole,qme,dth,nppmx0,idimp,
                     nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax,&irc);
      }
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tdjpost += time;
      if (irc != 0) {
         if (relativity==1) {
            printf("cgrjppostf2l error: irc=%d\n",irc);
         }
         else {
            printf("cgjppostf2l error: irc=%d\n",irc);
         }
```

```
         exit(1);
      }

/* reorder particles by cell with OpenMP: */
/* updates ppart, ppbuff, kpic, ncl, ihole, and irc */
      dtimer(&dtime,&itime,-1);
      cpporderf2l(ppart,ppbuff,kpic,ncl,ihole,idimp,nppmx0,mx1,my1,
                  npbmx,ntmax,&irc);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tsort += time;
      if (irc != 0) {
         printf("current cpporderf2l error: ntmax, irc=%d,%d\n",ntmax,
                irc);
         exit(1);
      }

/* deposit charge with OpenMP: updates qe */
      dtimer(&dtime,&itime,-1);
      for (j = 0; j < nxe*nye; j++) {
         qe[j] = 0.0;
      }
      cgppost2l(ppart,qe,kpic,qme,nppmx0,idimp,mx,my,nxe,nye,mx1,mxy1);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tdpost += time;

/* add guard cells with OpenMP: updates cue, qe */
      dtimer(&dtime,&itime,-1);
      cacguard2l(cue,nx,ny,nxe,nye);
      caguard2l(qe,nx,ny,nxe,nye);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* transform charge to fourier space with OpenMP: updates qe */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cwfft2rmx((float complex *)qe,isign,mixup,sct,indx,indy,nxeh,nye,
                nxhy,nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* transform current to fourier space with OpenMP: updates cue */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cwfft2rm3((float complex *)cue,isign,mixup,sct,indx,indy,nxeh,nye,
                nxhy,nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* take transverse part of current with OpenMP: updates cue */
```

```
      dtimer(&dtime,&itime,-1);
      cmcuperp2((float complex *)cue,nx,ny,nxeh,nye);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* calculate electromagnetic fields in fourier space with OpenMP: */
/* updates exyz, bxyz */
      dtimer(&dtime,&itime,-1);
      if (ntime==0) {
         cmibpois23((float complex *)cue,bxyz,ffc,ci,&wm,nx,ny,nxeh,nye,
                    nxh,nyh);
         wf = 0.0;
         dth = 0.5*dt;
      }
      else {
         cmmaxwel2(exyz,bxyz,(float complex *)cue,ffc,ci,dt,&wf,&wm,nx,
                   ny,nxeh,nye,nxh,nyh);
      }
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* calculate force/charge in fourier space OpenMP: updates fxyze */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cmpois23((float complex *)qe,(float complex *)fxyze,isign,ffc,ax,
               ay,affp,&we,nx,ny,nxeh,nye,nxh,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* add longitudinal and transverse electric fields with OpenMP: */
/* updates fxyze */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cmemfield2((float complex *)fxyze,exyz,ffc,isign,nx,ny,nxeh,nye,
                 nxh,nyh);
/* copy magnetic field with OpenMP: updates bxyze */
      isign = -1;
      cmemfield2((float complex *)bxyze,bxyz,ffc,isign,nx,ny,nxeh,nye,
                 nxh,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* transform electric force to real space with OpenMP: updates fxyze */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cwfft2rm3((float complex *)fxyze,isign,mixup,sct,indx,indy,nxeh,
                nye,nxhy,nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;
```

```
/* transform magnetic force to real space with OpenMP: updates bxyze */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cwfft2rm3((float complex *)bxyze,isign,mixup,sct,indx,indy,nxeh,
                nye,nxhy,nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* copy guard cells with OpenMP: updates fxyze, bxyze */
      dtimer(&dtime,&itime,-1);
      cbguard2l(fxyze,nx,ny,nxe,nye);
      cbguard2l(bxyze,nx,ny,nxe,nye);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* push particles with OpenMP: updates ppart, ncl, ihole, wke, irc */
      wke = 0.0;
      dtimer(&dtime,&itime,-1);
      if (relativity==1) {
         cgrbppushf23l(ppart,fxyze,bxyze,kpic,ncl,ihole,qbme,dt,dth,ci,
                       &wke,idimp,nppmx0,nx,ny,mx,my,nxe,nye,mx1,mxy1,
                       ntmax,&irc);
      }
      else {
         cgbppushf23l(ppart,fxyze,bxyze,kpic,ncl,ihole,qbme,dt,dth,&wke,
                      idimp,nppmx0,nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax,
                      &irc);
      }
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tpush += time;
      if (irc != 0) {
         if (relativity==1) {
            printf("cgrbppushf23l error: irc=%d\n",irc);
         }
         else {
            printf("cgbppushf23l error: irc=%d\n",irc);
         }
         exit(1);
      }

/* reorder particles by cell with OpenMP: */
/* updates ppart, ppbuff, kpic, ncl, ihole, and irc */
      dtimer(&dtime,&itime,-1);
      cpporderf2l(ppart,ppbuff,kpic,ncl,ihole,idimp,nppmx0,mx1,my1,
                  npbmx,ntmax,&irc);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tsort += time;
      if (irc != 0) {
         printf("push cpporderf2l error: ntmax, irc=%d,%d\n",ntmax,irc);
```

```
         exit(1);
      }

      if (ntime==0) {
         wt = we + wf + wm;
         printf("Initial Total Field, Kinetic and Total Energies:\n");
         printf("%e %e %e\n",wt,wke,wke+wt);
         printf("Initial Electrostatic, Transverse Electric and Magnetic \
Field Energies:\n");
         printf("%e %e %e\n",we,wf,wm);
      }
      ntime += 1;
      goto L500;
L2000:

/* * * * end main iteration loop * * * */

   printf("ntime, relativity = %i,%i\n",ntime,relativity);
   wt = we + wf + wm;
   printf("Final Total Field, Kinetic and Total Energies:\n");
   printf("%e %e %e\n",wt,wke,wke+wt);
   printf("Final Electrostatic, Transverse Electric and Magnetic Field \
Energies:\n");
   printf("%e %e %e\n",we,wf,wm);

   printf("\n");
   printf("deposit time = %f\n",tdpost);
   printf("current deposit time = %f\n",tdjpost);
   tdpost += tdjpost;
   printf("total deposit time = %f\n",tdpost);
   printf("guard time = %f\n",tguard);
   printf("solver time = %f\n",tfield);
   printf("fft time = %f\n",tfft);
   printf("push time = %f\n",tpush);
   printf("sort time = %f\n",tsort);
   tfield += tguard + tfft;
   printf("total solver time = %f\n",tfield);
   time = tdpost + tpush + tsort;
   printf("total particle time = %f\n",time);
   wt = time + tfield;
   printf("total time = %f\n",wt);
   printf("\n");

   wt = 1.0e+09/(((float) nloop)*((float) np));
   printf("Push Time (nsec) = %f\n",tpush*wt);
   printf("Deposit Time (nsec) = %f\n",tdpost*wt);
   printf("Sort Time (nsec) = %f\n",tsort*wt);
   printf("Total Particle Time (nsec) = %f\n",time*wt);

   return 0;
}
```