```
/*-----------------------------------------------------------------------*/
/* Skeleton 2D Electrostatic MPI PIC code */
/* written by Viktor K. Decyk, UCLA */
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <sys/time.h>
#include "ppush2.h"
#include "pplib2.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 = 2;
   float tend = 10.0, dt = 0.1, qme = -1.0;
   float vtx = 1.0, vty = 1.0, vx0 = 0.0, vy0 = 0.0;
   float ax = .912871, ay = .912871;
/* idimp = dimension of phase space = 4 */
/* sortime = number of time steps between standard electron sorting */
   int idimp = 4, ipbc = 1, sortime = 50;
/* idps = number of partition boundaries */
   int idps = 2;
   float wke = 0.0, we = 0.0, wt = 0.0;
/* declare scalars for standard code */
   int j;
   int nx, ny, nxh, nyh, nxe, nye, nxeh, nnxe, nxyh, nxhy;
   int ny1, ntime, nloop, isign, ierr;
   float qbme, affp;
   double np;

/* declare scalars for MPI code */
   int ntpose = 1;
   int nvp, idproc, kstrt, npmax, kxp, kyp, nypmx, nypmn;
   int nyp, noff, npp, nps, nbmax, ntmax;

/* declare arrays for standard code */
   float *part = NULL, *part2 = NULL, *tpart = NULL;
   float *qe = NULL;
   float *fxye = NULL;
   float complex *qt = NULL;
   float complex *fxyt = NULL;
   float complex *ffc = NULL;
   int *mixup = NULL;
   float complex *sct = NULL;
   int *ihole = NULL;
   int *npic = NULL;
   float wtot[4], work[4];
   int info[7];

/* declare arrays for MPI code */
   float complex *bs = NULL, *br = NULL;
   float *sbufl = NULL, *sbufr = NULL, *rbufl = NULL, *rbufr = NULL;
   float *edges = NULL;
   float *scr = NULL;

/* declare and initialize timing data */
   float time;
   struct timeval itime;
   float tdpost = 0.0, tguard = 0.0, ttp = 0.0, tfield = 0.0;
   float tpush = 0.0, tsort = 0.0, tmov = 0.0;
   float tfft[2] = {0.0,0.0};
   double dtime;

/* initialize scalars for standard code */
   np =  (double) npx*(double) npy;
   nx = 1L<<indx; ny = 1L<<indy; nxh = nx/2; nyh = ny/2;
   nxe = nx + 2; nye = ny + 2; nxeh = nxe/2; nnxe = ndim*nxe;
   nxyh = (nx > ny ? nx : ny)/2; nxhy = nxh > ny ? nxh : ny;
   ny1 = ny + 1;
   nloop = tend/dt + .0001; ntime = 0;
   qbme = qme;
   affp = (double) nx*(double) ny/np;

/* nvp = number of distributed memory nodes */
/* initialize for distributed memory parallel processing */
   cppinit2(&idproc,&nvp,argc,argv);
   kstrt = idproc + 1;
/* check if too many processors */
   if (nvp > ny) {
      if (kstrt==1) {
         printf("Too many processors requested: ny, nvp=%d,%d\n",ny,nvp);
      }
      goto L3000;
   }

/* initialize data for MPI code */
   edges = (float *) malloc(idps*sizeof(float));
/* calculate partition variables: edges, nyp, noff, nypmx            */
/* edges[0:1] = lower:upper boundary of particle partition           */
/* nyp = number of primary (complete) gridpoints in particle partition */
/* noff = lowermost global gridpoint in particle partition           */
/* nypmx = maximum size of particle partition, including guard cells   */
/* nypmn = minimum value of nyp                                      */
   cpdicomp2l(edges,&nyp,&noff,&nypmx,&nypmn,ny,kstrt,nvp,idps);
   if (nypmn < 1) {
      if (kstrt==1) {
         printf("combination not supported nvp, ny =%d,%d\n",ny,nvp);
      }
      goto L3000;
   }

/* initialize additional scalars for MPI code */
/* kxp = number of complex grids in each field partition in x direction */
   kxp = (nxh - 1)/nvp + 1;
/* kyp = number of complex grids in each field partition in y direction */
   kyp = (ny - 1)/nvp + 1;
```

```
/* npmax = maximum number of electrons in each partition */
   npmax = (np/nvp)*1.25;
/* nbmax = size of buffer for passing particles between processors */
   nbmax = 0.1*npmax;
/* ntmax = size of ihole buffer for particles leaving processor */
   ntmax = 2*nbmax;

/* allocate and initialize data for standard code */
   part = (float *) malloc(idimp*npmax*sizeof(float));
   part2 = (float *) malloc(idimp*npmax*sizeof(float));
   qe = (float *) malloc(nxe*nypmx*sizeof(float));
   fxye = (float *) malloc(ndim*nxe*nypmx*sizeof(float));
   qt = (float complex *) malloc(nye*kxp*sizeof(float complex));
   fxyt = (float complex *) malloc(ndim*nye*kxp*sizeof(float complex));
   ffc = (float complex *) malloc(nyh*kxp*sizeof(float complex));
   mixup = (int *) malloc(nxhy*sizeof(int));
   sct = (float complex *) malloc(nxyh*sizeof(float complex));
   ihole = (int *) malloc((ntmax+1)*sizeof(int));
   npic = (int *) malloc(nypmx*sizeof(int));

/* allocate and initialize data for MPI code */
   bs = (float complex *) malloc(ndim*kxp*kyp*sizeof(float complex));
   br = (float complex *) malloc(ndim*kxp*kyp*sizeof(float complex));
   sbufl = (float *) malloc(idimp*nbmax*sizeof(float));
   sbufr = (float *) malloc(idimp*nbmax*sizeof(float));
   rbufl = (float *) malloc(idimp*nbmax*sizeof(float));
   rbufr = (float *) malloc(idimp*nbmax*sizeof(float));
   scr = (float *) malloc(nxe*2*sizeof(float));

/* prepare fft tables */
   cwpfft2rinit(mixup,sct,indx,indy,nxhy,nxyh);
/* calculate form factors */
   isign = 0;
   cppois22(qt,fxyt,isign,ffc,ax,ay,affp,&we,nx,ny,kstrt,nye,kxp,nyh);
/* initialize electrons */
   nps = 1;
   npp = 0;
   cpdistr2(part,edges,&npp,nps,vtx,vty,vx0,vy0,npx,npy,nx,ny,idimp,
            npmax,idps,ipbc,&ierr);
/* check for particle initialization error */
   if (ierr != 0) {
      if (kstrt==1) {
         printf("particle initialization error: ierr=%d\n",ierr);
      }
      goto L3000;
   }

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

L500: if (nloop <= ntime)
         goto L2000;
/*    if (kstrt==1) printf("ntime = %i\n",ntime); */

/* deposit charge with standard procedure: updates qe */
```

```
      dtimer(&dtime,&itime,-1);
      for (j = 0; j < nxe*nypmx; j++) {
         qe[j] = 0.0;
      }
      cppgpost2l(part,qe,npp,noff,qme,idimp,npmax,nxe,nypmx);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tdpost += time;

/* add guard cells with standard procedure: updates qe */
      dtimer(&dtime,&itime,-1);
      cppaguard2xl(qe,nyp,nx,nxe,nypmx);
      cppnaguard2l(qe,scr,nyp,nx,kstrt,nvp,nxe,nypmx);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* transform charge to fourier space with standard procedure: updates qt */
/* modifies qe */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cwppfft2r((float complex *)qe,qt,bs,br,isign,ntpose,mixup,sct,&ttp,
                indx,indy,kstrt,nvp,nxeh,nye,kxp,kyp,nypmx,nxhy,nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft[0] += time;
      tfft[1] += ttp;

/* calculate force/charge in fourier space with standard procedure: */
/* updates fxyt */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cppois22(qt,fxyt,isign,ffc,ax,ay,affp,&we,nx,ny,kstrt,nye,kxp,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* transform force to real space with standard procedure: updates fxye */
/* modifies fxyt */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cwppfft2r2((float complex *)fxye,fxyt,bs,br,isign,ntpose,mixup,sct,
                 &ttp,indx,indy,kstrt,nvp,nxeh,nye,kxp,kyp,nypmx,nxhy,
                 nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft[0] += time;
      tfft[1] += ttp;

/* copy guard cells with standard procedure: updates fxye */
      dtimer(&dtime,&itime,-1);
      cppncguard2l(fxye,nyp,kstrt,nvp,nnxe,nypmx);
      cppcguard2xl(fxye,nyp,nx,ndim,nxe,nypmx);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* push particles: updates part, wke, and ihole */
      dtimer(&dtime,&itime,-1);
      wke = 0.0;
      cppgpush2l(part,fxye,edges,npp,noff,ihole,qbme,dt,&wke,nx,ny,idimp,
                 npmax,nxe,nypmx,idps,ntmax,ipbc);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tpush += time;
/* check for ihole overflow error */
      if (ihole[0] < 0) {
         ierr = -ihole[0];
         printf("ihole overflow error: ntmax,ih=%d,%d\n",ntmax,ierr);
         cppabort();
         goto L3000;
      }
/* move electrons into appropriate spatial regions: updates part, npp */
      dtimer(&dtime,&itime,-1);
      cppmove2(part,edges,&npp,sbufr,sbufl,rbufr,rbufl,ihole,ny,kstrt,nvp,
               idimp,npmax,idps,nbmax,ntmax,info);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tmov += time;
/* check for particle manager error */
      if (info[0] != 0) {
         ierr = info[0];
         if (kstrt==1) {
            printf("particle manager error: ierr=%d\n",ierr);
         }
         goto L3000;
      }

/* sort particles for standard code: updates part */
      if (sortime > 0) {
         if (ntime%sortime==0) {
            dtimer(&dtime,&itime,-1);
            cppdsortp2yl(part,part2,npic,npp,noff,nyp,idimp,npmax,nypmx);
/* exchange pointers */
            tpart = part;
            part = part2;
            part2 = tpart;
            dtimer(&dtime,&itime,1);
            time = (float) dtime;
            tsort += time;
         }
      }

/* energy diagnostic */
      wtot[0] = we;
      wtot[1] = wke;
      wtot[2] = 0.0;
      wtot[3] = we + wke;
      cppsum(wtot,work,4);
      we = wtot[0];
      wke = wtot[1];
      if (ntime==0) {
         if (kstrt==1) {
            printf("Initial Field, Kinetic and Total Energies:\n");
            printf("%e %e %e\n",we,wke,wke+we);
         }
      }
      ntime += 1;
      goto L500;
L2000:

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

   if (kstrt==1) {
      printf("ntime = %i\n",ntime);
      printf("MPI nodes nvp = %i\n",nvp);
      printf("Final Field, Kinetic and Total Energies:\n");
      printf("%e %e %e\n",we,wke,wke+we);

      printf("\n");
      printf("deposit time = %f\n",tdpost);
      printf("guard time = %f\n",tguard);
      printf("solver time = %f\n",tfield);
      printf("fft and transpose time = %f,%f\n",tfft[0],tfft[1]);
      printf("push time = %f\n",tpush);
      printf("particle move time = %f\n",tmov);
      printf("sort time = %f\n",tsort);
      tfield += tguard + tfft[0];
      printf("total solver time = %f\n",tfield);
      time = tdpost + tpush + tmov + 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);
   }

L3000:
   cppexit();
   return 0;
}
```