```
#include "config.h"
#include "mesh_constants_cuda.h"
#include <cuda.h>
#include <cuda_runtime.h>
#include <stdio.h>
__global__ void compute_forces(int nb_blocks_to_compute,
                           int NGLOB,
                           int* d_ibool,
                           int* d_phase_ispec_inner_elastic, int
num_phase_ispec_elastic,
                           int d_iphase,
                           realw* d_displ,realw* d_veloc,realw* d_accel,
                           realw* d_xix, realw* d_xiy, realw* d_xiz,
                           realw* d_etax, realw* d_etay, realw* d_etaz,
                           realw* d_gammax, realw* d_gammay, realw* d_gammaz,
                           realw* d_hprime_xx,
                           realw* d_hprimewgll_xx,
                           realw* d_wgllwgll_xy,realw* d_wgllwgll_xz,realw*
d_wgllwgll_yz,
                           realw* d_kappav, realw* d_muv,
                           int NSPEC,
                           realw* d_rhostore,
                           realw* wgll_cube,
                                                int* maskx,
                                                int* maskax,
                                                int myrank){


// elastic compute kernel without attenuation
// holds for: ATTENUATION = .false.
//            COMPUTE_AND_STORE_STRAIN = .true. or .false. (true for kernel simulations)

  int bx = blockIdx.y*gridDim.x+blockIdx.x;
  int tx = threadIdx.x;

  const int NGLL3_ALIGN = NGLL3_PADDED;

  int K = (tx/NGLL2);
  int J = ((tx-K*NGLL2)/NGLLX);
  int I = (tx-K*NGLL2-J*NGLLX);

  int active,offset;
  int iglob = 0;
  int working_element;

  realw tempx1l,tempx2l,tempx3l,tempy1l,tempy2l,tempy3l,tempz1l,tempz2l,tempz3l;
  realw xixl,xiyl,xizl,etaxl,etayl,etazl,gammaxl,gammayl,gammazl,jacobianl;
  realw duxdxl,duxdyl,duxdzl,duydxl,duydyl,duydzl,duzdxl,duzdyl,duzdzl;
  realw duxdxl_plus_duydyl,duxdxl_plus_duzdzl,duydyl_plus_duzdzl;
  realw duxdyl_plus_duydxl,duzdxl_plus_duxdzl,duzdyl_plus_duydzl;

  realw fac1,fac2,fac3,lambdal,mul,lambdalplus2mul,kappal;
  realw sigma_xx,sigma_yy,sigma_zz,sigma_xy,sigma_xz,sigma_yz;

  realw sum_terms1,sum_terms2,sum_terms3;

  // gravity variables
  realw sigma_yx,sigma_zx,sigma_zy;


  __shared__ realw s_dummyx_loc[NGLL3];
```

```
  __shared__ realw s_dummyy_loc[NGLL3];
  __shared__ realw s_dummyz_loc[NGLL3];

  __shared__ realw s_tempx1[NGLL3];
  __shared__ realw s_tempx2[NGLL3];
  __shared__ realw s_tempx3[NGLL3];

  __shared__ realw s_tempy1[NGLL3];
  __shared__ realw s_tempy2[NGLL3];
  __shared__ realw s_tempy3[NGLL3];

  __shared__ realw s_tempz1[NGLL3];
  __shared__ realw s_tempz2[NGLL3];
  __shared__ realw s_tempz3[NGLL3];

  //__shared__ realw sh_hprime_xx[NGLL2];


// use only NGLL^3 = 125 active threads, plus 3 inactive/ghost threads,
// because we used memory padding from NGLL^3 = 125 to 128 to get coalescent memory
accesses
  active = (tx < NGLL3 && bx < nb_blocks_to_compute) ? 1:0;

//  printf("\nwe are here: tx = %d\n",tx);
// copy from global memory to shared memory
// each thread writes one of the NGLL^3 = 125 data points
  if (active) {

    working_element = d_phase_ispec_inner_elastic[bx + num_phase_ispec_elastic*
(d_iphase-1)]-1;
    iglob = d_ibool[working_element*NGLL3 + tx]-1;
    // debug
    //if( iglob < 0 || iglob >= NGLOB ){ printf("wrong iglob %d\n",iglob);  }

    // changing iglob indexing to match fortran row changes fast style
    s_dummyx_loc[tx] = d_displ[iglob*3];
    s_dummyy_loc[tx] = d_displ[iglob*3 + 1];
    s_dummyz_loc[tx] = d_displ[iglob*3 + 2];
        s_dummyx_loc[tx] = s_dummyx_loc[tx] * maskx[iglob*3] * (-1.0f);
        s_dummyy_loc[tx] = s_dummyy_loc[tx] * maskx[iglob*3 + 1] * (-1.0f);
        s_dummyz_loc[tx] = s_dummyz_loc[tx] * maskx[iglob*3 + 2] * (-1.0f);

//      if(!maskx[iglob*3]) printf("maskvaluefalse : %d\n",maskx[iglob*3]);
  // JC JC here we will need to add GPU support for the new C-PML routines
//      if(maskx[iglob*3]) printf("maskvaluetrue : %d\n",maskx[iglob*3]);
  // JC JC here we will need to add GPU support for the new C-PML routines


  }


  __syncthreads();

  if (active) {



    tempx1l = s_dummyx_loc[K*NGLL2+J*NGLLX]*d_hprime_xx[I]
            + s_dummyx_loc[K*NGLL2+J*NGLLX+1]*d_hprime_xx[NGLLX+I]
            + s_dummyx_loc[K*NGLL2+J*NGLLX+2]*d_hprime_xx[2*NGLLX+I]
```

```
            + s_dummyx_loc[K*NGLL2+J*NGLLX+3]*d_hprime_xx[3*NGLLX+I]
            + s_dummyx_loc[K*NGLL2+J*NGLLX+4]*d_hprime_xx[4*NGLLX+I];

    tempy1l = s_dummyy_loc[K*NGLL2+J*NGLLX]*d_hprime_xx[I]
            + s_dummyy_loc[K*NGLL2+J*NGLLX+1]*d_hprime_xx[NGLLX+I]
            + s_dummyy_loc[K*NGLL2+J*NGLLX+2]*d_hprime_xx[2*NGLLX+I]
            + s_dummyy_loc[K*NGLL2+J*NGLLX+3]*d_hprime_xx[3*NGLLX+I]
            + s_dummyy_loc[K*NGLL2+J*NGLLX+4]*d_hprime_xx[4*NGLLX+I];

    tempz1l = s_dummyz_loc[K*NGLL2+J*NGLLX]*d_hprime_xx[I]
            + s_dummyz_loc[K*NGLL2+J*NGLLX+1]*d_hprime_xx[NGLLX+I]
            + s_dummyz_loc[K*NGLL2+J*NGLLX+2]*d_hprime_xx[2*NGLLX+I]
            + s_dummyz_loc[K*NGLL2+J*NGLLX+3]*d_hprime_xx[3*NGLLX+I]
            + s_dummyz_loc[K*NGLL2+J*NGLLX+4]*d_hprime_xx[4*NGLLX+I];

    tempx2l = s_dummyx_loc[K*NGLL2+I]*d_hprime_xx[J]
            + s_dummyx_loc[K*NGLL2+NGLLX+I]*d_hprime_xx[NGLLX+J]
            + s_dummyx_loc[K*NGLL2+2*NGLLX+I]*d_hprime_xx[2*NGLLX+J]
            + s_dummyx_loc[K*NGLL2+3*NGLLX+I]*d_hprime_xx[3*NGLLX+J]
            + s_dummyx_loc[K*NGLL2+4*NGLLX+I]*d_hprime_xx[4*NGLLX+J];

    tempy2l = s_dummyy_loc[K*NGLL2+I]*d_hprime_xx[J]
            + s_dummyy_loc[K*NGLL2+NGLLX+I]*d_hprime_xx[NGLLX+J]
            + s_dummyy_loc[K*NGLL2+2*NGLLX+I]*d_hprime_xx[2*NGLLX+J]
            + s_dummyy_loc[K*NGLL2+3*NGLLX+I]*d_hprime_xx[3*NGLLX+J]
            + s_dummyy_loc[K*NGLL2+4*NGLLX+I]*d_hprime_xx[4*NGLLX+J];

    tempz2l = s_dummyz_loc[K*NGLL2+I]*d_hprime_xx[J]
            + s_dummyz_loc[K*NGLL2+NGLLX+I]*d_hprime_xx[NGLLX+J]
            + s_dummyz_loc[K*NGLL2+2*NGLLX+I]*d_hprime_xx[2*NGLLX+J]
            + s_dummyz_loc[K*NGLL2+3*NGLLX+I]*d_hprime_xx[3*NGLLX+J]
            + s_dummyz_loc[K*NGLL2+4*NGLLX+I]*d_hprime_xx[4*NGLLX+J];

    tempx3l = s_dummyx_loc[J*NGLLX+I]*d_hprime_xx[K]
            + s_dummyx_loc[NGLL2+J*NGLLX+I]*d_hprime_xx[NGLLX+K]
            + s_dummyx_loc[2*NGLL2+J*NGLLX+I]*d_hprime_xx[2*NGLLX+K]
            + s_dummyx_loc[3*NGLL2+J*NGLLX+I]*d_hprime_xx[3*NGLLX+K]
            + s_dummyx_loc[4*NGLL2+J*NGLLX+I]*d_hprime_xx[4*NGLLX+K];

    tempy3l = s_dummyy_loc[J*NGLLX+I]*d_hprime_xx[K]
            + s_dummyy_loc[NGLL2+J*NGLLX+I]*d_hprime_xx[NGLLX+K]
            + s_dummyy_loc[2*NGLL2+J*NGLLX+I]*d_hprime_xx[2*NGLLX+K]
            + s_dummyy_loc[3*NGLL2+J*NGLLX+I]*d_hprime_xx[3*NGLLX+K]
            + s_dummyy_loc[4*NGLL2+J*NGLLX+I]*d_hprime_xx[4*NGLLX+K];

    tempz3l = s_dummyz_loc[J*NGLLX+I]*d_hprime_xx[K]
            + s_dummyz_loc[NGLL2+J*NGLLX+I]*d_hprime_xx[NGLLX+K]
            + s_dummyz_loc[2*NGLL2+J*NGLLX+I]*d_hprime_xx[2*NGLLX+K]
            + s_dummyz_loc[3*NGLL2+J*NGLLX+I]*d_hprime_xx[3*NGLLX+K]
            + s_dummyz_loc[4*NGLL2+J*NGLLX+I]*d_hprime_xx[4*NGLLX+K];
/*  if(myrank == 31 && bx == 0) printf("\ntx = %d, s_dummyx_loc=%f\n",tx,s_dummyx_loc
[tx]);
  if(myrank == 31 && bx == 0) printf("\ntx = %d, d_hprime_xx=%f\n",tx,d_hprime_xx[tx]);
  if(myrank == 31 && bx == 0) printf("\ntx = %d, tempx1l=%e,tempx2l=%e,tempx3l=%
e,tempy1l=%e\n",tx,tempx1l,tempx2l,tempx3l,tempy1l);
  if(myrank == 31 && bx == 0) printf("\nNGLLX = %d, NGLL2=%d\n",NGLLX,NGLL2);
  if(myrank == 31 && bx == 0) printf("\ntx = %d, x0=%e,x0c=%e,x1=%e,x1c=%e,x2=%e,x2c=%
e,x3=%e,x3c=%e,x4=%e,x4c=%e\n",tx,
s_dummyx_loc[K*NGLL2+J*NGLLX],d_hprime_xx[I],
s_dummyx_loc[K*NGLL2+J*NGLLX+1],d_hprime_xx[NGLLX+I],
s_dummyx_loc[K*NGLL2+J*NGLLX+2],d_hprime_xx[2*NGLLX+I],
```

```
s_dummyx_loc[K*NGLL2+J*NGLLX+3],d_hprime_xx[3*NGLLX+I],
s_dummyx_loc[K*NGLL2+J*NGLLX+4],d_hprime_xx[4*NGLLX+I]);
*/



    // JC JC here we will need to add GPU support for the new C-PML routines


// compute derivatives of ux, uy and uz with respect to x, y and z
    offset = working_element*NGLL3_ALIGN + tx;

    xixl = d_xix[offset];
    xiyl = d_xiy[offset];
    xizl = d_xiz[offset];
    etaxl = d_etax[offset];
    etayl = d_etay[offset];
    etazl = d_etaz[offset];
    gammaxl = d_gammax[offset];
    gammayl = d_gammay[offset];
    gammazl = d_gammaz[offset];

    duxdxl = xixl*tempx1l + etaxl*tempx2l + gammaxl*tempx3l;
    duxdyl = xiyl*tempx1l + etayl*tempx2l + gammayl*tempx3l;
    duxdzl = xizl*tempx1l + etazl*tempx2l + gammazl*tempx3l;

    duydxl = xixl*tempy1l + etaxl*tempy2l + gammaxl*tempy3l;
    duydyl = xiyl*tempy1l + etayl*tempy2l + gammayl*tempy3l;
    duydzl = xizl*tempy1l + etazl*tempy2l + gammazl*tempy3l;

    duzdxl = xixl*tempz1l + etaxl*tempz2l + gammaxl*tempz3l;
    duzdyl = xiyl*tempz1l + etayl*tempz2l + gammayl*tempz3l;
    duzdzl = xizl*tempz1l + etazl*tempz2l + gammazl*tempz3l;

        //for dbg
/*      if(myrank == 31 && iglob == 1) printf("gpu info: %d: duxdxl= %
f",iglob,duxdxl);*/
    // JC JC here we will need to add GPU support for the new C-PML routines

    // precompute some sums to save CPU time
    duxdxl_plus_duydyl = duxdxl + duydyl;
    duxdxl_plus_duzdzl = duxdxl + duzdzl;
    duydyl_plus_duzdzl = duydyl + duzdzl;
    duxdyl_plus_duydxl = duxdyl + duydxl;
    duzdxl_plus_duxdzl = duzdxl + duxdzl;
    duzdyl_plus_duydzl = duzdyl + duydzl;

    // JC JC here we will need to add GPU support for the new C-PML routines

    // computes deviatoric strain for kernel calculations

    // compute elements with an elastic isotropic rheology
    kappal = d_kappav[offset];
    mul = d_muv[offset];

    // full anisotropic case, stress calculations
      // isotropic case

      lambdalplus2mul = kappal + 1.3333333333333333333f * mul;  // 4./3. = 1.3333333
      lambdal = lambdalplus2mul - 2.0f * mul;
/*
```

```
        if(tx == 4 && myrank == 31 && bx == 0) printf("\n4__stempx1: %e,%e,%e,%e,%e
\n",lambdal,mul,duxdxl,duydyl,duzdzl);
        if(tx == 3 && myrank == 31 && bx == 0) printf("\n3__stempx1: %e,%e,%e,%e,%e
\n",lambdal,mul,duxdxl,duydyl,duzdzl);
*/

      // compute the six components of the stress tensor sigma
      sigma_xx = lambdalplus2mul*duxdxl + lambdal*duydyl_plus_duzdzl;
      sigma_yy = lambdalplus2mul*duydyl + lambdal*duxdxl_plus_duzdzl;
      sigma_zz = lambdalplus2mul*duzdzl + lambdal*duxdxl_plus_duydyl;

      sigma_xy = mul*duxdyl_plus_duydxl;
      sigma_xz = mul*duzdxl_plus_duxdzl;
      sigma_yz = mul*duzdyl_plus_duydzl;

    jacobianl = 1.0f / (xixl*(etayl*gammazl-etazl*gammayl)-xiyl*(etaxl*gammazl-
etazl*gammaxl)+xizl*(etaxl*gammayl-etayl*gammaxl));

/*      if(myrank == 31 && iglob == 1) printf("gpu:jacobian:%f,I,%d,J,%d,K,%d,element,%
d,iphase,%d",jacobianl,I,J,K,working_element,d_iphase);*/
    // define symmetric components (needed for non-symmetric dot product and sigma for
gravity)
    sigma_yx = sigma_xy;
    sigma_zx = sigma_xz;
    sigma_zy = sigma_yz;

    // form dot product with test vector, non-symmetric form
    s_tempx1[tx] = jacobianl * (sigma_xx*xixl + sigma_yx*xiyl + sigma_zx*xizl);
    s_tempy1[tx] = jacobianl * (sigma_xy*xixl + sigma_yy*xiyl + sigma_zy*xizl);
    s_tempz1[tx] = jacobianl * (sigma_xz*xixl + sigma_yz*xiyl + sigma_zz*xizl);

    s_tempx2[tx] = jacobianl * (sigma_xx*etaxl + sigma_yx*etayl + sigma_zx*etazl);
    s_tempy2[tx] = jacobianl * (sigma_xy*etaxl + sigma_yy*etayl + sigma_zy*etazl);
    s_tempz2[tx] = jacobianl * (sigma_xz*etaxl + sigma_yz*etayl + sigma_zz*etazl);

    s_tempx3[tx] = jacobianl * (sigma_xx*gammaxl + sigma_yx*gammayl + sigma_zx*gammazl);
    s_tempy3[tx] = jacobianl * (sigma_xy*gammaxl + sigma_yy*gammayl + sigma_zy*gammazl);
    s_tempz3[tx] = jacobianl * (sigma_xz*gammaxl + sigma_yz*gammayl + sigma_zz*gammazl);

  }

// synchronize all the threads (one thread for each of the NGLL grid points of the
// current spectral element) because we need the whole element to be ready in order
// to be able to compute the matrix products along cut planes of the 3D element below
  __syncthreads();

  // JC JC here we will need to add GPU support for the new C-PML routines

  if (active) {

    tempx1l = s_tempx1[K*NGLL2+J*NGLLX]*d_hprimewgll_xx[I*NGLLX]
            + s_tempx1[K*NGLL2+J*NGLLX+1]*d_hprimewgll_xx[I*NGLLX+1]
            + s_tempx1[K*NGLL2+J*NGLLX+2]*d_hprimewgll_xx[I*NGLLX+2]
            + s_tempx1[K*NGLL2+J*NGLLX+3]*d_hprimewgll_xx[I*NGLLX+3]
            + s_tempx1[K*NGLL2+J*NGLLX+4]*d_hprimewgll_xx[I*NGLLX+4];

    tempy1l = s_tempy1[K*NGLL2+J*NGLLX]*d_hprimewgll_xx[I*NGLLX]
            + s_tempy1[K*NGLL2+J*NGLLX+1]*d_hprimewgll_xx[I*NGLLX+1]
            + s_tempy1[K*NGLL2+J*NGLLX+2]*d_hprimewgll_xx[I*NGLLX+2]
            + s_tempy1[K*NGLL2+J*NGLLX+3]*d_hprimewgll_xx[I*NGLLX+3]
            + s_tempy1[K*NGLL2+J*NGLLX+4]*d_hprimewgll_xx[I*NGLLX+4];
```

```
    tempz1l = s_tempz1[K*NGLL2+J*NGLLX]*d_hprimewgll_xx[I*NGLLX]
            + s_tempz1[K*NGLL2+J*NGLLX+1]*d_hprimewgll_xx[I*NGLLX+1]
            + s_tempz1[K*NGLL2+J*NGLLX+2]*d_hprimewgll_xx[I*NGLLX+2]
            + s_tempz1[K*NGLL2+J*NGLLX+3]*d_hprimewgll_xx[I*NGLLX+3]
            + s_tempz1[K*NGLL2+J*NGLLX+4]*d_hprimewgll_xx[I*NGLLX+4];

    tempx2l = s_tempx2[K*NGLL2+I]*d_hprimewgll_xx[J*NGLLX]
            + s_tempx2[K*NGLL2+NGLLX+I]*d_hprimewgll_xx[J*NGLLX+1]
            + s_tempx2[K*NGLL2+2*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+2]
            + s_tempx2[K*NGLL2+3*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+3]
            + s_tempx2[K*NGLL2+4*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+4];

    tempy2l = s_tempy2[K*NGLL2+I]*d_hprimewgll_xx[J*NGLLX]
            + s_tempy2[K*NGLL2+NGLLX+I]*d_hprimewgll_xx[J*NGLLX+1]
            + s_tempy2[K*NGLL2+2*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+2]
            + s_tempy2[K*NGLL2+3*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+3]
            + s_tempy2[K*NGLL2+4*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+4];

    tempz2l = s_tempz2[K*NGLL2+I]*d_hprimewgll_xx[J*NGLLX]
            + s_tempz2[K*NGLL2+NGLLX+I]*d_hprimewgll_xx[J*NGLLX+1]
            + s_tempz2[K*NGLL2+2*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+2]
            + s_tempz2[K*NGLL2+3*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+3]
            + s_tempz2[K*NGLL2+4*NGLLX+I]*d_hprimewgll_xx[J*NGLLX+4];

    tempx3l = s_tempx3[J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX]
            + s_tempx3[NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+1]
            + s_tempx3[2*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+2]
            + s_tempx3[3*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+3]
            + s_tempx3[4*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+4];

    tempy3l = s_tempy3[J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX]
            + s_tempy3[NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+1]
            + s_tempy3[2*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+2]
            + s_tempy3[3*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+3]
            + s_tempy3[4*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+4];

    tempz3l = s_tempz3[J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX]
            + s_tempz3[NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+1]
            + s_tempz3[2*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+2]
            + s_tempz3[3*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+3]
            + s_tempz3[4*NGLL2+J*NGLLX+I]*d_hprimewgll_xx[K*NGLLX+4];


    fac1 = d_wgllwgll_yz[K*NGLLX+J];
    fac2 = d_wgllwgll_xz[K*NGLLX+I];
    fac3 = d_wgllwgll_xy[J*NGLLX+I];

    sum_terms1 =  (fac1*tempx1l + fac2*tempx2l + fac3*tempx3l) * maskax[iglob*3] *
(-1.0f);
    sum_terms2 =  (fac1*tempy1l + fac2*tempy2l + fac3*tempy3l) * maskax[iglob*3 + 1] *
(-1.0f);
    sum_terms3 =  (fac1*tempz1l + fac2*tempz2l + fac3*tempz3l) * maskax[iglob*3 + 2] *
(-1.0f);

    // adds gravity term
      atomicAdd(&d_accel[iglob*3], sum_terms1);
      atomicAdd(&d_accel[iglob*3+1], sum_terms2);
      atomicAdd(&d_accel[iglob*3+2], sum_terms3);
//        if(iglob == 362503 && myrank == 4) printf("\nGPU side: %e daccel: d_accel= %
e, block: %d iphase: %d\n",sum_terms1,d_accel[iglob*3],working_element,d_iphase);
```

```
//        if(iglob == 26164   && myrank==11) printf("\nGPU side: %e daccel: d_accel= %
e, block: %d iphase: %d\n",sum_terms1,d_accel[iglob*3],working_element,d_iphase);
//        if(myrank == 11 && working_element == 344) printf("\nGPUint:%d,%d
\n",iglob,maskx[3*iglob]);


/*        if(iglob == 1 && myrank == 31) printf("\ngpu : delta: %f\n",sum_terms1);
          if(iglob == 1 && myrank == 31) printf("\ngpu : fac1 %f tempx1l%f\n, s_tempx1,%
f,tx= %d\n",fac1,tempx1l,s_tempx1[tx],tx);

          if(iglob == 1 && myrank == 31)
          {
          for(int ii = 0;ii<125;ii++) printf("\nstemp%d: = %f\n",ii,s_tempx1[ii]);
          }
          */
  } // if(active)
} // kernel_2_noatt_impl()


extern "C"
void FC_FUNC_(compute_forces_fault,
                      COMPUTE_FORCES_FAULT)(long* Mesh_pointer,
                                  int* iphase,
                                  realw* deltat,
                                                                  realw*
CG_d_displ,
                                                                  realw*
CG_d_accel,
                                                                  int* maskx,
                                                                  int* maskax,
                                  int* nspec_outer_elastic,
                                  int* nspec_inner_elastic,
                                                                  int* myrank )
{

  TRACE("\tcompute_forces_fault");
  // EPIK_TRACER("compute_forces_viscoelastic_cuda");
  //printf("Running compute_forces\n");
  //double start_time = get_time();

  Mesh* mp = (Mesh*)(*Mesh_pointer); // get Mesh from fortran integer wrapper

  int num_elements;

  if( *iphase == 1 )
    num_elements = *nspec_outer_elastic;
  else
    num_elements = *nspec_inner_elastic;

  // checks if anything to do
  if( num_elements == 0 ) return;

  int blocksize = NGLL3_PADDED;

  int num_blocks_x, num_blocks_y;
  get_blocks_xy(num_elements,&num_blocks_x,&num_blocks_y);

  dim3 grid(num_blocks_x,num_blocks_y);
  dim3 threads(blocksize,1,1);
```

```
  compute_forces<<<grid,threads>>>(
                  num_elements,
                  mp->NGLOB_AB,
                  mp->d_ibool,
                  mp->d_phase_ispec_inner_elastic,
                  mp->num_phase_ispec_elastic,
                  *iphase,
                  CG_d_displ,   mp->d_veloc,    CG_d_accel,
                  mp->d_xix,    mp->d_xiy,              mp->d_xiz,
                  mp->d_etax,   mp->d_etay,             mp->d_etaz,
                  mp->d_gammax, mp->d_gammay,   mp->d_gammaz,
                  mp->d_hprime_xx,mp->d_hprimewgll_xx,
                  mp->d_wgllwgll_xy,
                  mp->d_wgllwgll_xz,
                  mp->d_wgllwgll_yz,
                  mp->d_kappav, mp->d_muv,
                  mp->NSPEC_AB,
                  mp->d_rhostore,
                  mp->d_wgll_cube,
                  maskx,
                  maskax,
                  *myrank);

}
```