```
/*-----------------------------------------------------------------------*/
/* Skeleton 2D Electrostatic GPU-MPI PIC code */
/* written by Viktor K. Decyk, UCLA */
#include <stdlib.h>
#include <stdio.h>
#include <complex.h>
#include <sys/time.h>
#include "gpulib2.h"
#include "gpuppush2.h"
#include "gpupfft2.h"
#include "ppush2.h"
#include "pplib2.h"
#include "gpplib2.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 */
   int idimp = 4, ipbc = 1;
/* idps = number of partition boundaries */
   int idps = 2;
   float wke = 0.0, we = 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 nx, ny, nxh, nyh, nxh1, nxe, nye, nxeh;
   int nxyh, nxhy, ny1, mx1, ntime, nloop, isign, ierr;
   float qbme, affp;
   double np;

/* declare scalars for MPI code */
   int nvp, idproc, kstrt, npmax, kxp, kyp, nypmx, nypmn;
   int nyp, noff, npp, nps;
   int myp1, mxyp1, kxpp, kypp;

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

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

```
   float complex *sct = NULL;
   float wtot[4], work[4];

/* declare arrays for MPI code */
   float *sbufl = NULL, *sbufr = NULL, *rbufl = NULL, *rbufr = NULL;
   float *edges = NULL;
   float *scs = NULL, *scr = NULL;
   int *locl = NULL;

/* declare arrays for GPU code */
   float *g_qe = NULL, *g_fxye = NULL;
   float complex *g_ffct = NULL;
   int *g_mixup = NULL;
   float complex *g_sct = NULL;
   float complex *g_q = NULL, *g_qt = NULL;
   float complex *g_fxy = NULL, *g_fxyt = NULL;
   float *g_wke = NULL, *g_we = NULL;
   float *g_ppart = NULL, *g_ppbuff = NULL;
   int *g_kpic = NULL;
   float *g_sbufl = NULL, *g_sbufr = NULL;
   int *g_ncl = NULL, *g_ihole = NULL;
   int *g_ncll = NULL, *g_nclr = NULL;
   float complex *g_bsm = NULL, *g_brm = NULL;
   float *g_scs = NULL;
   float *g_sum = NULL;
   int *g_irc = NULL;
   float complex *qt = NULL;
   float complex *fxyt = NULL;
   float *ppart = NULL;
   int *kpic = NULL;
   int *ncll = NULL, *nclr = NULL, *mcll = NULL, *mclr = NULL;
   float complex *bsm = NULL, *brm = NULL;

/* declare and initialize timing data */
   float time;
   struct timeval itime;
   float tpush = 0.0, tdpost = 0.0, tsort = 0.0;
   float tmov = 0.0, tfield = 0.0, tguard = 0.0;
   float tmsort[2] = {0.0,0.0};
   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;
   nxh1 = nxh + 1;
   nxe = nx + 2; nye = ny + 2; nxeh = nxe/2;
   nxyh = (nx > ny ? nx : ny)/2; nxhy = nxh > ny ? nxh : ny;
   ny1 = ny + 1; mx1 = (nx - 1)/mx + 1;
   nloop = tend/dt + .0001; ntime = 0;
   qbme = qme;
   affp = (double) nx*(double) ny/np;
/* set size for FFT arrays */
   nxhd = nxh1;
```

```
/* 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 */
/* kxpd = number of complex grids in fft partition in x direction       */
   kxp = (nxh - 1)/nvp + 1;
/* set size for FFT arrays */
   kxpd = (nxh1 - 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;
/* myp1 = number of tiles in y direction */
   myp1 = (nyp - 1)/my + 1; mxyp1 = mx1*myp1;
/* kxpp/kypp = actual size of GPU field partition */
   kxpp = nxhd - kxpd*idproc;
   kxpp = 0 > kxpp ? 0 : kxpp;
   kxpp = kxpd < kxpp ? kxpd : kxpp;
   kypp = ny - kyp*idproc;
   kypp = 0 > kypp ? 0 : kypp;
   kypp = kyp < kypp ? kyp : kypp;

/* allocate and initialize data for standard code */
   part = (float *) malloc(idimp*npmax*sizeof(float));
   ffct = (float complex *) malloc(nyh*kxpd*sizeof(float complex));
   mixup = (int *) malloc(nxhy*sizeof(int));
   sct = (float complex *) malloc(nxyh*sizeof(float complex));
   kpic = (int *) malloc(mxyp1*sizeof(int));
```

```
   qt = (float complex *) malloc(ny*kxpd*sizeof(float complex));
   fxyt = (float complex *) malloc(ny*ndim*kxpd*sizeof(float complex));

/* allocate and initialize data for MPI code */
   locl = (int *) malloc(nvp*sizeof(int));

/* set up GPU */
   irc = 0;
/* get unique GPU device ids */
   cppfndgrp(locl,kstrt,nvp,&idev,&ndev);
   if (idev < 0) {
      printf("%d,GPU device error!\n",kstrt);
      cppabort();
      exit(1);
   }
   gpu_setgbsize(nblock);
   init_cu(idev,&irc);
   if (irc != 0) {
      printf("%d,CUDA initialization error!\n",kstrt);
      cppabort();
      exit(1);
   }
/* obtain compute capability */
   mmcc = getmmcc();
   if (mmcc < 20) {
      printf("%d,compute capability 2.x or higher required\n",kstrt);
      cppabort();
      exit(1);
   }
/* set cache size */
   gpu_set_cache_size(nscache);
/* create asynchronous streams */
   gpu_initstream(1);
   gpu_initstream(2);
   gpu_initstream(3);
/* allocate and initialize data for GPU code */
   gpu_fallocate(&g_qe,nxe*nypmx,&irc);
   gpu_fallocate(&g_fxye,ndim*nxe*nypmx,&irc);
   gpu_callocate(&g_ffct,nyh*kxpd,&irc);
   gpu_iallocate(&g_mixup,nxhy,&irc);
   gpu_callocate(&g_sct,nxyh,&irc);
   gpu_callocate(&g_q,nxhd*kyp,&irc);
   gpu_callocate(&g_qt,ny*kxpd,&irc);
   gpu_callocate(&g_fxy,nxhd*ndim*kyp,&irc);
   gpu_callocate(&g_fxyt,ny*ndim*kxpd,&irc);
   gpu_fallocate(&g_wke,mxyp1,&irc);
   gpu_fallocate(&g_we,kxpd,&irc);
   gpu_fallocate(&g_sum,1,&irc);
   if (irc != 0) {
      printf("%d,GPU allocate error!\n",kstrt);
      cppabort();
      exit(1);
   }
```

```
/* prepare fft tables */
   cwpfft2rinit(mixup,sct,indx,indy,nxhy,nxyh);
/* prepare NVIDIA ffts */
   gpupfft2rrcuinit(nx,kypp,ndim);
   gpupfft2cuinit(kxpp,ny,ndim);
/* calculate form factors */
   isign = 0;
   cppois22t(qt,fxyt,isign,ffct,ax,ay,affp,&we,nx,ny,kstrt,ny,kxpd,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*kxpd);
/* 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;
   }

/* find number of particles in each of mx, my tiles: updates kpic, nppmx */
   cppdblkp2l(part,kpic,npp,noff,&nppmx,idimp,npmax,mx,my,mx1,mxyp1,
              &irc);
   if (irc != 0) {
      printf("%d,cppdblkp2l error, irc=%d\n",kstrt,irc);
      cppabort();
      exit(1);
   }
/* allocate vector particle data */
   nppmx0 = (1.0 + xtras)*nppmx;
   ntmaxp = xtras*nppmx;
   npbmx = xtras*nppmx;
   nbmaxp = 0.25*mx1*npbmx;
/* align data to warp size */
   nppmx0 = 32*((nppmx0 - 1)/32 + 1);
   ntmaxp = 32*(ntmaxp/32 + 1);
   npbmx = 32*((npbmx - 1)/32 + 1);
   nbmaxp = 32*((nbmaxp - 1)/32 + 1);
   gpu_fallocate(&g_ppart,nppmx0*idimp*mxyp1,&irc);
   gpu_fallocate(&g_ppbuff,npbmx*idimp*mxyp1,&irc);
   gpu_iallocate(&g_kpic,mxyp1+1,&irc);
   gpu_fallocate(&g_sbufl,nbmaxp*idimp,&irc);
   gpu_fallocate(&g_sbufr,nbmaxp*idimp,&irc);
   gpu_iallocate(&g_ncl,8*mxyp1,&irc);
   gpu_iallocate(&g_ihole,2*(ntmaxp+1)*mxyp1,&irc);
   gpu_iallocate(&g_ncll,3*mx1,&irc);
   gpu_iallocate(&g_nclr,3*mx1,&irc);
   gpu_callocate(&g_bsm,kxpd*ndim*kyp*nvp,&irc);
   gpu_callocate(&g_brm,kxpd*ndim*kyp*nvp,&irc);
```

```
   gpu_fallocate(&g_scs,nxe*ndim,&irc);
   gpu_iallocate(&g_irc,1,&irc);
   if (irc != 0) {
      printf("%d,GPU allocate error!\n",kstrt);
      cppabort();
      exit(1);
   }
   ppart = (float *) malloc(nppmx0*idimp*mxyp1*sizeof(float));
   ncll = (int *) malloc(3*mxyp1*sizeof(int));
   nclr = (int *) malloc(3*mxyp1*sizeof(int));
   mcll = (int *) malloc(3*mxyp1*sizeof(int));
   mclr = (int *) malloc(3*mxyp1*sizeof(int));

/* allocate data for GPU-MPI buffers */
/* scs = (float *) malloc(nxe*ndim*sizeof(float));       */
/* scr = (float *) malloc(nxe*ndim*sizeof(float));       */
/* sbufl = (float *) malloc(idimp*nbmaxp*sizeof(float)); */
/* sbufr = (float *) malloc(idimp*nbmaxp*sizeof(float)); */
/* rbufl = (float *) malloc(idimp*nbmaxp*sizeof(float)); */
/* rbufr = (float *) malloc(idimp*nbmaxp*sizeof(float)); */
/* bsm = (float complex *) malloc(kxpd*ndim*kyp*nvp*     */
/*        sizeof(float complex));                        */
/* brm = (float complex *) malloc(kxpd*ndim*kyp*nvp*     */
/*        sizeof(float complex));                        */
/* allocate host page-locked memory for GPU-MPI buffers  */
   hpl_fallocate(&scs,nxe*ndim,&irc);
   hpl_fallocate(&scr,nxe*ndim,&irc);
   hpl_fallocate(&sbufl,idimp*nbmaxp,&irc);
   hpl_fallocate(&sbufr,idimp*nbmaxp,&irc);
   hpl_fallocate(&rbufl,idimp*nbmaxp,&irc);
   hpl_fallocate(&rbufr,idimp*nbmaxp,&irc);
   hpl_callocate(&bsm,kxpd*ndim*kyp*nvp,&irc);
   hpl_callocate(&brm,kxpd*ndim*kyp*nvp,&irc);
   if (irc != 0) {
      printf("%d,hpl_fallocate error, irc=%d\n",kstrt,irc);
      cppabort();
      exit(1);
   }

/* copy ordered particle data for GPU code: updates ppart, kpic */
   cpppmovin2lt(part,ppart,kpic,npp,noff,nppmx0,idimp,npmax,mx,my,mx1,
                mxyp1,&irc);
   if (irc != 0) {
      printf("%d,cpppmovin2lt overflow error, irc=%d\n",kstrt,irc);
      cppabort();
      exit(1);
   }
/* sanity check */
   cpppcheck2lt(ppart,kpic,noff,nyp,idimp,nppmx0,nx,mx,my,mx1,myp1,
                &irc);
   if (irc != 0) {
      printf("%d,cpppcheck2lt error: irc=%d\n",kstrt,irc);
      cppabort();
      exit(1);
```

```
   }
/* copy to GPU */
   gpu_icopyin(&irc,g_irc,1);
   gpu_fcopyin(ppart,g_ppart,nppmx0*idimp*mxyp1);
   gpu_icopyin(kpic,g_kpic,mxyp1);
   gpu_zfmem(g_we,kxpd);

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

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

/* deposit charge with GPU code: updates g_qe */
      dtimer(&dtime,&itime,-1);
      gpu_zfmem(g_qe,nxe*nypmx);
      cgpu2ppgppost2l(g_ppart,g_qe,g_kpic,noff,qme,idimp,nppmx0,mx,my,
                      nxe,nypmx,mx1,mxyp1);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tdpost += time;

/* add and copy guard cells with GPU code: updates g_q */
      dtimer(&dtime,&itime,-1);
      cgppcaguard2l(g_q,g_qe,g_scs,scs,scr,nx,nyp,kstrt,nvp,nxe,
                    nypmx,nxhd,kyp);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* transform charge to fourier space with GPU code: updates g_q, g_qt, */
/* as well as various buffers */
      isign = -1;
      cwappfft2rcs(g_q,g_qt,g_bsm,g_brm,bsm,brm,isign,g_mixup,g_sct,
                   tfft,indx,indy,kstrt,nvp,kxpd,kyp,nxhd,ny,kyp,nxhy,
                   nxyh);
/* NVIDIA fft */
/*    gpuppfft2rrcu(g_q,g_qt,g_bsm,g_brm,bsm,brm,isign,tfft,indx,indy, */
/*                  kstrt,nvp,kxpd,kyp,nxhd,ny,kyp);                   */

/* calculate force/charge in fourier space with GPU code: */
/* updates g_fxyt, g_we */
      dtimer(&dtime,&itime,-1);
      cgpuppois22t(g_qt,g_fxyt,g_ffct,g_we,nx,ny,kstrt,ny,kxpd,nyh);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tfield += time;

/* transform force to real space with GPU code: updates g_fxy, g_fxyt, */
/* as well as various buffers */
      isign = 1;
      cwappfft2rcsn(g_fxy,g_fxyt,g_bsm,g_brm,bsm,brm,isign,g_mixup,
                    g_sct,tfft,indx,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,
                    ny,kyp,nxhy,nxyh);
```

```
/* NVIDIA fft */
/*    gpuppfft2rrcun(g_fxy,g_fxyt,g_bsm,g_brm,bsm,brm,isign,tfft,indx, */
/*                   indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp);        */

/* copy guard cells with GPU code: updates g_fxye */
      dtimer(&dtime,&itime,-1);
      cgppccguard2l(g_fxy,g_fxye,g_scs,scs,scr,nx,nyp,kstrt,nvp,ndim,
                    nxe,nypmx,nxhd,kyp);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tguard += time;

/* push particles with GPU code: updates g_ppart, g_wke */
      dtimer(&dtime,&itime,-1);
      cgpuppgppush2l(g_ppart,g_fxye,g_kpic,noff,nyp,qbme,dt,g_wke,nx,ny,
                     mx,my,idimp,nppmx0,nxe,nypmx,mx1,mxyp1,ipbc);
      dtimer(&dtime,&itime,1);
      time = (float) dtime;
      tpush += time;

/*reorder particles by tile with GPU code: */
/* updates g_ppart, g_ppbuff, g_kpic, g_ncl, g_ihole, and g_irc */
/* as well as various buffers */
      cgpporder2l(g_ppart,g_ppbuff,g_sbufl,g_sbufr,g_kpic,g_ncl,g_ihole,
                  g_ncll,g_nclr,sbufl,sbufr,rbufl,rbufr,ncll,nclr,mcll,
                  mclr,tmsort,noff,nyp,kstrt,nvp,idimp,nppmx0,nx,ny,mx,
                  my,mx1,myp1,npbmx,ntmaxp,nbmaxp,g_irc);
      tsort = tmsort[0];
      tmov = tmsort[1];

/* sanity check */
      gpu_icopyout(&irc,g_irc,1);
      if (irc != 0) {
         printf("%d,cgpupppord2l error: irc=%d\n",kstrt,irc);
         cppabort();
         exit(1);
      }

/* energy diagnostic */
      if (ntime==0) {
         gpu_zfmem(g_sum,1);
         cgpusum2(g_we,g_sum,kxpd);
         gpu_fcopyout(&we,g_sum,1);
         gpu_zfmem(g_sum,1);
         cgpusum2(g_wke,g_sum,mxyp1);
         gpu_fcopyout(&wke,g_sum,1);
         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 (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 * * * */

/* energy diagnostic */
   gpu_zfmem(g_sum,1);
   cgpusum2(g_we,g_sum,kxpd);
   gpu_fcopyout(&we,g_sum,1);
   gpu_zfmem(g_sum,1);
   cgpusum2(g_wke,g_sum,mxyp1);
   gpu_fcopyout(&wke,g_sum,1);
   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 (kstrt==1) {
      printf("ntime = %i\n",ntime);
      printf("MPI nodes nvp = %i, GPUs per host = %i\n",nvp,ndev);
      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 times = %f,%f,%f\n",
             tfft[0]+tfft[1],tfft[0],tfft[1]);
      printf("push time = %f\n",tpush);
      printf("move time = %f\n",tmov);
      printf("sort time = %f\n",tsort);
      tfield += tguard + (tfft[0]+tfft[1]);
      printf("total solver time = %f\n",tfield);
      time = tdpost + tpush + tsort + tmov;
      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("Move Time (nsec) = %f\n",tmov*wt);
      printf("Total Particle Time (nsec) = %f\n",time*wt);
```

```
   }

/* close down NVIDIA fft */
   gpupfft2cudel();
   gpupfft2rrcudel();
/* deallocate memory on GPU */
   gpu_deallocate((void *)g_irc,&irc);
   gpu_deallocate((void *)g_scs,&irc);
   gpu_deallocate((void *)g_brm,&irc);
   gpu_deallocate((void *)g_bsm,&irc);
   gpu_deallocate((void *)g_nclr,&irc);
   gpu_deallocate((void *)g_ncll,&irc);
   gpu_deallocate((void *)g_ihole,&irc);
   gpu_deallocate((void *)g_ncl,&irc);
   gpu_deallocate((void *)g_sbufr,&irc);
   gpu_deallocate((void *)g_sbufl,&irc);
   gpu_deallocate((void *)g_kpic,&irc);
   gpu_deallocate((void *)g_ppbuff,&irc);
   gpu_deallocate((void *)g_ppart,&irc);
   gpu_deallocate((void *)g_sum,&irc);
   gpu_deallocate((void *)g_we,&irc);
   gpu_deallocate((void *)g_wke,&irc);
   gpu_deallocate((void *)g_fxyt,&irc);
   gpu_deallocate((void *)g_fxy,&irc);
   gpu_deallocate((void *)g_qt,&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_fxye,&irc);
   gpu_deallocate((void *)g_qe,&irc);
/* deallocate host page-locked memory */
   hpl_deallocate((void *)scs,&irc); scs = NULL;
   hpl_deallocate((void *)scr,&irc); scr = NULL;
   hpl_deallocate((void *)sbufl,&irc); sbufl = NULL;
   hpl_deallocate((void *)sbufr,&irc); sbufr = NULL;
   hpl_deallocate((void *)rbufl,&irc); rbufl = NULL;
   hpl_deallocate((void *)rbufr,&irc); rbufr = NULL;
   hpl_deallocate((void *)bsm,&irc); bsm = NULL;
   hpl_deallocate((void *)brm,&irc); brm = NULL;
L3000:

/* delete asynchronous streams */
   gpu_delstream(3);
   gpu_delstream(2);
   gpu_delstream(1);
/* close down GPU */
   end_cu();
/* close down MPI */
   cppexit();

   return 0;
}
```

```
/* Only used by Fortran, not needed in C */
void getfcptr_(unsigned long *carrayref, float *carray, int *nx) {
   return;
}

void getf2cptr_(unsigned long *carrayref, float *carray, int *nx,
                int *ny) {
   return;
}

void getc2cptr_(unsigned long *carrayref, float complex *carray,
                int *nx, int *ny) {
   return;
}
```