```
/*-----------------------------------------------------------------------*/
/* Skeleton 2D Electrostatic OpenMP/Vector PIC code */
/* written by Viktor K. Decyk, UCLA and Ricardo Fonseca, ISCTE */
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <sys/time.h>
#include "vmpush2.h"
#include "omplib.h"
#include "sselib2.h"
#include "ssempush2.h"

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

int main(int argc, char *argv[]) {
/* indx/indy = exponent which determines grid points in x/y direction: */
/* nx = 2**indx, ny = 2**indy */
   int indx =   9, indy =   9;
/* npx/npy = number of electrons distributed in x/y direction */
   int npx =  3072, npy =   3072;
/* ndim = number of velocity coordinates = 2 */
   int ndim = 2;
/* tend = time at end of simulation, in units of plasma frequency */
/* dt = time interval between successive calculations */
/* qme = charge on electron, in units of e */
   float tend = 10.0, dt = 0.1, qme = -1.0;
/* vtx/vty = thermal velocity of electrons in x/y direction */
/* vx0/vy0 = drift velocity of electrons in x/y direction */
   float vtx = 1.0, vty = 1.0, vx0 = 0.0, vy0 = 0.0;
/* ax/ay = smoothed particle size in x/y direction */
   float ax = .912871, ay = .912871;
/* idimp = number of particle coordinates = 4 */
/* ipbc = particle boundary condition: 1 = periodic */
   int idimp = 4, ipbc = 1;
/* wke/we/wt = particle kinetic/electric field/total energy */
   float wke = 0.0, we = 0.0, wt = 0.0;
/* mx/my = number of grids in x/y in sorting tiles */
   int mx = 16, my = 16;
/* xtras = fraction of extra particles needed for particle management */
   float xtras = 0.2;
/* kvec = (1,2) = run (autovector,SSE2) version */
   int kvec = 1;

/* 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, lvect;
   int irc = 0;
   float qbme, affp;

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

```
/* declare arrays for standard code: */
/* part = original particle array */
   float *part = NULL;
/* qe = electron charge density with guard cells */
   float *qe = NULL;
/* fxye = smoothed electric field with guard cells */
   float *fxye = NULL;
/* ffc = form factor array for poisson solver */
   float complex *ffc = NULL;
/* mixup = bit reverse table for FFT */
   int *mixup = NULL;
/* sct = sine/cosine table for FFT */
   float complex *sct = NULL;

/* declare arrays for OpenMP (tiled) code: */
/* ppartt = tiled particle array */
/* ppbuff = buffer array for reordering tiled particle array */
   float *ppartt = NULL, *ppbuff = NULL;
/* kpic = number of particles in each tile */
   int *kpic = NULL;
/* ncl = number of particles departing tile in each direction */
   int *ncl = NULL;
/* ihole = location/destination of each particle departing tile */
   int *ihole = NULL;
/* kp = original location of reordered particle */
   int *kp = 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 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 = total number of particles in simulation */
/* nx/ny = number of grid points in x/y direction */
   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/my1 = number of tiles in x/y direction */
   mx1 = (nx - 1)/mx + 1; my1 = (ny - 1)/my + 1; mxy1 = mx1*my1;
/* nloop = number of time steps in simulation */
/* ntime = current time step */
   nloop = tend/dt + .0001; ntime = 0;
   qbme = qme;
```

```
   affp = (float) (nx*ny)/(float ) np;

/* allocate data for standard code */
   part = (float *) malloc(idimp*np*sizeof(float));
   mixup = (int *) malloc(nxhy*sizeof(int));
   sct = (float complex *) malloc(nxyh*sizeof(float complex));
   kpic = (int *) malloc(mxy1*sizeof(int));

   lvect = 4;
/* allocate vector field data */
   nxe = lvect*((nxe - 1)/lvect + 1);
   nxeh = nxe/2;
   sse_fallocate(&qe,nxe*nye,&irc);
   sse_fallocate(&fxye,ndim*nxe*nye,&irc);
   sse_callocate(&ffc,nxh*nyh,&irc);
   if (irc != 0) {
      printf("aligned field allocation error: irc = %d\n",irc);
   }

/* prepare fft tables */
   cwfft2rinit(mixup,sct,indx,indy,nxhy,nxyh);
/* calculate form factors */
   isign = 0;
   cvmpois22((float complex *)qe,(float complex *)fxye,isign,ffc,ax,ay,
             affp,&we,nx,ny,nxeh,nye,nxh,nyh);
/* initialize electrons */
   cdistr2(part,vtx,vty,vx0,vy0,npx,npy,idimp,np,nx,ny,ipbc);

/* 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;
/* align data for Vector Processor */
   nppmx0 = lvect*((nppmx0 - 1)/lvect + 1);
   ntmax = lvect*(ntmax/lvect + 1);
   npbmx = lvect*((npbmx - 1)/lvect + 1);
   sse_fallocate(&ppartt,nppmx0*idimp*mxy1,&irc);
   sse_fallocate(&ppbuff,npbmx*idimp*mxy1,&irc);
   ncl = (int *) malloc(8*mxy1*sizeof(int));
   ihole = (int *) malloc(2*(ntmax+1)*mxy1*sizeof(int));
   kp = (int *) malloc(nppmx0*mxy1*sizeof(int));
   if (irc != 0) {
      printf("aligned particle allocation error: irc = %d\n",irc);
   }

/* copy ordered particle data for OpenMP: updates ppartt, kpic, and kp */
   cppmovin2ltp(part,ppartt,kpic,kp,nppmx0,idimp,np,mx,my,mx1,mxy1,
                &irc);
```

```
   if (irc != 0) {
      printf("cppmovin2ltp overflow error, irc=%d\n",irc);
      exit(1);
   }
/* sanity check */
   cppcheck2lt(ppartt,kpic,idimp,nppmx0,nx,ny,mx,my,mx1,my1,&irc);
   if (irc != 0) {
      printf("cppcheck2lt error: irc=%d\n",irc);
      exit(1);
   }

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

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

/* deposit charge with OpenMP: updates qe */
      dtimer(&dtime,&itime,-1);
      for (j = 0; j < nxe*nye; j++) {
         qe[j] = 0.0;
      }
      if (kvec==1)
         cvgppost2lt(ppartt,qe,kpic,qme,nppmx0,idimp,mx,my,nxe,nye,mx1,
                     mxy1);
/* SSE2 function */
      else if (kvec==2)
         csse2gppost2lt(ppartt,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 qe */
      dtimer(&dtime,&itime,-1);
      if (kvec==1)
         caguard2l(qe,nx,ny,nxe,nye);
/* SSE2 function */
      else if (kvec==2)
         csse2aguard2l(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;
      if (kvec==1)
         cwfft2rvmx((float complex *)qe,isign,mixup,sct,indx,indy,nxeh,
                    nye,nxhy,nxyh);
/* SSE2 function */
      else if (kvec==2)
         csse2wfft2rmx((float complex *)qe,isign,mixup,sct,indx,indy,
                       nxeh,nye,nxhy,nxyh);
```

```
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* calculate force/charge in fourier space with OpenMP: updates fxye, we */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      if (kvec==1)
         cvmpois22((float complex *)qe,(float complex *)fxye,isign,ffc,
                   ax,ay,affp,&we,nx,ny,nxeh,nye,nxh,nyh);
/* SSE2 function */
      else if (kvec==2)
         csse2mpois22((float complex *)qe,(float complex *)fxye,isign,
                      ffc,ax,ay,affp,&we,nx,ny,nxeh,nye,nxh,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* transform force to real space with OpenMP: updates fxye */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      if (kvec==1)
         cwfft2rvm2((float complex *)fxye,isign,mixup,sct,indx,indy,nxeh,
                    nye,nxhy,nxyh);
/* SSE2 function */
      else if (kvec==2)
         csse2wfft2rm2((float complex *)fxye,isign,mixup,sct,indx,indy,
                       nxeh,nye,nxhy,nxyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* copy guard cells with OpenMP: updates fxye */
      dtimer(&dtime,&itime,-1);
      if (kvec==1)
         ccguard2l(fxye,nx,ny,nxe,nye);
/* SSE2 function */
      else if (kvec==2)
         csse2cguard2l(fxye,nx,ny,nxe,nye);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* push particles with OpenMP: */
      wke = 0.0;
      dtimer(&dtime,&itime,-1);
/* updates ppartt, wke */
/*    if (kvec==1)                                                    */
/*       cvgppush2lt(ppartt,fxye,kpic,qbme,dt,&wke,idimp,nppmx0,nx,ny, */
/*                   mx,my,nxe,nye,mx1,mxy1,ipbc);                   */
/* SSE2 function */
/*    else if (kvec==2)                                               */
/*       csse2gppush2lt(ppartt,fxye,kpic,qbme,dt,&wke,idimp,nppmx0,nx, */
/*                      ny,mx,my,nxe,nye,mx1,mxy1,ipbc);             */
```

```
/* updates ppartt, ncl, ihole, wke, irc */
      if (kvec==1)
         cvgppushf2lt(ppartt,fxye,kpic,ncl,ihole,qbme,dt,&wke,idimp,
                      nppmx0,nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax,&irc);
/* SSE2 function */
      else if (kvec==2)
         csse2gppushf2lt(ppartt,fxye,kpic,ncl,ihole,qbme,dt,&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) {
         printf("cvgppushf2lt error: irc=%d\n",irc);
         exit(1);
      }

/* reorder particles by tile with OpenMP: */
      dtimer(&dtime,&itime,-1);
/* updates ppartt, ppbuff, kpic, ncl, ihole, and irc */
/*    if (kvec==1)                                                    */
/*       cvpporder2lt(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,nx,ny, */
/*                    mx,my,mx1,my1,npbmx,ntmax,&irc);                 */
/* SSE2 function */
/*    else if (kvec==2)                                               */
/*       csse2pporder2lt(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,nx, */
/*                       ny,mx,my,mx1,my1,npbmx,ntmax,&irc);          */
/* updates ppartt, ppbuff, kpic, ncl, and irc */
      if (kvec==1)
         cvpporderf2lt(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,mx1,
                       my1,npbmx,ntmax,&irc);
/* SSE2 function */
      else if (kvec==2)
         csse2pporderf2lt(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,
                          mx1,my1,npbmx,ntmax,&irc);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tsort += time;
      if (irc != 0) {
         printf("cvpporderf2lt error: ntmax, irc=%d,%d\n",ntmax,irc);
         exit(1);
      }

      if (ntime==0) {
         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 * * * */

   printf("ntime = %i, kvec = %i\n",ntime,kvec);
```

```
   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 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);
   printf("\n");

   sse_deallocate(ppartt);
   sse_deallocate(ppbuff);
   sse_deallocate(ffc);
   sse_deallocate(fxye);
   sse_deallocate(qe);

   return 0;
}
```