```
/*----------------------------------------------------------------------*/
/* Skeleton 2-1/2D Electromagnetic GPU PIC code */
/* written by Viktor K. Decyk, UCLA */
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <sys/time.h>
#include "gpubpush2.h"
#include "gpulib2.h"
#include "gpufft2.h"
#include "bpush2.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 */
   int mx = 16, my = 16;
/* fraction of extra particles needed for particle management */
   float xtras = 0.2;
/* declare scalars for standard code */
   int np, nx, ny, nxh, nyh, nxh1, nxe, nye, nxeh, nxyh, nxhy;
   int mx1, my1, mxy1, ntime, nloop, isign;
   float qbme, affp, dth;

/* declare scalars for GPU code */
   int nblock = 128;
/* nscache = (0,1,2) = (no,small,big) cache size */
   int nscache = 1;
   int mmcc, nppmx, nppmx0, ntmax, npbmx, irc;
   int nxhd;

/* declare arrays for standard code */
   float *part = NULL;
   float complex *ffct = NULL, *sct = NULL;
   int *mixup = NULL;

/* declare arrays for GPU code */
   float *g_qe = NULL;
   float *g_cue = NULL, *g_fxyze = NULL, *g_bxyze = NULL;
   float complex *g_ffct = NULL;
   int *g_mixup = NULL;
   float complex *g_sct = NULL;
   float complex *g_q = NULL, *g_cu = NULL;
   float complex *g_qt = NULL, *g_cut = NULL;
   float complex *g_fxyz = NULL, *g_hxyz = NULL;
   float complex *g_fxyzt = NULL, *g_hxyzt = NULL;
   float complex *g_exyzt = NULL, *g_bxyzt = NULL;
   float *g_wke = NULL, *g_we = NULL;
   float *g_wf = NULL, *g_wm = NULL;
   float *g_ppart = NULL, *g_ppbuff = NULL;
   int *g_kpic = NULL;
   int *g_ncl = NULL, *g_ihole = NULL;
   float *g_sum = NULL;
   int *g_irc = NULL;
   float complex *qt = NULL;
   float complex *fxyzt = NULL;
   float *ppart = NULL;
   int *kpic = NULL;

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

/* initialize scalars for standard code */
   np = npx*npy; nx = 1L<<indx; ny = 1L<<indy; nxh = nx/2; nyh = ny/2;
   nxh1 = nxh + 1; 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;
/* set size for FFT arrays */
   nxhd = nxh1;

/* allocate and initialize data for standard code */
   part = (float *) malloc(idimp*np*sizeof(float));
   ffct = (float complex *) malloc(nyh*nxh*sizeof(float complex));
   mixup = (int *) malloc(nxhy*sizeof(int));
   sct = (float complex *) malloc(nxyh*sizeof(float complex));
   kpic = (int *) malloc(mxy1*sizeof(int));
   qt = (float complex *) malloc(ny*nxh1*sizeof(float complex));
   fxyzt = (float complex *) malloc(ny*ndim*nxh1*sizeof(float complex));

/* set up GPU */
   irc = 0;
   gpu_setgbsize(nblock);
   init_cu(0,&irc);
   if (irc != 0) {
      printf("CUDA initialization error!\n");
      exit(1);
   }
/* obtain compute capability */
   mmcc = getmmcc();
   if (mmcc < 20) {
      printf("compute capability 2.x or higher required\n");
      exit(1);
   }
/* set cache size */
   gpu_set_cache_size(nscache);

/* allocate and initialize data for GPU code */
   gpu_fallocate(&g_qe,nxe*nye,&irc);
   gpu_fallocate(&g_cue,ndim*nxe*nye,&irc);
   gpu_fallocate(&g_fxyze,ndim*nxe*nye,&irc);
   gpu_fallocate(&g_bxyze,ndim*nxe*nye,&irc);
   gpu_callocate(&g_ffct,nyh*nxh,&irc);
   gpu_iallocate(&g_mixup,nxhy,&irc);
   gpu_callocate(&g_sct,nxyh,&irc);
   gpu_callocate(&g_q,nxhd*ny,&irc);
   gpu_callocate(&g_cu,nxhd*ndim*ny,&irc);
   gpu_callocate(&g_qt,ny*nxh1,&irc);
   gpu_callocate(&g_cut,ny*ndim*nxh1,&irc);
   gpu_callocate(&g_fxyz,nxhd*ndim*ny,&irc);
   gpu_callocate(&g_hxyz,nxhd*ndim*ny,&irc);
   gpu_callocate(&g_fxyzt,ny*ndim*nxh1,&irc);
   gpu_callocate(&g_hxyzt,ny*ndim*nxh1,&irc);
   gpu_callocate(&g_exyzt,ny*ndim*nxh1,&irc);
   gpu_callocate(&g_bxyzt,ny*ndim*nxh1,&irc);
   gpu_fallocate(&g_wke,mxy1,&irc);
   gpu_fallocate(&g_we,nxh1,&irc);
   gpu_fallocate(&g_wf,nxh1,&irc);
   gpu_fallocate(&g_wm,nxh1,&irc);
   gpu_fallocate(&g_sum,1,&irc);
   if (irc != 0) {
      printf("GPU allocate error!\n");
      exit(1);
   }

/* prepare fft tables */
   cwfft2rinit(mixup,sct,indx,indy,nxhy,nxyh);
/* prepare NVIDIA ffts */
   gpufft2rrcuinit(nx,ny,ndim);
   gpufft2cuinit(nx,ny,ndim);
/* calculate form factors */
   isign = 0;
   cpois23t(qt,fxyzt,isign,ffct,ax,ay,affp,&we,nx,ny,nxh1,ny,nxh,nyh);
/* copy in solver arrays to GPU */
   gpu_icopyin(mixup,g_mixup,nxhy);
   gpu_ccopyin(sct,g_sct,nxyh);
   gpu_ccopyin(ffct,g_ffct,nyh*nxh);
/* initialize electrons */
   cdistr2h(part,vtx,vty,vtz,vx0,vy0,vz0,npx,npy,idimp,np,nx,ny,ipbc);

/* initialize transverse electromagnetic fields */
   gpu_zcmem(g_exyzt,ny*ndim*nxh1);
   gpu_zcmem(g_bxyzt,ny*ndim*nxh1);
```

```
/* 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 = 0.5*xtras*nppmx;
   npbmx = 0.5*xtras*nppmx;
/* align data to warp size */
   nppmx0 = 32*((nppmx0 - 1)/32 + 1);
   ntmax = 32*(ntmax/32 + 1);
   npbmx = 32*((npbmx - 1)/32 + 1);
   gpu_fallocate(&g_ppart,nppmx0*idimp*mxy1,&irc);
   gpu_fallocate(&g_ppbuff,npbmx*idimp*mxy1,&irc);
   gpu_iallocate(&g_kpic,mxy1,&irc);
   gpu_iallocate(&g_ncl,8*mxy1,&irc);
   gpu_iallocate(&g_ihole,2*(ntmax+1)*mxy1,&irc);
   gpu_iallocate(&g_irc,1,&irc);
   if (irc != 0) {
      printf("GPU allocate error!\n");
      exit(1);
   }
   ppart = (float *) malloc(nppmx0*idimp*mxy1*sizeof(float));

/* copy particle data for GPU code: updates ppart and kpic */

   cppmovin2lt(part,ppart,kpic,nppmx0,idimp,np,mx,my,mx1,mxy1,&irc);
   if (irc != 0) {
      printf("cppmovin2lt overflow error, irc=%d\n",irc);
      exit(1);
   }
/* sanity check */
   cppcheck2lt(ppart,kpic,idimp,nppmx0,nx,ny,mx,my,mx1,my1,&irc);
   if (irc != 0) {
      printf("cppcheck2lt error, irc=%d\n",irc);
      exit(1);
   }

/* copy to GPU */
   gpu_icopyin(&irc,g_irc,1);
   gpu_fcopyin(ppart,g_ppart,nppmx0*idimp*mxy1);
   gpu_icopyin(kpic,g_kpic,mxy1);

   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 GPU code:                */
      dtimer(&dtime,&itime,-1);
      gpu_zfmem(g_cue,ndim*nxe*nye);
      if (relativity==1) {
/* updates g_ppart, g_cue */
         cgpu2rjppost2l(g_ppart,g_cue,g_kpic,qme,dth,ci,nppmx0,idimp,
                        nx,ny,mx,my,nxe,nye,mx1,mxy1,ipbc);
/* updates g_ppart, g_cue, g_ncl, g_ihole, g_irc */
/*       cgpu2rjppostf2l(g_ppart,g_cue,g_kpic,g_ncl,g_ihole,qme,dth,ci, */
/*                       nppmx0,idimp,nx,ny,mx,my,nxe,nye,mx1,mxy1,     */
/*                       ntmax,g_irc);                                  */
      }
      else {
/* updates g_ppart, g_cue */
         cgpu2jppost2l(g_ppart,g_cue,g_kpic,qme,dth,nppmx0,idimp,nx,
                       ny,mx,my,nxe,nye,mx1,mxy1,ipbc);
/* updates g_ppart, g_cue, g_ncl, g_ihole, g_irc */
/*       cgpu2jppostf2l(g_ppart,g_cue,g_kpic,g_ncl,g_ihole,qme,dth,      */
/*                      nppmx0,idimp,nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax, */
/*                      g_irc);
      }
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tdjpost += time;

/* reorder particles by tile with GPU code:            */
      dtimer(&dtime,&itime,-1);
/* updates g_ppart, g_ppbuff, g_kpic, g_ncl, g_ihole,and g_irc */
      cgpuppord2l(g_ppart,g_ppbuff,g_kpic,g_ncl,g_ihole,idimp,nppmx0,
                  nx,ny,mx,my,mx1,my1,npbmx,ntmax,g_irc);
/* updates g_ppart, g_ppbuff, g_kpic, g_ncl, and g_irc */
/*    cgpuppordf2l(g_ppart,g_ppbuff,g_kpic,g_ncl,g_ihole,idimp,nppmx0, */
/*                 mx1,my1,npbmx,ntmax,g_irc);                         */
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tsort += time;

/* deposit charge with GPU code: updates g_qe */
      dtimer(&dtime,&itime,-1);
      gpu_zfmem(g_qe,nxe*nye);
      cgpu2ppost2l(g_ppart,g_qe,g_kpic,qme,nppmx0,idimp,mx,my,nxe,nye,
                   mx1,mxy1);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tdpost += time;

/* add and copy guard cells with GPU code: updates g_q and g_cu */
      dtimer(&dtime,&itime,-1);
      cgpucacguard2l(g_cu,g_cue,nx,ny,nxe,nye,nxhd,ny);
      cgpucaguard2l(g_q,g_qe,nx,ny,nxe,nye,nxhd,ny);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;
```

```
/* transform charge to fourier space with GPU code: updates g_q, g_qt */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cgpuwfft2rcs(g_q,g_qt,isign,g_mixup,g_sct,indx,indy,nxhd,ny,
                   nxhy,nxyh);
/* NVIDIA fft */
/*    gpufft2rrcu(g_q,g_qt,isign,indx,indy,nxhd,ny); */
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* transform current to fourier space with GPU code: updates g_cu, g_cut */
      dtimer(&dtime,&itime,-1);
      isign = -1;
      cgpuwfft2rcsn(g_cu,g_cut,isign,g_mixup,g_sct,indx,indy,ndim,
                    nxhd,ny,nxhy,nxyh);
/* NVIDIA fft */
/*    gpufft2rrcun(g_cu,g_cut,isign,indx,indy,ndim,nxhd,ny); */
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* take transverse part of current with GPU code: updates g_cut */
      dtimer(&dtime,&itime,-1);
      cgpucuperp2t(g_cut,nx,ny,nxhd,ny);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* calculate electromagnetic fields in fourier space with GPU code: */
/* updates g_exyzt, g_bxyzt, g_wf, g_wm                             */
      dtimer(&dtime,&itime,-1);
      if (ntime==0) {
         cgpuibpois23t(g_cut,g_bxyzt,g_ffct,ci,g_wm,nx,ny,nxh1,ny,nxh,
                       nyh);
         gpu_zfmem(g_wf,nxh1);
         dth = 0.5*dt;
      }
      else {
         cgpumaxwel2t(g_exyzt,g_bxyzt,g_cut,g_ffct,ci,dt,g_wf,g_wm,nx,
                      ny,nxh1,ny,nxh,nyh);
      }
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* calculate force/charge in fourier space GPU code: */
/* updates g_fxyzt, g_we                             */
      dtimer(&dtime,&itime,-1);
      cgpupois23t(g_qt,g_fxyzt,g_ffct,g_we,nx,ny,nxh1,ny,nxh,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;
```

```
/* add longitudinal and transverse electric fields with with GPU code: */
/* updates g_fxyzt                                                     */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cgpuemfield2t(g_fxyzt,g_exyzt,g_ffct,isign,nx,ny,nxh1,ny,nxh,nyh);
/* copy magnetic field with GPU code: updates g_hxyzt */
      isign = -1;
      cgpuemfield2t(g_hxyzt,g_bxyzt,g_ffct,isign,nx,ny,nxh1,ny,nxh,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* transform electric force to real space  with GPU code: */
/* updates g_fxyzt, g_fxyz                                */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cgpuwfft2rcsn(g_fxyz,g_fxyzt,isign,g_mixup,g_sct,indx,indy,ndim,
                    nxhd,ny,nxhy,nxyh);
/* NVIDIA fft */
/*    gpufft2rrcun(g_fxyz,g_fxyzt,isign,indx,indy,ndim,nxhd,ny); */
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* transform magnetic force to real space with GPU code: */
/* updates g_hxyzt, g_hxyz                               */
      dtimer(&dtime,&itime,-1);
      isign = 1;
      cgpuwfft2rcsn(g_hxyz,g_hxyzt,isign,g_mixup,g_sct,indx,indy,ndim,
                    nxhd,ny,nxhy,nxyh);
/* NVIDIA fft */
/*    gpufft2rrcun(g_hxyz,g_hxyzt,isign,indx,indy,ndim,nxhd,ny); */
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfft += time;

/* copy guard cells with GPU code: updates g_fxyze, g_bxyze */
      dtimer(&dtime,&itime,-1);
      cgpucbguard2l(g_fxyz,g_fxyze,nx,ny,nxe,nye,nxhd,ny);
      cgpucbguard2l(g_hxyz,g_bxyze,nx,ny,nxe,nye,nxhd,ny);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* push particles with GPU code:                 */
      dtimer(&dtime,&itime,-1);
      if (relativity==1) {
/* updates g_ppart, g_wke */
         cgpurbppush23l(g_ppart,g_fxyze,g_bxyze,g_kpic,qbme,dt,dth,ci,
                        g_wke,idimp,nppmx0,nx,ny,mx,my,nxe,nye,mx1,
                        mxy1,ipbc);
/* updates g_ppart, g_ncl, g_ihole, g_wke, g_irc */
/*       cgpurbppushf23l(g_ppart,g_fxyze,g_bxyze,g_kpic,g_ncl,g_ihole,  */
/*                       qbme,dt,dth,ci,g_wke,idimp,nppmx0,nx,ny,mx,my, */
/*                       nxe,nye,mx1,mxy1,ntmax,g_irc);                  */
      }
      else {
/* updates g_ppart, g_wke */
         cgpubppush23l(g_ppart,g_fxyze,g_bxyze,g_kpic,qbme,dt,dth,
                       g_wke,idimp,nppmx0,nx,ny,mx,my,nxe,nye,mx1,
                       mxy1,ipbc);
/* updates g_ppart, g_ncl, g_ihole, g_wke, g_irc */
/*       cgpubppushf23l(g_ppart,g_fxyze,g_bxyze,g_kpic,g_ncl,g_ihole,   */
/*                      qbme,dt,dth,g_wke,idimp,nppmx0,nx,ny,mx,my,nxe, */
/*                      nye,mx1,mxy1,ntmax,g_irc);                      */
      }
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tpush += time;

/* reorder particles by tile with GPU code:            */
      dtimer(&dtime,&itime,-1);
/* updates g_ppart, g_ppbuff, g_kpic, g_ncl, g_ihole,and g_irc */
      cgpuppord2l(g_ppart,g_ppbuff,g_kpic,g_ncl,g_ihole,idimp,nppmx0,
                  nx,ny,mx,my,mx1,my1,npbmx,ntmax,g_irc);
/* updates g_ppart, g_ppbuff, g_kpic, g_ncl, and g_irc */
/*    cgpuppordf2l(g_ppart,g_ppbuff,g_kpic,g_ncl,g_ihole,idimp,nppmx0, */
/*                   mx1,my1,npbmx,ntmax,g_irc);                       */
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tsort += time;

/* sanity check */
      gpu_icopyout(&irc,g_irc,1);
      if (irc != 0) {
         printf("deposit/push/reorder error: ntmax, irc=%d,%d\n",ntmax,
                irc);
         exit(1);
      }

/* energy diagnostic */
      if (ntime==0) {
         gpu_zfmem(g_sum,1);
         cgpusum2(g_we,g_sum,nxh1);
         gpu_fcopyout(&we,g_sum,1);
         gpu_zfmem(g_sum,1);
         cgpusum2(g_wf,g_sum,nxh1);
         gpu_fcopyout(&wf,g_sum,1);
         gpu_zfmem(g_sum,1);
         cgpusum2(g_wm,g_sum,nxh1);
         gpu_fcopyout(&wm,g_sum,1);
         gpu_zfmem(g_sum,1);
         cgpusum2(g_wke,g_sum,mxy1);
         gpu_fcopyout(&wke,g_sum,1);
         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 = %i\n",ntime);
   printf("relativity = %i\n",relativity);
/* energy diagnostic */
   gpu_zfmem(g_sum,1);
   cgpusum2(g_we,g_sum,nxh1);
   gpu_fcopyout(&we,g_sum,1);
   gpu_zfmem(g_sum,1);
   cgpusum2(g_wf,g_sum,nxh1);
   gpu_fcopyout(&wf,g_sum,1);
   gpu_zfmem(g_sum,1);
   cgpusum2(g_wm,g_sum,nxh1);
   gpu_fcopyout(&wm,g_sum,1);
   gpu_zfmem(g_sum,1);
   cgpusum2(g_wke,g_sum,mxy1);
   gpu_fcopyout(&wke,g_sum,1);
   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);
   printf("\n");

/* close down NVIDIA fft */
   gpufft2cudel();
   gpufft2rrcudel();
/* deallocate memory on GPU */
   gpu_deallocate((void *)g_irc,&irc);
   gpu_deallocate((void *)g_ihole,&irc);
   gpu_deallocate((void *)g_ncl,&irc);
   gpu_deallocate((void *)g_kpic,&irc);
   gpu_deallocate((void *)g_sum,&irc);
   gpu_deallocate((void *)g_wm,&irc);
   gpu_deallocate((void *)g_wf,&irc);
   gpu_deallocate((void *)g_we,&irc);
   gpu_deallocate((void *)g_wke,&irc);
   gpu_deallocate((void *)g_bxyzt,&irc);
   gpu_deallocate((void *)g_exyzt,&irc);
   gpu_deallocate((void *)g_hxyzt,&irc);
   gpu_deallocate((void *)g_fxyzt,&irc);
   gpu_deallocate((void *)g_hxyz,&irc);
   gpu_deallocate((void *)g_fxyz,&irc);
   gpu_deallocate((void *)g_cut,&irc);
   gpu_deallocate((void *)g_qt,&irc);
   gpu_deallocate((void *)g_cu,&irc);
   gpu_deallocate((void *)g_q,&irc);
   gpu_deallocate((void *)g_sct,&irc);
   gpu_deallocate((void *)g_mixup,&irc);
   gpu_deallocate((void *)g_ffct,&irc);
   gpu_deallocate((void *)g_ppbuff,&irc);
   gpu_deallocate((void *)g_ppart,&irc);
   gpu_deallocate((void *)g_bxyze,&irc);
   gpu_deallocate((void *)g_fxyze,&irc);
   gpu_deallocate((void *)g_cue,&irc);
   gpu_deallocate((void *)g_qe,&irc);
/* close down GPU */
   end_cu();

   return 0;
}
```