GPUE2

0

Generated by Doxygen 1.8.7

Mon Apr 20 2015 10:44:26

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Namespace Documentation

3.1 hist3d Namespace Reference

Functions

- def plot_xyz_histogram
- def plot_hist_pcolor

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple sep = (c.getfloat('Params','dx'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple dt = (c.getfloat('Params','dt'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yMax = (c.getfloat('Params','yMax'))
- tuple num_vort = int(c.getfloat('Params','Num_vort'))

3.1.1 Function Documentation

3.1.1.1 def hist3d.plot_hist_pcolor (*start, fin, incr, barcolor*)

Definition at line 54 of file hist3d.py.

```
00054
00055 def plot_hist_pcolor(start,fin,incr, barcolor):
00056     fig = plt.figure()
00057
00058     data =[]
00059     for i in range(start, fin, incr):
00060         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',' )
00061         datatmp=[]
00062         count=0
00063
00064         for i1 in range(0,v_arr.size/2):
00065             for i2 in range(i1,v_arr.size/2):
00066                 m_tmp = m.sqrt(abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2  +  abs(v_arr[i1][1]*sep - v_arr[i2][1]*sep)**2 )
00067                 datatmp.append( m_tmp )
00068                 count = count + 1
```

```
00069         hist=np.histogram(datatmp,bins=np.arange(0.0,240.0,0.1))
00070         data.append(hist[:][0])
00071 
00072       #  print data
00073         ax = fig.add_subplot(111)
00074         ax.imshow(data)
00075     plt.gca().invert_yaxis()
00076         ax.set_aspect('auto')
00077 #         plt.jet()
00078     fig.savefig("HIST_PCOLOR.pdf")
00079 
00080 #plot_xyz_histogram(0,100000,100,'b')
00081 #plot_hist_pcolor(0,100000,100,'b')
00082 
```

3.1.1.2 def hist3d.plot_xyz_histogram (*start, fin, incr, barcolor*)

Definition at line 24 of file hist3d.py.

```
00024 
00025 def plot_xyz_histogram(start,fin,incr, barcolor):
00026     fig = plt.figure()
00027     ax = Axes3D(fig)
00028     data =[]
00029     for i in range(start, fin, incr):
00030         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',' )
00031         datatmp=[]
00032         count=0
00033 
00034         for i1 in range(0,v_arr.size/2):
00035             for i2 in range(i1,v_arr.size/2):
00036                 datatmp.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2  +  abs(v_arr[i1][1]*sep
      - v_arr[i2][1]*sep)**2 ))
00037                 count = count + 1
00038         hist=np.histogram(datatmp,bins=np.arange(1.0,m.sqrt(xDim**2 + yDim**2),1.0))
00039         data.append(hist[:][0])
00040     """ Takes in a matrix (see structure above) and generate a pseudo-3D histogram by overlaying close,
      semitransparent bars. """
00041     for time, occurrence in zip(range(len(data)), data):
00042         dist = range(len(occurrence))
00043         barband = range(-45, 45, 5)
00044         #for modifier in barband:
00045         ax.bar(dist, occurrence, zs=time, zdir='y', color=np.random.rand(3,1), alpha=0.8)
00046             #ax.bar(current, occurrence, zs=duration+(float(modifier)/100), zdir='y',
      color=np.random.rand(3,1), alpha=0.6)
00047 
00048     ax.set_xlabel('Dist')
00049     ax.set_ylabel('Time')
00050     ax.set_zlabel('Occurrances')
00051 
00052     plt.savefig("HIST_N.pdf")
00053     plt.show()
```

3.1.2 Variable Documentation

3.1.2.1 tuple hist3d.c = ConfigParser.ConfigParser()

Definition at line 8 of file hist3d.py.

Referenced by complexDiv(), and conj().

3.1.2.2 tuple hist3d.dt = (c.getfloat('Params','dt'))

Definition at line 18 of file hist3d.py.

3.1.2.3 tuple hist3d.dx = (c.getfloat('Params','dx'))

Definition at line 17 of file hist3d.py.

3.1.2.4 tuple hist3d.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 14 of file hist3d.py.

3.1.2.5 tuple hist3d.gndMaxVal = int(c.getfloat('Params','gsteps'))

Definition at line 13 of file hist3d.py.

3.1.2.6 tuple hist3d.incr = int(c.getfloat('Params','print_out'))

Definition at line 15 of file hist3d.py.

3.1.2.7 tuple hist3d.num_vort = int(c.getfloat('Params','Num_vort'))

Definition at line 21 of file hist3d.py.

3.1.2.8 float hist3d.sep = (c.getfloat('Params','dx'))

Definition at line 16 of file hist3d.py.

3.1.2.9 tuple hist3d.xDim = int(c.getfloat('Params','xDim'))

Definition at line 11 of file hist3d.py.

3.1.2.10 tuple hist3d.xMax = (c.getfloat('Params','xMax'))

Definition at line 19 of file hist3d.py.

3.1.2.11 tuple hist3d.yDim = int(c.getfloat('Params','yDim'))

Definition at line 12 of file hist3d.py.

3.1.2.12 tuple hist3d.yMax = (c.getfloat('Params','yMax'))

Definition at line 20 of file hist3d.py.

3.2 hist_it Namespace Reference

3.3 image_gen Namespace Reference

3.4 observables Namespace Reference

Functions

- def kinertrum
- def dens_struct_fact
- def energy_total
- def energy_kinetic

- def energy_potential
- def ang_mom
- def expec_val_monopole
- def expec_val_quadrupole
- def expec_val_

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple sep = (c.getfloat('Params','dx'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple dkx = (c.getfloat('Params','dpx'))
- tuple dt = (c.getfloat('Params','dt'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yMax = (c.getfloat('Params','yMax'))
- tuple num_vort = int(c.getfloat('Params','Num_vort'))
- tuple N = int(c.getfloat('Params','atoms'))
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- tuple x = np.asarray(open('x_0').read().splitlines(),dtype='f8')
- tuple y = np.asarray(open('y_0').read().splitlines(),dtype='f8')
- tuple kx = np.asarray(open('px_0').read().splitlines(),dtype='f8')
- tuple ky = np.asarray(open('py_0').read().splitlines(),dtype='f8')

3.4.1 Function Documentation

3.4.1.1 def observables.ang_mom (*dataName, initValue, finalValue, incr, ev_type, imgdpi*)

Definition at line 170 of file observables.py.

Referenced by expec_val_().

```
00170
00171 def ang_mom(dataName, initValue, finalValue, incr, ev_type, imgdpi):
00172     xm, ym = np.meshgrid(x,y)
00173     pxm, pym = np.meshgrid(px,py)
00174     dx2=dx**2
00175     Lz = np.zeros( (finalValue/incr))
00176     for i in range(initValue,incr*(finalValue/incr),incr):
00177         if os.path.exists(dataName + '_' + str(i)):
00178             real=open(dataName + '_' + str(i)).read().splitlines()
00179             img=open(dataName + 'i_' + str(i)).read().splitlines()
00180             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00181             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00182             a = a_r[:] + 1j*a_i[:]
00183             wfc = np.reshape(a,(xDim,yDim))
00184             conjwfc = np.conj(wfc)
00185
00186             wfc_ypx = np.multiply(ym,np.fft.ifft(np.multiply(pxm,np.fft.fft(wfc,axis=1)),axis=1))
00187             wfc_xpy = np.multiply(xm,np.fft.ifft(np.multiply(pym,np.fft.fft(wfc,axis=0)),axis=0))
00188             result = np.sum( np.sum( np.multiply(conjwfc,wfc_xpy - wfc_ypx) ) )*dx2
00189         else:
00190             print "Skipped " + dataName + "_"+ str(i)
00191             result = np.nan
00192
00193         print i, incr
00194         Lz[(i/incr)] = np.real(result)
00195     type=""
00196     if ev_type == 0:
00197         type = "gnd"
00198     else:
```

```
00199         type = "ev"
00200     np.savetxt(’Lz.csv’,Lz,delimiter=’,’)
00201 
00202     plt.plot(Lz)
00203     plt.savefig("Lz_"+type+".pdf",dpi=imgdpi)
00204     plt.axis(’off’)
00205     plt.savefig("Lz_"+type+"_axis0.pdf",bbox_inches=’tight’,dpi=imgdpi)
00206     plt.close()
```

Here is the caller graph for this function:

3.4.1.2 def observables.dens_struct_fact (*dataName, initValue, finalValue, incr*)

Definition at line 113 of file observables.py.

Referenced by expec_val_().

```
00113 
00114 def dens_struct_fact(dataName, initValue, finalValue,incr):
00115     n_k=np.zeros(finalValue/incr)
00116     n_k_t=np.zeros((finalValue/incr,xDim,yDim),dtype=np.complex128)
00117     for i in range(initValue,incr*(finalValue/incr),incr):
00118         if os.path.exists(dataName + ’_’ + str(i)):
00119             real=open(dataName + ’_’ + str(i)).read().splitlines()
00120             img=open(dataName + ’i_’ + str(i)).read().splitlines()
00121             a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
00122             a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
00123             a = a_r[:] + 1j*a_i[:]
00124             n = np.abs(a)**2
00125             sf = np.fft.fftshift(np.fft.fft2(np.reshape(n,(xDim,yDim))))
00126             n_k_t[i/incr][:][:] = sf[:][:];
00127             n_k[i/incr]=(abs(np.sum(np.sum(sf))*dkx**2))
00128 
00129             fig, ax = plt.subplots()
00130             f = plt.imshow(np.log10(abs(sf)),cmap=plt.get_cmap(’gnuplot2’))
00131             cbar = fig.colorbar(f)
00132             plt.gca().invert_yaxis()
00133             plt.savefig("struct_" + str(i/incr) + ".png",vmin=0,vmax=12,dpi=200)
00134             plt.close()
00135             print i/incr
00136 
00137     np.savetxt(’Struct’ + ’.csv’,n_k,delimiter=’,’)
00138     plt.plot(range(initValue,finalValue,incr),n_k)
00139     sp.io.savemat(’Struct_t.mat’,mdict={’n_k_t’,n_k_t})
00140     plt.savefig("Struct.pdf",dpi=200)
00141     plt.close()
```

Here is the caller graph for this function:

3.4.1.3 def observables.energy_kinetic (*dataName, initValue, finalValue, increment*)

Definition at line 145 of file observables.py.

Referenced by expec_val_().

```
00145 
00146 def energy_kinetic(dataName, initValue, finalValue, increment):
00147     px1 = np.fft.fftshift(px)
00148     py1 = np.fft.fftshift(py)
00149     dk=[]
00150     dk2[:] = (px1[:]**2 + py1[:]**2)
00151     Lz = np.zeros( (finalValue/incr))
00152     for i in range(initValue,incr*(finalValue/incr),incr):
00153         if os.path.exists(dataName + ’_’ + str(i)):
00154             real=open(dataName + ’_’ + str(i)).read().splitlines()
00155             img=open(dataName + ’i_’ + str(i)).read().splitlines()
00156             a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
00157             a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
00158             a = a_r[:] + 1j*a_i[:]
00159             wfcp = np.fft.fft2(np.reshape(a,(xDim,yDim)))
00160             conjwfcp = np.conj(wfcp)
00161             E_k = np.zeros(len(px1))
00162             for ii in range(0,len(px1)):
00163                 E_k[ii] = np.sum( np.sum( np.multiply(wfcp,conjwfcp) )  )*dk2[ii]
```

```
00164
00165         np.savetxt('E_k_' + str(i) + '.csv',E_k,delimiter=',')
00166         print i
```

Here is the caller graph for this function:

3.4.1.4 def observables.energy_potential (*dataName,* *initValue,* *finalValue,* *increment*)

Definition at line 167 of file observables.py.

```
00167
00168 def energy_potential(dataName, initValue, finalValue, increment):
00169     print 'energy'
```

3.4.1.5 def observables.energy_total (*dataName,* *initValue,* *finalValue,* *increment*)

Definition at line 142 of file observables.py.

```
00142
00143 def energy_total(dataName, initValue, finalValue, increment):
00144     print 'energy'
```

3.4.1.6 def observables.expec_val_ (*quant_name,* *quantity,* *dataName,* *initValue,* *finalValue,* *incr*)

Definition at line 259 of file observables.py.

References ang_mom(), dens_struct_fact(), energy_kinetic(), expec_val_monopole(), and expec_val_quadrupole().

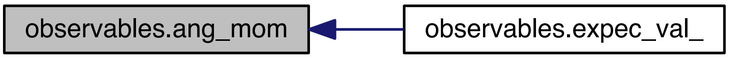
```
00259
00260 def expec_val_(quant_name, quantity, dataName, initValue, finalValue, incr):
00261     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00262     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00263 #    px=open('px_0')
00264 #    py=open('py_0')
00265     xm, ym = np.meshgrid(x, y)
00266     result = []
00267     for i in range(initValue,finalValue,incr):
00268         if not os.path.exists(dataName):
00269             real=open(dataName + '_' + str(i)).read().splitlines()
00270             img=open(dataName + 'i_' + str(i)).read().splitlines()
00271             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00272             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00273             a = a_r[:] + 1j*a_i[:]
00274             wfc = np.reshape(a,(xDim,yDim))
00275             conjwfc = np.conj(wfc)
00276
00277             d1 = np.multiply( quantity, wfc )
00278             d2 = np.multiply( conjwfc, d1)
00279             result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
00280         print str(100*float(i)/finalValue) + '%'
00281     np.savetxt(quant_name + '.csv',result,delimiter=',')
00282     plt.plot(range(initValue,finalValue,incr),result)
00283     plt.savefig(quant_name + ".pdf",dpi=200)
00284     plt.close()
```

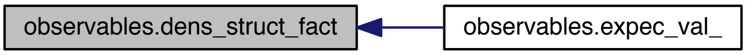
Here is the call graph for this function:

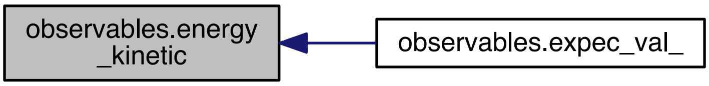
3.4.1.7 def observables.expec_val_monopole (*dataName,* *initValue,* *finalValue,* *incr*)

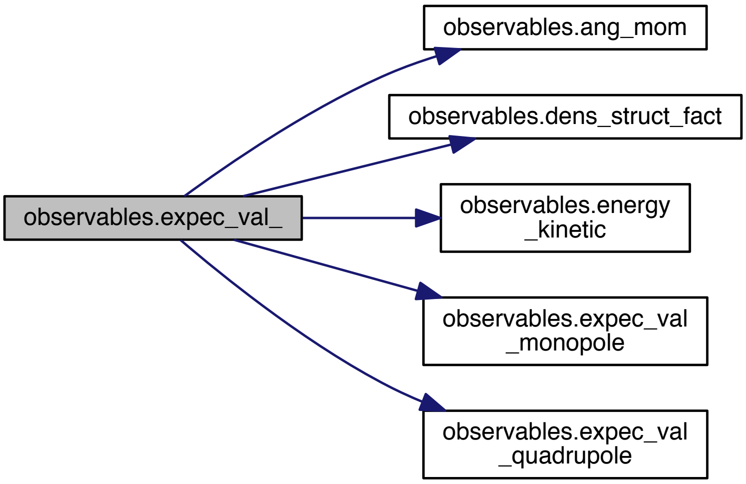
Definition at line 207 of file observables.py.

Referenced by expec_val_().


observables.ang_mom
observables.expec_val_


observables.dens_struct_fact
observables.expec_val_


observables.energy _kinetic
observables.expec_val_


observables.expec_val_
observables.ang_mom
observables.dens_struct_fact
observables.energy _kinetic
observables.expec_val _monopole
observables.expec_val _quadrupole

```
00207
00208 def expec_val_monopole(dataName, initValue, finalValue, incr):
00209     x=np.asarray(open(’x_0’).read().splitlines(),dtype=’f8’)
00210     y=np.asarray(open(’y_0’).read().splitlines(),dtype=’f8’)
00211 #    px=open(’px_0’)
00212 #    py=open(’py_0’)
00213     xm, ym = np.meshgrid(x, y)
00214     result = []
00215     for i in range(initValue,finalValue,incr):
00216         if not os.path.exists(dataName):
00217             real=open(dataName + ’_’ + str(i)).read().splitlines()
00218             img=open(dataName + ’i_’ + str(i)).read().splitlines()
00219             a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
00220             a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
00221             a = a_r[:] + 1j*a_i[:]
00222             wfc = np.reshape(a,(xDim,yDim))
00223             conjwfc = np.conj(wfc)
00224
00225             d1 = np.multiply( np.square(xm) + np.square(ym), wfc )
00226             d2 = np.multiply( conjwfc, d1)
00227             result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
00228         print str(100*float(i)/finalValue) + ’%’
00229     np.savetxt(’monopole.csv’,result,delimiter=’,’)
00230     plt.plot(range(initValue,finalValue,incr),result)
00231     plt.savefig("Monopole.png",dpi=200)
00232     plt.close()
```

Here is the caller graph for this function:

3.4.1.8 def observables.expec_val_quadrupole (*dataName, initValue, finalValue, incr*)

Definition at line 233 of file observables.py.

Referenced by expec_val_().

```
00233
00234 def expec_val_quadrupole(dataName, initValue, finalValue, incr):
00235     x=np.asarray(open(’x_0’).read().splitlines(),dtype=’f8’)
00236     y=np.asarray(open(’y_0’).read().splitlines(),dtype=’f8’)
00237 #    px=open(’px_0’)
00238 #    py=open(’py_0’)
00239     xm, ym = np.meshgrid(x, y)
00240     result = []
00241     for i in range(initValue,finalValue,incr):
00242         if not os.path.exists(dataName):
00243             real=open(dataName + ’_’ + str(i)).read().splitlines()
00244             img=open(dataName + ’i_’ + str(i)).read().splitlines()
00245             a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
00246             a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
00247             a = a_r[:] + 1j*a_i[:]
00248             wfc = np.reshape(a,(xDim,yDim))
00249             conjwfc = np.conj(wfc)
00250
00251             d1 = np.multiply( np.square(xm) - np.square(ym), wfc )
00252             d2 = np.multiply( conjwfc, d1)
00253             result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
00254         print str(100*float(i)/finalValue) + ’%’
00255     np.savetxt(’quadrupole.csv’,result,delimiter=’,’)
00256     plt.plot(range(initValue,finalValue,incr),result)
00257     plt.savefig("Quadrupole.png",dpi=200)
00258     plt.close()
```

Here is the caller graph for this function:

3.4.1.9 def observables.kinertrum (*Psi, dx*)

Definition at line 65 of file observables.py.

```
00065
00066 def kinertrum(Psi, dx):
00067     kxm, kym = np.meshgrid(px,py)
00068     kMax = np.max(np.max(kx))
00069
00070     n_r = np.abs(Psi)**2
```

```
    cPsi = np.conj(Psi)
    phi = np.angle(Psi)

    ph1 = np.unwrap(phi, axis=0)
    ph2 = np.unwrap(phi, axis=1)

    vel_ph1_x, vel_ph1_y = np.gradient(np1,dx,dy)
    vel_ph2_x, vel_ph2_y = np.gradient(np2,dx,dy)

    v_x = (hbar/mass)*vel_ph1_x;
    v_y = (hbar/mass)*vel_ph2_y;

    u_x = np.multiply(np.abs(Psi),v_x)
    u_y = np.multiply(np.abs(Psi),v_y)

    F_x = (1.0/(2*3.14159))*np.fft.fftn(u_x)
    F_y = (1.0/(2*3.14159))*np.fft.fftn(u_y)

    uc_kx = ( np.multiply(np.multiply(kxm,kxm),F_x) + np.multiply(np.multiply(kxm,kym),F_y))
    uc_ky = ( np.multiply(np.multiply(kym,kxm),F_x) + np.multiply(np.multiply(kym,kym),F_y))

    ui_kx = F_x - uc_kx
    ui_ky = F_y - uc_ky

    uc_x = np.fft.ifftn(uc_kx)
    uc_y = np.fft.ifftn(uc_ky)
    ui_x = np.fft.ifftn(ui_kx)
    ui_y = np.fft.ifftn(ui_ky)

    Ec = 0.5*np.real(np.square(uc_x) + np.square(uc_y))
    Ei = 0.5*np.real(np.square(ui_x) + np.square(ui_y))

    k_bins=np.arange(0,max(np.sqrt(kx**2 + ky**2)),np.sqrt(dkx**2 + dky**2))
    num_bins = len(k_bins)

    for i1 in np.arange(0,num_bins-1):
        iX = np.where(k >=k_bins[i1] & k<k_bins[i1+1])
        Ei_kx = np.sum(np.sum(np.abs(ui_kx[iX]**2*k[iX]))
        Ei_ky = np.sum(np.sum(np.abs(ui_ky[iX]**2*k[iX]))
        Ei_k[i1] = (Ei_kx + Ei_ky)/len(iX)
    np.savetxt('Ek_i' + str(i) + '.csv',E_k,delimiter=',')

```

3.4.2 Variable Documentation

3.4.2.1 tuple observables.c = ConfigParser.ConfigParser()

Definition at line 37 of file observables.py.

3.4.2.2 tuple observables.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 57 of file observables.py.

3.4.2.3 tuple observables.dkx = (c.getfloat('Params','dpx'))

Definition at line 47 of file observables.py.

3.4.2.4 tuple observables.dt = (c.getfloat('Params','dt'))

Definition at line 48 of file observables.py.

3.4.2.5 tuple observables.dx = (c.getfloat('Params','dx'))

Definition at line 46 of file observables.py.

3.4.2.6 tuple observables.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 43 of file observables.py.

3.4.2.7 tuple observables.gndMaxVal = int(c.getfloat('Params','gsteps'))

Definition at line 42 of file observables.py.

3.4.2.8 tuple observables.incr = int(c.getfloat('Params','print_out'))

Definition at line 44 of file observables.py.

3.4.2.9 tuple observables.kx = np.asarray(open('px_0').read().splitlines(),dtype='f8')

Definition at line 61 of file observables.py.

3.4.2.10 tuple observables.ky = np.asarray(open('py_0').read().splitlines(),dtype='f8')

Definition at line 62 of file observables.py.

3.4.2.11 tuple observables.N = int(c.getfloat('Params','atoms'))

Definition at line 55 of file observables.py.

Referenced by cMultDensity().

3.4.2.12 tuple observables.num_vort = int(c.getfloat('Params','Num_vort'))

Definition at line 52 of file observables.py.

3.4.2.13 tuple observables.sep = (c.getfloat('Params','dx'))

Definition at line 45 of file observables.py.

3.4.2.14 tuple observables.x = np.asarray(open('x_0').read().splitlines(),dtype='f8')

Definition at line 59 of file observables.py.

3.4.2.15 tuple observables.xDim = int(c.getfloat('Params','xDim'))

Definition at line 40 of file observables.py.

3.4.2.16 tuple observables.xMax = (c.getfloat('Params','xMax'))

Definition at line 49 of file observables.py.

3.4.2.17 tuple observables.y = np.asarray(open('y_0').read().splitlines(),dtype='f8')

Definition at line 60 of file observables.py.

3.4.2.18 tuple observables.yDim = int(c.getfloat('Params','yDim'))

Definition at line 41 of file observables.py.

3.4.2.19 tuple observables.yMax = (c.getfloat('Params','yMax'))

Definition at line 50 of file observables.py.

3.5 overlap Namespace Reference

Functions

- def overlap
- def densitydiff

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple sep = (c.getfloat('Params','dx'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple dt = (c.getfloat('Params','dt'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yMax = (c.getfloat('Params','yMax'))
- tuple num_vort = int(c.getfloat('Params','Num_vort'))
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- tuple real = open("wfc_ev_" + str(0))
- tuple img = open("wfc_evi_" + str(0))
- tuple a_r = numpy.asanyarray(real,dtype='f8')
- tuple a_i = numpy.asanyarray(img,dtype='f8')
- list wfc0 = a_r[:]
- tuple rho0 = abs(np.reshape(wfc0,(xDim,yDim)))
- float norm_coef = 1.0
- list evImgList = [ ]
- list ev_proc = [ ]
- tuple val = evImgList.pop()
- tuple p = ev_proc.pop()

3.5.1 Function Documentation

3.5.1.1 def overlap.densitydiff (*dataName*, *value*, *rho0*)

Definition at line 63 of file overlap.py.

```
00063
00064 def densitydiff(dataName,value,rho0):
00065     real=open(dataName + '_' + str(value)).read().splitlines()
00066     img=open(dataName + 'i_' + str(value)).read().splitlines()
00067     a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00068     a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
```

```
    a = a_r[:] + 1j*a_i[:]
    b = reshape(abs(a)**2,(xDim,yDim))
    c = rho0 - b

    fig, ax = plt.subplots()
    f = plt.imshow(c)
    cbar = fig.colorbar(f)
    #getcontext().prec = 5
    plt.title(’wfc(t=0) - wfc(t=’ + str(value*dt) + ’)’)
    plt.gca().set_xlabel(’x ’+ str((dx)))
    plt.gca().set_ylabel(’y ’+ str(dx))
    plt.gca().invert_yaxis()
    plt.jet()
    plt.savefig(dataName+"r_"+str(value)+"_diff.png",dpi=imgdpi)
    plt.close()
```

3.5.1.2 def overlap.overlap (*dataName,* *value,* *norm_coef*)

Definition at line 53 of file overlap.py.

```
00053
00054 def overlap(dataName,value,norm_coef):
00055     real=open(dataName + ’_’ + str(value)).read().splitlines()
00056     img=open(dataName + ’i_’ + str(value)).read().splitlines()
00057     a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
00058     a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
00059     a = a_r[:] + 1j*a_i[:]
00060     b = np.vdot(wfc0,a)
00061     s = np.sum(b)
00062     print str(value) + ’\t’ +  str(s) + ’\t’ +  str(norm_coef*abs(s)**2)
```

3.5.2 Variable Documentation

3.5.2.1 tuple overlap.a_i = numpy.asanyarray(img,dtype=’f8’)

Definition at line 88 of file overlap.py.

3.5.2.2 tuple overlap.a_r = numpy.asanyarray(real,dtype=’f8’)

Definition at line 87 of file overlap.py.

3.5.2.3 tuple overlap.c = ConfigParser.ConfigParser()

Definition at line 35 of file overlap.py.

3.5.2.4 tuple overlap.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 50 of file overlap.py.

3.5.2.5 tuple overlap.dt = (c.getfloat(’Params’,’dt’))

Definition at line 45 of file overlap.py.

3.5.2.6 tuple overlap.dx = (c.getfloat(’Params’,’dx’))

Definition at line 44 of file overlap.py.

3.5.2.7 list overlap.ev_proc = [ ]

Definition at line 96 of file overlap.py.

3.5.2.8 list overlap.evImgList = [ ]

Definition at line 93 of file overlap.py.

3.5.2.9 tuple overlap.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 41 of file overlap.py.

3.5.2.10 tuple overlap.gndMaxVal = int(c.getfloat('Params','gsteps'))

Definition at line 40 of file overlap.py.

3.5.2.11 tuple overlap.img = open("wfc_evi_" + str(0))

Definition at line 86 of file overlap.py.

3.5.2.12 tuple overlap.incr = int(c.getfloat('Params','print_out'))

Definition at line 42 of file overlap.py.

3.5.2.13 float overlap.norm_coef = 1.0

Definition at line 91 of file overlap.py.

3.5.2.14 tuple overlap.num_vort = int(c.getfloat('Params','Num_vort'))

Definition at line 48 of file overlap.py.

3.5.2.15 tuple overlap.p = ev_proc.pop()

Definition at line 106 of file overlap.py.

Referenced by appendData(), and newParam().

3.5.2.16 tuple overlap.real = open("wfc_ev_" + str(0))

Definition at line 85 of file overlap.py.

3.5.2.17 tuple overlap.rho0 = abs(np.reshape(wfc0,(xDim,yDim)))

Definition at line 90 of file overlap.py.

3.5.2.18 tuple overlap.sep = (c.getfloat('Params','dx'))

Definition at line 43 of file overlap.py.

3.5.2.19 tuple overlap.val = evImgList.pop()

Definition at line 98 of file overlap.py.

3.5.2.20 list overlap.wfc0 = a_r[:]

Definition at line 89 of file overlap.py.

3.5.2.21 tuple overlap.xDim = int(c.getfloat('Params','xDim'))

Definition at line 38 of file overlap.py.

3.5.2.22 tuple overlap.xMax = (c.getfloat('Params','xMax'))

Definition at line 46 of file overlap.py.

3.5.2.23 tuple overlap.yDim = int(c.getfloat('Params','yDim'))

Definition at line 39 of file overlap.py.

3.5.2.24 tuple overlap.yMax = (c.getfloat('Params','yMax'))

Definition at line 47 of file overlap.py.

3.6 py_upload Namespace Reference

Functions

- def get_authenticated_service
- def initialize_upload
- def resumable_upload

Variables

- int MAX_RETRIES = 10
- tuple RETRIABLE_EXCEPTIONS
- list RETRIABLE_STATUS_CODES = [500, 502, 503, 504]
- string CLIENT_SECRETS_FILE = "client_secrets.json"
- string YOUTUBE_UPLOAD_SCOPE = "https://www.googleapis.com/auth/youtube.upload"
- string YOUTUBE_API_SERVICE_NAME = "youtube"
- string YOUTUBE_API_VERSION = "v3"
- string MISSING_CLIENT_SECRETS_MESSAGE
- tuple parser = OptionParser()
- string default = "Test Title"
- string help = "Video description"

3.6.1 Function Documentation

3.6.1.1 def py_upload.get_authenticated_service ( )

Definition at line 70 of file py_upload.py.

Referenced by initialize_upload().

```
00070
00071 def get_authenticated_service():
00072   flow = flow_from_clientsecrets(CLIENT_SECRETS_FILE, scope=YOUTUBE_UPLOAD_SCOPE,
00073     message=MISSING_CLIENT_SECRETS_MESSAGE)
00074
00075   storage = Storage("%s-oauth2.json" % sys.argv[0])
00076   credentials = storage.get()
00077
00078   if credentials is None or credentials.invalid:
00079     credentials = run(flow, storage)
00080
00081   return build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION,
00082     http=credentials.authorize(httplib2.Http()))
00083
```

Here is the caller graph for this function:

3.6.1.2 def py_upload.initialize_upload (*options*)

Definition at line 84 of file py_upload.py.

References get_authenticated_service(), and resumable_upload().

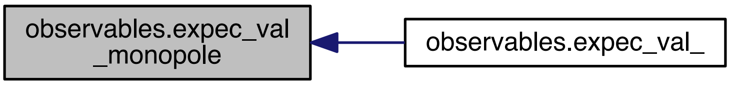
```
00084
00085 def initialize_upload(options):
00086   youtube = get_authenticated_service()
00087
00088   tags = None
00089   if options.keywords:
00090     tags = options.keywords.split(",")
00091
00092   insert_request = youtube.videos().insert(
00093     part="snippet,status",
00094     body=dict(
00095       snippet=dict(
00096         title=options.title,
00097         description=options.description,
00098         tags=tags,
00099         categoryId=options.category
00100       ),
00101       status=dict(
00102         privacyStatus=options.privacyStatus
00103       )
00104     ),
00105     # chunksize=-1 means that the entire file will be uploaded in a single
00106     # HTTP request. (If the upload fails, it will still be retried where it
00107     # left off.) This is usually a best practice, but if you’re using Python
00108     # older than 2.6 or if you’re running on App Engine, you should set the
00109     # chunksize to something like 1024 * 1024 (1 megabyte).
00110     media_body=MediaFileUpload(options.file, chunksize=-1, resumable=True)
00111   )
00112
00113   resumable_upload(insert_request)
00114
```

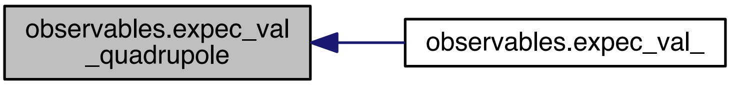
Here is the call graph for this function:

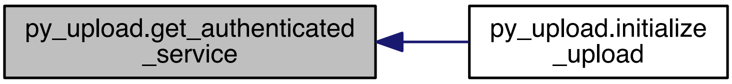
3.6.1.3 def py_upload.resumable_upload (*insert_request*)

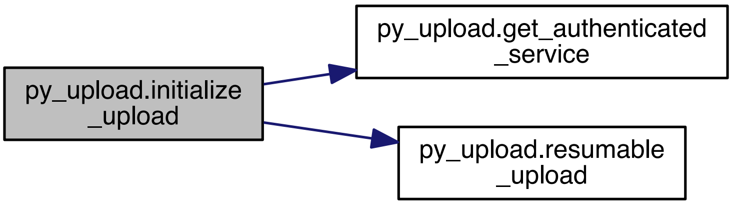
Definition at line 115 of file py_upload.py.

Referenced by initialize_upload().


observables.expec_val
_monopole
observables.expec_val_


observables.expec_val
_quadrupole
observables.expec_val_


py_upload.get_authenticated
_service
py_upload.initialize
_upload


py_upload.initialize
_upload
py_upload.get_authenticated
_service
py_upload.resumable
_upload

```
def resumable_upload(insert_request):
  response = None
  error = None
  retry = 0
  while response is None:
    try:
      print "Uploading file..."
      status, response = insert_request.next_chunk()
      if ’id’ in response:
        print "’%s’ (video id: %s) was successfully uploaded." % (
          options.title, response[’id’])
      else:
        exit("The upload failed with an unexpected response: %s" % response)
    except HttpError, e:
      if e.resp.status in RETRIABLE_STATUS_CODES:
        error = "A retriable HTTP error %d occurred:\n%s" % (e.resp.status,
                                                             e.content)
      else:
        raise
    except RETRIABLE_EXCEPTIONS, e:
      error = "A retriable error occurred: %s" % e

    if error is not None:
      print error
      retry += 1
      if retry > MAX_RETRIES:
        exit("No longer attempting to retry.")

      max_sleep = 2 ** retry
      sleep_seconds = random.random() * max_sleep
      print "Sleeping %f seconds and then retrying..." % sleep_seconds
      time.sleep(sleep_seconds)

```

Here is the caller graph for this function:

3.6.2 Variable Documentation

3.6.2.1 string py_upload.CLIENT_SECRETS_FILE = "client_secrets.json"

Definition at line 45 of file py_upload.py.

3.6.2.2 string py_upload.default = "Test Title"

Definition at line 153 of file py_upload.py.

3.6.2.3 string py_upload.help = "Video description"

Definition at line 155 of file py_upload.py.

3.6.2.4 int py_upload.MAX_RETRIES = 10

Definition at line 24 of file py_upload.py.

3.6.2.5 string py_upload.MISSING_CLIENT_SECRETS_MESSAGE

Initial value:

```
= """
WARNING: Please configure OAuth 2.0

To make this sample run you will need to populate the client_secrets.json file
found at:

   %s

```

```
with information from the APIs Console
https://code.google.com/apis/console#access

For more information about the client_secrets.json file format, please visit:
https://developers.google.com/api-client-library/python/guide/aaa_client_secrets
"""
```

Definition at line 54 of file py_upload.py.

3.6.2.6 tuple py_upload.parser = OptionParser()

Definition at line 150 of file py_upload.py.

3.6.2.7 tuple py_upload.RETRIABLE_EXCEPTIONS

Initial value:

```
= (httplib2.HttpLib2Error, IOError, httplib.NotConnected,
  httplib.IncompleteRead, httplib.ImproperConnectionState,
  httplib.CannotSendRequest, httplib.CannotSendHeader,
  httplib.ResponseNotReady, httplib.BadStatusLine)
```

Definition at line 27 of file py_upload.py.

3.6.2.8 list py_upload.RETRIABLE_STATUS_CODES = [500, 502, 503, 504]

Definition at line 34 of file py_upload.py.

3.6.2.9 string py_upload.YOUTUBE_API_SERVICE_NAME = "youtube"

Definition at line 50 of file py_upload.py.

3.6.2.10 string py_upload.YOUTUBE_API_VERSION = "v3"

Definition at line 51 of file py_upload.py.

3.6.2.11 string py_upload.YOUTUBE_UPLOAD_SCOPE = "https://www.googleapis.com/auth/youtube.upload"

Definition at line 49 of file py_upload.py.

3.7 run Namespace Reference

3.8 stats Namespace Reference

Functions

- def IsFit

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple yDim = int(c.getfloat('Params','yDim'))

3.8.1 Function Documentation

3.8.1.1 def stats.lsFit (*start*, *end*, *incr*)

Definition at line 42 of file stats.py.

```
00042
00043 def lsFit(start,end,incr):
00044     L = np.matrix([
00045             [0,0,1],
00046             [1,0,1],
00047             [0,1,1],
00048             [1,1,1]
00049             ])
00050     LSQ = np.linalg.inv(np.transpose(L)*L)*np.transpose(L)
00051     for i in range(start,end,incr):
00052         v_arr=genfromtxt(’vort_arr_’ + str(i),delimiter=’,’ )
00053         real=open(’wfc_ev_’ + str(i)).read().splitlines()
00054         img=open(’wfc_evi_’ + str(i)).read().splitlines()
00055         a_r = np.asanyarray(real,dtype=’f8’) #64-bit double
00056         a_i = np.asanyarray(img,dtype=’f8’) #64-bit double
00057         a = a_r[:] + 1j*a_i[:]
00058         wfc = (np.reshape(a,(xDim,yDim)))
00059
00060         indX = [row[0] for row in v_arr]
00061         indY = [row[1] for row in v_arr]
00062         wind = [row[2] for row in v_arr]
00063         sign = [row[3] for row in v_arr]
00064         data=[]
00065         for ii in range(0,len(indX)):
00066             p=np.matrix([[0],[0],[0],[0]],dtype=np.complex)
00067             p[0]=(wfc[indX[ii], indY[ii]])
00068             p[1]=(wfc[indX[ii]+1, indY[ii]])
00069             p[2]=(wfc[indX[ii], indY[ii]+1])
00070             p[3]=(wfc[indX[ii]+1, indY[ii]+1])
00071             rc = LSQ * np.real(p)
00072             ic = LSQ * np.imag(p)
00073
00074             A=np.squeeze([row[0:2] for row in [rc,ic]])
00075             B=-np.squeeze([row[2] for row in [rc,ic]])
00076             r=np.linalg.lstsq(A,B)[0]
00077             data.append([indX[ii]+r[0],indY[ii]+r[1],sign[ii]])
00078
00079 #       f = plt.imshow(abs(wfc)**2)
00080 #       plt.jet()
00081 #       plt.gca().invert_yaxis()
00082 #       plt.hold(True)
00083 #       X = [row[0] for row in data]
00084 #       Y = [row[1] for row in data]
00085 #       plt.scatter(Y,X,s=0.2,marker=’.’,c=’red’,lw=0)
00086 #       plt.scatter(indY,indX,s=0.2,marker=’.’,c=’yellow’,lw=0)
00087 #       plt.savefig("fig.png",dpi=1200)
00088 #       plt.close()
00089         np.savetxt(’vort_lsq_’+str(i)+’.csv’,data,delimiter=’,’)
```

3.8.2 Variable Documentation

3.8.2.1 tuple stats.c = ConfigParser.ConfigParser()

Definition at line 35 of file stats.py.

3.8.2.2 tuple stats.incr = int(c.getfloat(’Params’,’print_out’))

Definition at line 38 of file stats.py.

3.8.2.3 tuple stats.xDim = int(c.getfloat(’Params’,’xDim’))

Definition at line 39 of file stats.py.

3.8.2.4 tuple stats.yDim = int(c.getfloat('Params','yDim'))

Definition at line 40 of file stats.py.

3.9 track Namespace Reference

Variables

- tuple img = cv.LoadImage("foo2.jpg",cv.CV_LOAD_IMAGE_GRAYSCALE)
- tuple eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
- tuple temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

3.9.1 Variable Documentation

3.9.1.1 tuple track.eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

Definition at line 3 of file track.py.

3.9.1.2 tuple track.img = cv.LoadImage("foo2.jpg",cv.CV_LOAD_IMAGE_GRAYSCALE)

Definition at line 2 of file track.py.

3.9.1.3 tuple track.temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

Definition at line 4 of file track.py.

3.10 track_circles Namespace Reference

Variables

- tuple img = cv.LoadImage("wfc_1000.png",cv.CV_LOAD_IMAGE_GRAYSCALE)
- tuple eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
- tuple temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
- tuple circles = cv.CreateMat(img.width,1,cv.CV_32FC3)
- tuple c = numpy.asarray(circles)

3.10.1 Variable Documentation

3.10.1.1 tuple track_circles.c = numpy.asarray(circles)

Definition at line 8 of file track_circles.py.

3.10.1.2 tuple track_circles.circles = cv.CreateMat(img.width,1,cv.CV_32FC3)

Definition at line 6 of file track_circles.py.

3.10.1.3 tuple track_circles.eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

Definition at line 3 of file track_circles.py.

3.10.1.4 tuple track_circles.img = cv.LoadImage("wfc_1000.png",cv.CV_LOAD_IMAGE_GRAYSCALE)

Definition at line 2 of file track_circles.py.

3.10.1.5 tuple track_circles.temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

Definition at line 4 of file track_circles.py.

3.11 vis Namespace Reference

Functions

- def delaunay
- def voronoi
- def laplacian
- def struct_fact
- def opPot
- def hist_gen
- def image_gen
- def image_gen_single
- def vort_traj
- def scaleAxis
- def overlap

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple sep = (c.getfloat('Params','dx'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple dt = (c.getfloat('Params','dt'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yMax = (c.getfloat('Params','yMax'))
- tuple num_vort = int(c.getfloat('Params','Num_vort'))
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- list gndImgList = [ ]
- list evImgList = [ ]
- list gnd_proc = [ ]
- list ev_proc = [ ]
- tuple i = gndImgList.pop()
- proc = gnd_proc+ev_proc
- tuple p = proc.pop()

3.11.1 Function Documentation

3.11.1.1 def vis.delaunay (*dataName*, *dataType*, *value*)

Definition at line 57 of file vis.py.

```
00057
00058 def delaunay(dataName,dataType,value):
00059     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=’,’ )
00060     data = np.array([[row[0],row[1]] for row in v_arr])
00061     dln = sp.spatial.Delaunay(data)
00062     plt.triplot(data[:,0],data[:,1],dln.simplices.copy(),linewidth=0.5,color=’b’,marker=’.’)
00063     plt.xlim(300,700);plt.ylim(300,700);
00064     plt.savefig(’delaun_’ + str(value) + ’.png’,dpi=200)
00065     print ’Saved Delaunay @ t=’ + str(value)
```

3.11.1.2 def vis.hist_gen (*name*, *value*, *num_bins*)

Definition at line 112 of file vis.py.

```
00112
00113 def hist_gen(name,value,num_bins):
00114     v_arr=genfromtxt(’vort_arr_’ + str(value),delimiter=’,’ )
00115     H=[]
00116     count=0
00117
00118     for i1 in range(0,v_arr.size/2):
00119         for i2 in range(i1,v_arr.size/2):
00120             H.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2  +  abs(v_arr[i1][1]*sep - v_arr[
      i2][1]*sep)**2 ))
00121             count = count + 1
00122     plt.title(’Vortex lattice @ t=’ + str(value*dt))
00123     plt.ticklabel_format(style=’scientific’)
00124     plt.ticklabel_format(style=’scientific’,axis=’x’, scilimits=(0,0))
00125     h = plt.hist(H, bins=num_bins)
00126     plt.savefig(name + "_" + str(value) + ".pdf")
00127     plt.close()
```

3.11.1.3 def vis.image_gen (*dataName*, *initValue*, *finalValue*, *increment*, *imgdpi*)

Definition at line 128 of file vis.py.

```
00128
00129 def image_gen(dataName, initValue, finalValue, increment,imgdpi):
00130     for i in range(initValue,finalValue,increment):
00131         if not os.path.exists(dataName+"r_"+str(i)+"_abspsi2.png"):
00132             real=open(dataName + ’_’ + str(i)).read().splitlines()
00133             img=open(dataName + ’i_’ + str(i)).read().splitlines()
00134             a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
00135             a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
00136             a = a_r[:] + 1j*a_i[:]
00137             b = np.reshape(a,(xDim,yDim))
00138             f = plt.imshow(abs(b)**2)
00139             plt.jet()
00140             plt.gca().invert_yaxis()
00141             plt.savefig(dataName+"r_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00142             plt.close()
00143             g = plt.imshow(np.angle(b))
00144             plt.gca().invert_yaxis()
00145             plt.savefig(dataName+"r_"+str(i)+"_phi.png",dpi=imgdpi)
00146             plt.close()
00147             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))**2)
00148             plt.gca().invert_yaxis()
00149             plt.jet()
00150             plt.savefig(dataName+"p_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00151             plt.close()
00152             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00153             plt.gca().invert_yaxis()
00154             plt.savefig(dataName+"p_"+str(i)+"_phi.png",dpi=imgdpi)
00155             plt.close()
00156             print "Saved figure: " + str(i) + ".png"
00157             plt.close()
```

```
00158         else:
00159             print "File(s) " + str(i) +".png already exist."
```

3.11.1.4 def vis.image_gen_single (*dataName*, *value*, *imgdpi*, *opmode*)

Definition at line 160 of file vis.py.

References laplacian(), and struct_fact().

```
00160
00161 def image_gen_single(dataName, value, imgdpi,opmode):
00162     real=open(dataName + ’_’ + str(0)).read().splitlines()
00163     img=open(dataName + ’i_’ + str(0)).read().splitlines()
00164     a1_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
00165     a1_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
00166     a1 = a1_r[:] + 1j*a1_i[:]
00167     b1 = np.reshape(a1,(xDim,yDim))
00168
00169     if not os.path.exists(dataName+"r_"+str(value)+"_abspsi2.png"):
00170         real=open(dataName + ’_’ + str(value)).read().splitlines()
00171         img=open(dataName + ’i_’ + str(value)).read().splitlines()
00172         a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
00173         a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
00174         a = a_r[:] + 1j*a_i[:]
00175         b = np.reshape(a,(xDim,yDim))
00176
00177         #scaleAxis(b,dataName,"_abspsi2",value,imgdpi)
00178         if opmode & 0b100000 > 0:
00179             fig, ax = plt.subplots()
00180             #plt.rc(’text’,usetex=True)
00181             #plt.rc(’font’,family=’serif’)
00182             f = plt.imshow((abs(b)**2 - abs(b1)**2),cmap=’gnuplot2’,vmin=-6,vmax=6)
00183             plt.title(r’$\left(\rho( r,t ) - \rho( r,t_0 )\right),t=$’ + str(value*dt))
00184             cbar = fig.colorbar(f)
00185             plt.gca().set_xlabel(’x ’+ str((dx)))
00186             plt.gca().set_ylabel(’x ’+ str(dx))
00187             plt.gca().invert_yaxis()
00188             plt.savefig(dataName+"r_"+str(value)+"_diffabspsi2.png",dpi=imgdpi)
00189             plt.close()
00190             #plt.rc(’text’,usetex=True)
00191             #plt.rc(’font’,family=’serif’)
00192             fig, ax = plt.subplots()
00193             f = plt.imshow((abs(b)**2),cmap=’gnuplot2’,vmin=0,vmax=8)
00194             plt.title(’rho(r) @ t=’ + str(value*dt))
00195             plt.title(r’$\log_{10}\rho \left( r,t \right),\,t=$’ + str(value*dt))
00196
00197             cbar = fig.colorbar(f)
00198             plt.gca().set_xlabel(’x ’+ str((dx)))
00199             plt.gca().set_ylabel(’x ’+ str(dx))
00200             plt.gca().invert_yaxis()
00201             plt.savefig(dataName+"r_"+str(value)+"_abspsi2.png",dpi=imgdpi)
00202             plt.axis(’off’)
00203             plt.savefig(dataName+"r_"+str(value)+"_abspsi2_axis0.pdf",bbox_inches=’tight’,dpi=imgdpi)
00204             plt.close()
00205
00206         if opmode & 0b010000 > 0:
00207             fig, ax = plt.subplots()
00208             g = plt.imshow(np.angle(b))
00209             cbar = fig.colorbar(g)
00210             plt.gca().invert_yaxis()
00211             plt.title(’theta(r) @ t=’ + str(value*dt))
00212             plt.savefig(dataName+"r_"+str(value)+"_phi.png",dpi=imgdpi)
00213             plt.close()
00214
00215         if opmode & 0b001000 > 0:
00216             fig, ax = plt.subplots()
00217             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))**2)
00218             cbar = fig.colorbar(f)
00219             plt.gca().invert_yaxis()
00220             plt.jet()
00221             plt.title(’rho(p) @ t=’ + str(value*dt))
00222             plt.savefig(dataName+"p_"+str(value)+"_abspsi2.png",dpi=imgdpi)
00223             plt.close()
00224
00225         if opmode & 0b000100 > 0:
00226             fig, ax = plt.subplots()
00227             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00228             cbar = fig.colorbar(g)
00229             plt.gca().invert_yaxis()
00230             plt.title(’theta(p) @ t=’ + str(value*dt))
00231             plt.savefig(dataName+"p_"+str(value)+"_phi.png",dpi=imgdpi)
```

```
00232             plt.close()
00233 
00234         if opmode & 0b000010 > 0:
00235             struct_fact(abs(b)**2,dataName+"_" + str(value),imgdpi)
00236 
00237         if opmode & 0b000001 > 0:
00238             laplacian(abs(b)**2,dataName+"_" + str(value),imgdpi)
00239 
00240         print "Saved figure: " + str(value) + ".png"
00241         plt.close()
00242     else:
00243         print "File(s) " + str(value) +".png already exist."
```

Here is the call graph for this function:

3.11.1.5 def vis.laplacian (*density*, *name*, *imgdpi*)

Definition at line 75 of file vis.py.

Referenced by image_gen_single().

```
00075 
00076 def laplacian(density,name,imgdpi):
00077     gx,gy = np.gradient(density)
00078     g2x,gxgy = np.gradient(gx)
00079     gygx,g2y = np.gradient(gy)
00080     fig, ax = plt.subplots()
00081     #f = plt.quiver(gx,gy)
00082     f = plt.imshow((g2x**2 + g2y**2),cmap=plt.get_cmap('spectral'))
00083     cbar = fig.colorbar(f)
00084     plt.savefig(name + "_laplacian.png",dpi=imgdpi)
00085     plt.close()
00086     f = plt.imshow((gxgy - gygx),cmap=plt.get_cmap('spectral'))
00087     cbar = fig.colorbar(f)
00088     plt.savefig(name + "_dxdy.png",dpi=imgdpi)
00089     plt.close()
```

Here is the caller graph for this function:

3.11.1.6 def vis.opPot (*dataName*, *imgdpi*)

Definition at line 100 of file vis.py.

Referenced by overlap().

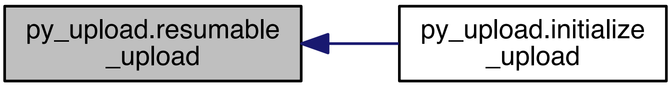
```
00100 
00101 def opPot(dataName,imgdpi):
00102     data = open(dataName).read().splitlines()
00103     a = numpy.asanyarray(data,dtype='f8')
00104     b = np.reshape(a,(xDim,yDim))
00105     fig, ax = plt.subplots()
00106     f = plt.imshow((b))
00107     plt.gca().invert_yaxis()
00108     cbar = fig.colorbar(f)
00109     plt.jet()
00110     plt.savefig(dataName + ".png",dpi=imgdpi)
00111     plt.close()
```

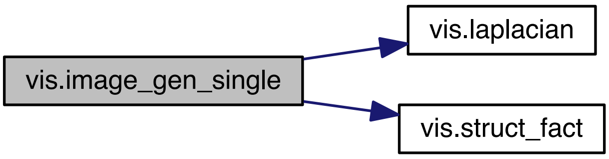
Here is the caller graph for this function:

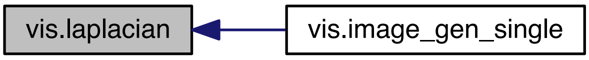
3.11.1.7 def vis.overlap (*dataName*, *initValue*, *finalValue*, *increment*)

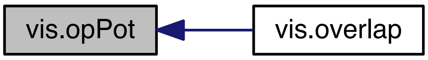
Definition at line 285 of file vis.py.

References opPot().


py_upload.resumable
_upload
py_upload.initialize
_upload


vis.image_gen_single
vis.laplacian
vis.struct_fact


vis.laplacian
vis.image_gen_single


vis.opPot
vis.overlap

```
00285
00286 def overlap(dataName, initValue, finalValue, increment):
00287     real=open(dataName + ’_’ + str(0)).read().splitlines()
00288     img=open(dataName + ’i_’ + str(0)).read().splitlines()
00289     a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
00290     a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
00291     wfc0 = a_r[:] + 1j*a_i[:]
00292     for i in range(initValue,finalValue,increment):
00293         real=open(dataName + ’_’ + str(value)).read().splitlines()
00294         img=open(dataName + ’i_’ + str(value)).read().splitlines()
00295         a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
00296         a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
00297         a = a_r[:] + 1j*a_i[:]
00298         b = np.dot(wfc0,a)
00299         print i, np.sum(b)
```

Here is the call graph for this function:

3.11.1.8 def vis.scaleAxis (*data, dataName, label, value, imgdpi*)

Definition at line 274 of file vis.py.

```
00274
00275 def scaleAxis(data,dataName,label,value,imgdpi):
00276     fig, ax = plt.subplots()
00277     ax.xaxis.set_major_locator(ScaledLocator(dx=dx))
00278     ax.xaxis.set_major_formatter(ScaledLocator(dx=dx))
00279     f = plt.imshow(abs(data)**2)
00280     cbar = fig.colorbar(f)
00281     plt.gca().invert_yaxis()
00282     plt.jet()
00283     plt.savefig(dataName+"r_"+str(value)+"_"+label +".png",dpi=imgdpi)
00284     plt.close()
```

3.11.1.9 def vis.struct_fact (*density, name, imgdpi*)

Definition at line 90 of file vis.py.

Referenced by image_gen_single().

```
00090
00091 def struct_fact(density,name,imgdpi):
00092     fig, ax = plt.subplots()
00093     #f = plt.quiver(gx,gy)
00094     f = plt.imshow((np.abs(np.fft.fftshift(np.fft.fft2(density)))),cmap=plt.get_cmap(’prism’))
00095     cbar = fig.colorbar(f)
00096     cbar.set_clim(1e6,1e11)
00097     plt.jet()
00098     plt.savefig(name + "_struct_log10.png",dpi=imgdpi)
00099     plt.close()
```

Here is the caller graph for this function:

3.11.1.10 def vis.voronoi (*dataName, dataType, value*)

Definition at line 66 of file vis.py.

```
00066
00067 def voronoi(dataName,dataType,value):
00068     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=’,’ )
00069     data = [[row[0],row[1]] for row in v_arr]
00070     vor = Voronoi(data)
00071     voronoi_plot_2d(vor)
00072     plt.xlim(300,700);plt.ylim(300,700);
00073     plt.savefig(’voronoi_’ + str(value) + ’.png’,dpi=200)
00074     print ’Saved Voronoi @ t=’ + str(value)
```

3.11.1.11 def vis.vort_traj (*name*, *imgdpi*)

Definition at line 244 of file vis.py.

```
00244
00245 def vort_traj(name,imgdpi):
00246     evMaxVal_l = evMaxVal
00247     H=genfromtxt(’vort_arr_0’,delimiter=’,’ )
00248     count=0
00249     for i1 in range(incr,evMaxVal_l,incr):
00250         try:
00251             v_arr=genfromtxt(’vort_lsq_’ + str(i1) + ’.csv’,delimiter=’,’ )
00252             H=np.column_stack((H,v_arr))
00253         except:
00254             evMaxVal_l = i1
00255             break
00256     X=np.zeros((evMaxVal_l/incr),dtype=np.float64)
00257     Y=np.zeros((evMaxVal_l/incr),dtype=np.float64)
00258     H=np.reshape(H,([num_vort,2,evMaxVal_l/incr]),order=’F’)
00259     for i1 in range(0, num_vort):
00260         for i2 in range(0,evMaxVal_l/incr):
00261             X[i2]=(H[i1,0,i2]*dx) - xMax
00262             Y[i2]=(H[i1,1,i2]*dx) - yMax
00263         h = plt.plot(X,Y,color=(r.random(),r.random(),r.random(),0.85),linewidth=0.1)
00264     plt.axis(’equal’)
00265     plt.title(’Vort(x,y) from t=0 to t=’+str(evMaxVal_l*dt)+" s")
00266
00267     plt.axis((-xMax/2.0, xMax/2.0, -yMax/2.0, yMax/2.0))
00268     plt.ticklabel_format(style=’scientific’)
00269     plt.ticklabel_format(style=’scientific’,axis=’x’, scilimits=(0,0))
00270     plt.ticklabel_format(style=’scientific’,axis=’y’, scilimits=(0,0))
00271     plt.savefig(name +".pdf")
00272     plt.close()
00273     print "Trajectories plotted."
```

3.11.2 Variable Documentation

3.11.2.1 tuple vis.c = ConfigParser.ConfigParser()

Definition at line 40 of file vis.py.

3.11.2.2 tuple vis.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 55 of file vis.py.

3.11.2.3 tuple vis.dt = (c.getfloat(’Params’,’dt’))

Definition at line 50 of file vis.py.

3.11.2.4 tuple vis.dx = (c.getfloat(’Params’,’dx’))

Definition at line 49 of file vis.py.

3.11.2.5 list vis.ev_proc = [ ]

Definition at line 318 of file vis.py.

3.11.2.6 list vis.evImgList = [ ]

Definition at line 312 of file vis.py.

3.11.2.7 tuple vis.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 46 of file vis.py.

3.11.2.8 list vis.gnd_proc = [ ]

Definition at line 317 of file vis.py.

3.11.2.9 list vis.gndImgList = [ ]

Definition at line 311 of file vis.py.

3.11.2.10 tuple vis.gndMaxVal = int(c.getfloat('Params','gsteps'))

Definition at line 45 of file vis.py.

3.11.2.11 tuple vis.i = gndImgList.pop()

Definition at line 320 of file vis.py.

Referenced by delta_define(), energy_angmom(), evolve(), findOLMaxima(), findVortex(), initialise(), main(), maxValue(), minValue(), multipass(), olPos(), optLatSetup(), pSum(), pSumT(), readIn(), scalVecMult_d2d(), scalVecMult_d2d2(), scalVecMult_dd(), scalVecMult_ii(), sepAvg(), sigVOL(), sumAvg(), sumVector_d(), sumVector_d2(), vecVecMult_d2d(), vecVecMult_d2d2(), vecVecMult_dd(), vecVecMult_ii(), vortAngle(), vortArrange(), vortCentre(), vortPos(), writeOut(), writeOutDouble(), writeOutInt(), writeOutInt2(), writeOutParam(), and writeOutVortex().

3.11.2.12 tuple vis.incr = int(c.getfloat('Params','print_out'))

Definition at line 47 of file vis.py.

3.11.2.13 tuple vis.num_vort = int(c.getfloat('Params','Num_vort'))

Definition at line 53 of file vis.py.

3.11.2.14 tuple vis.p = proc.pop()

Definition at line 333 of file vis.py.

3.11.2.15 vis.proc = gnd_proc+ev_proc

Definition at line 329 of file vis.py.

3.11.2.16 tuple vis.sep = (c.getfloat('Params','dx'))

Definition at line 48 of file vis.py.

3.11.2.17 tuple vis.xDim = int(c.getfloat('Params','xDim'))

Definition at line 43 of file vis.py.

3.11.2.18 tuple vis.xMax = (c.getfloat('Params','xMax'))

Definition at line 51 of file vis.py.

3.11.2.19 tuple vis.yDim = int(c.getfloat('Params','yDim'))

Definition at line 44 of file vis.py.

3.11.2.20 tuple vis.yMax = (c.getfloat('Params','yMax'))

Definition at line 52 of file vis.py.

3.12 vis_ev Namespace Reference

Variables

- int xDim = 256
- int yDim = 256
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- string s = "./wfc"
- tuple real = open(s + '_' + str(i))
- tuple img = open(s + 'i_' + str(i))
- tuple a_r = numpy.asanyarray(real,dtype='f8')
- tuple a_i = numpy.asanyarray(img,dtype='f8')
- list a = a_r[:]
- tuple b = np.reshape(a,(xDim,yDim))
- tuple f = plt.imshow(abs(b)∗∗2)

3.12.1 Variable Documentation

3.12.1.1 list vis_ev.a = a_r[:]

Definition at line 33 of file vis_ev.py.

3.12.1.2 tuple vis_ev.a_i = numpy.asanyarray(img,dtype='f8')

Definition at line 32 of file vis_ev.py.

3.12.1.3 tuple vis_ev.a_r = numpy.asanyarray(real,dtype='f8')

Definition at line 31 of file vis_ev.py.

3.12.1.4 tuple vis_ev.b = np.reshape(a,(xDim,yDim))

Definition at line 34 of file vis_ev.py.

Referenced by initialise().

3.12.1.5 tuple vis_ev.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 25 of file vis_ev.py.

3.12.1.6 tuple vis_ev.f = plt.imshow(abs(b)∗∗2)

Definition at line 35 of file vis_ev.py.

Referenced by readIn(), readState(), writeOut(), writeOutDouble(), writeOutInt(), writeOutInt2(), writeOutParam(), and writeOutVortex().

3.12.1.7 tuple vis_ev.img = open(s + 'i_' + str(i))

Definition at line 30 of file vis_ev.py.

3.12.1.8 tuple vis_ev.real = open(s + '_' + str(i))

Definition at line 29 of file vis_ev.py.

3.12.1.9 string vis_ev.s = "./wfc"

Definition at line 26 of file vis_ev.py.

3.12.1.10 int vis_ev.xDim = 256

Definition at line 23 of file vis_ev.py.

3.12.1.11 int vis_ev.yDim = 256

Definition at line 24 of file vis_ev.py.

3.13 visual_ev Namespace Reference

Variables

- int xDim = 256
- int yDim = 256
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- string s = "./wfc"
- tuple real = open(s + '_' + str(i))
- tuple img = open(s + 'i_' + str(i))
- tuple a_r = numpy.asanyarray(real,dtype='f8')
- tuple a_i = numpy.asanyarray(img,dtype='f8')
- list a = a_r[:]
- tuple b = numpy.reshape(a,(xDim,yDim))

3.13.1 Variable Documentation

3.13.1.1 list visual_ev.a = a_r[:]

Definition at line 33 of file visual_ev.py.

3.13.1.2 tuple visual_ev.a_i = numpy.asanyarray(img,dtype='f8')

Definition at line 32 of file visual_ev.py.

3.13.1.3 tuple visual_ev.a_r = numpy.asanyarray(real,dtype='f8')

Definition at line 31 of file visual_ev.py.

3.13.1.4 tuple visual_ev.b = numpy.reshape(a,(xDim,yDim))

Definition at line 34 of file visual_ev.py.

3.13.1.5 tuple visual_ev.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 25 of file visual_ev.py.

3.13.1.6 tuple visual_ev.img = open(s + 'i_' + str(i))

Definition at line 30 of file visual_ev.py.

3.13.1.7 tuple visual_ev.real = open(s + '_' + str(i))

Definition at line 29 of file visual_ev.py.

3.13.1.8 string visual_ev.s = "./wfc"

Definition at line 26 of file visual_ev.py.

3.13.1.9 int visual_ev.xDim = 256

Definition at line 23 of file visual_ev.py.

3.13.1.10 int visual_ev.yDim = 256

Definition at line 24 of file visual_ev.py.

3.14 visual_gnd Namespace Reference

Variables

- int xDim = 256
- int yDim = 256
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- string s = "./wfc_0"
- tuple real = open(s + '_' + str(i))
- tuple img = open(s + 'i_' + str(i))
- tuple a_r = numpy.asanyarray(real,dtype='f8')
- tuple a_i = numpy.asanyarray(img,dtype='f8')
- list a = a_r[:]
- tuple b = numpy.reshape(a,(xDim,yDim))

3.14.1 Variable Documentation

3.14.1.1 list visual_gnd.a = a_r[:]

Definition at line 32 of file visual_gnd.py.

3.14.1.2 tuple visual_gnd.a_i = numpy.asanyarray(img,dtype='f8')

Definition at line 31 of file visual_gnd.py.

3.14.1.3 tuple visual_gnd.a_r = numpy.asanyarray(real,dtype='f8')

Definition at line 30 of file visual_gnd.py.

3.14.1.4 tuple visual_gnd.b = numpy.reshape(a,(xDim,yDim))

Definition at line 33 of file visual_gnd.py.

3.14.1.5 tuple visual_gnd.data = numpy.ndarray(shape=(xDim,yDim))

Definition at line 24 of file visual_gnd.py.

3.14.1.6 tuple visual_gnd.img = open(s + 'i_' + str(i))

Definition at line 29 of file visual_gnd.py.

3.14.1.7 tuple visual_gnd.real = open(s + '_' + str(i))

Definition at line 28 of file visual_gnd.py.

3.14.1.8 string visual_gnd.s = "./wfc_0"

Definition at line 25 of file visual_gnd.py.

3.14.1.9 int visual_gnd.xDim = 256

Definition at line 22 of file visual_gnd.py.

3.14.1.10 int visual_gnd.yDim = 256

Definition at line 23 of file visual_gnd.py.

3.15 vort Namespace Reference

Classes

- class Vortex
- class VtxList

Functions

- def __init__
- def update_uid
- def update_on
- def update_next
- def dist
- def __init__
- def element
- def vtx_uid
- def max_uid
- def add
- def as_np
- def write_out
- def idx_min_dist
- def remove
- def swap_uid
- def vort_decrease
- def vort_increase
- def do_the_thing

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple dt = (c.getfloat('Params','dt'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yMax = (c.getfloat('Params','yMax'))
- tuple r = m.sqrt((self.x - vtx.x)∗∗2 + (self.y - vtx.y)∗∗2)
- int pos_l = 0
- vtx = self.head
- int pos = 0
- int val = 0
- list dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]
- list data = [ ]
- int i = 0
- int counter = 0
- ret_idx = counter
- tuple current = self.element(pos-1)
- tuple vtx_pos = self.vtx_uid(uid_i)
- tuple max_uid = vorts_p.max_uid()
- tuple v_arr_p = genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')

  v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')
- tuple vorts_p = VtxList()
- tuple vorts_c = VtxList()
- tuple v_arr_c = genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
- tuple v_arr_p_coords = np.array([[a for a in v][:2] for v in v_arr_p])
- tuple v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])
- tuple v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])

- tuple v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])
- tuple vtx_p = Vortex(i1,v_arr_p_coords[i1][0],v_arr_p_coords[i1][1],True,sign=v_arr_p_sign[i1])
- tuple vtx_c = Vortex(-1-i2,v_arr_c_coords[i2][0],v_arr_c_coords[i2][1],True,sign=v_arr_c_sign[i2])
- tuple index_r = vorts_c.idx_min_dist(vorts_p.element(i3))
- tuple v0c = vorts_c.element(index_r[0])
- tuple v0p = vorts_p.element(i3)
- tuple v1c = vorts_c.element(index_r[0])
- list uid_c = [[a for a in b][3] for b in vorts_c.as_np()]
- list uid_p = [[a for a in b][3] for b in vorts_p.as_np()]
- tuple dpc = set(uid_p)
- tuple dcp = set(uid_c)
- list vtx_pos_p = [ ]
- list vtx_pos_c = [ ]
- tuple vorts_c_update = sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])

3.15.1 Class Documentation

3.15.1.1 class vort::Vortex

Definition at line 41 of file vort.py.

Collaboration diagram for vort.Vortex:

3.15.1.2 class vort::VtxList

Definition at line 75 of file vort.py.

Collaboration diagram for vort.VtxList:

3.15.2 Function Documentation

3.15.2.1 def vort.__init__ (*self, uid, x, y, isOn, sign =* 1)

Definition at line 44 of file vort.py.

Referenced by __init__().

```
00044
    def __init__(self,uid,x,y,isOn,sign=1):
```

Here is the caller graph for this function:

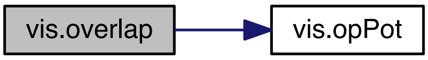
3.15.2.2 def vort.__init__ (*self*)

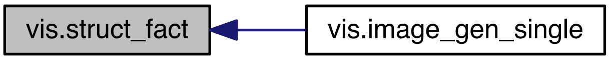
Definition at line 78 of file vort.py.

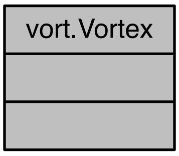
References __init__().

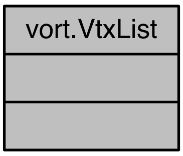
```
00078
    def __init__(self):
```

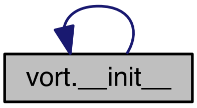
Here is the call graph for this function:


vis.overlap
vis.opPot


vis.struct_fact
vis.image_gen_single


vort.Vortex


vort.VtxList


vort.__init__

3.15.2.3 def vort.add (*self*, *Vtx*, *index* = None)

Definition at line 126 of file vort.py.

```
00126
    def add(self,Vtx,index=None):
```

3.15.2.4 def vort.as_np (*self*)

Definition at line 142 of file vort.py.

```
00142
    def as_np(self):
```

3.15.2.5 def vort.dist (*self*, *vtx*)

Definition at line 69 of file vort.py.

Referenced by vortArrange().

```
00069
    def dist(self,vtx):
```

Here is the caller graph for this function:

3.15.2.6 def vort.do_the_thing (*start*, *fin*, *incr*)

Definition at line 221 of file vort.py.

```
00221
def do_the_thing(start,fin,incr):
```

3.15.2.7 def vort.element (*self*, *pos*)

Definition at line 85 of file vort.py.

```
00085
    def element(self,pos):
```

3.15.2.8 def vort.idx_min_dist (*self*, *vortex*, *isSelf* = False)

Definition at line 160 of file vort.py.

```
00160
    def idx_min_dist(self,vortex, isSelf=False):
```

3.15.2.9 def vort.max_uid (*self*)

Definition at line 109 of file vort.py.

References max_uid.

```
00109
    def max_uid(self):
```

3.15.2.10 def vort.remove (*self, pos*)

Definition at line 176 of file vort.py.

```
00176
    def remove(self,pos):
```

3.15.2.11 def vort.swap_uid (*self, uid_i, uid_f*)

Definition at line 195 of file vort.py.

```
00195
    def swap_uid(self,uid_i,uid_f):
```

3.15.2.12 def vort.update_next (*self, next*)

Definition at line 64 of file vort.py.

```
00064
    def update_next(self,next):
```

3.15.2.13 def vort.update_on (*self, isOn*)

Definition at line 59 of file vort.py.

```
00059
    def update_on(self,isOn):
```

3.15.2.14 def vort.update_uid (*self, uid*)

Definition at line 54 of file vort.py.

Referenced by vort_increase().

```
00054
    def update_uid(self,uid):
```

Here is the caller graph for this function:

3.15.2.15 def vort.vort_decrease (*self, positions, vorts_p*)

Definition at line 202 of file vort.py.

```
00202
    def vort_decrease(self,positions,vorts_p):
```

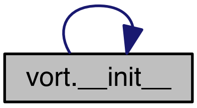
3.15.2.16 def vort.vort_increase (*self, positions, vorts_p*)

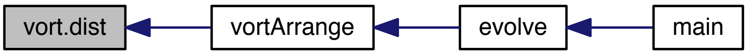
Definition at line 212 of file vort.py.

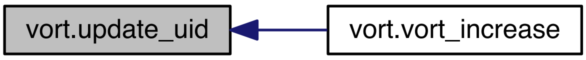
References update_uid().

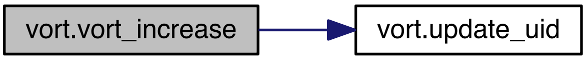
```
00212
    def vort_increase(self,positions,vorts_p):
```

Here is the call graph for this function:


vort.__init__


vort.dist
vortArrange
evolve
main


vort.update_uid
vort.vort_increase


vort.vort_increase
vort.update_uid

3.15.2.17 def vort.vtx_uid (*self*, *uid*)

Definition at line 99 of file vort.py.

```
00099
    def vtx_uid(self,uid):
```

3.15.2.18 def vort.write_out (*self*, *time*, *data*)

Definition at line 155 of file vort.py.

```
00155
    def write_out(self,time,data):
```

3.15.3 Variable Documentation

3.15.3.1 tuple vort.c = ConfigParser.ConfigParser()

Definition at line 27 of file vort.py.

3.15.3.2 int vort.counter = 0

Definition at line 162 of file vort.py.

Referenced by olPos(), vortCentre(), and vortPos().

3.15.3.3 vort.current = self.element(pos-1)

Definition at line 179 of file vort.py.

3.15.3.4 list vort.data = [ ]

Definition at line 145 of file vort.py.

3.15.3.5 tuple vort.dcp = set(uid_c)

Definition at line 258 of file vort.py.

3.15.3.6 tuple vort.dpc = set(uid_p)

Definition at line 257 of file vort.py.

3.15.3.7 tuple vort.dt = (c.getfloat('Params','dt'))

Definition at line 36 of file vort.py.

3.15.3.8 list vort.dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]

Definition at line 144 of file vort.py.

3.15.3.9 tuple vort.dx = (c.getfloat('Params','dx'))

Definition at line 35 of file vort.py.

3.15.3.10 tuple vort.evMaxVal = int(c.getfloat('Params','esteps'))

Definition at line 33 of file vort.py.

3.15.3.11 tuple vort.gndMaxVal = int(c.getfloat('Params','gsteps'))

Definition at line 32 of file vort.py.

3.15.3.12 int vort.i = 0

Definition at line 146 of file vort.py.

3.15.3.13 tuple vort.incr = int(c.getfloat('Params','print_out'))

Definition at line 34 of file vort.py.

3.15.3.14 tuple vort.index_r = vorts_c.idx_min_dist(vorts_p.element(i3))

Definition at line 243 of file vort.py.

3.15.3.15 tuple vort.max_uid = vorts_p.max_uid()

Definition at line 204 of file vort.py.

Referenced by max_uid().

3.15.3.16 int vort.pos = 0

Definition at line 102 of file vort.py.

3.15.3.17 int vort.pos_l = 0

Definition at line 87 of file vort.py.

3.15.3.18 tuple vort.r = m.sqrt((self.x - vtx.x)**2 + (self.y - vtx.y)**2)

Definition at line 71 of file vort.py.

3.15.3.19 vort.ret_idx = counter

Definition at line 163 of file vort.py.

3.15.3.20 list vort.uid_c = [[a for a in b][3] for b in vorts_c.as_np()]

Definition at line 254 of file vort.py.

3.15.3.21 list vort.uid_p = [[a for a in b][3] for b in vorts_p.as_np()]

Definition at line 255 of file vort.py.

3.15.3.22 tuple vort.v0c = vorts_c.element(index_r[0])

Definition at line 245 of file vort.py.

3.15.3.23 tuple vort.v0p = vorts_p.element(i3)

Definition at line 246 of file vort.py.

3.15.3.24 tuple vort.v1c = vorts_c.element(index_r[0])

Definition at line 247 of file vort.py.

3.15.3.25 tuple vort.v_arr_c = genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')

Definition at line 229 of file vort.py.

3.15.3.26 tuple vort.v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])

Definition at line 231 of file vort.py.

3.15.3.27 tuple vort.v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])

Definition at line 233 of file vort.py.

3.15.3.28 tuple vort.v_arr_p = genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')

v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')

Definition at line 224 of file vort.py.

3.15.3.29 tuple vort.v_arr_p_coords = np.array([[a for a in v][:2] for v in v_arr_p])

Definition at line 230 of file vort.py.

3.15.3.30 tuple vort.v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])

Definition at line 232 of file vort.py.

3.15.3.31 vort.val = 0

Definition at line 111 of file vort.py.

3.15.3.32 tuple vort.vorts_c = VtxList()

Definition at line 227 of file vort.py.

3.15.3.33 tuple vort.vorts_c_update = sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])

Definition at line 269 of file vort.py.

3.15.3.34 tuple vort.vorts_p = VtxList()

Definition at line 226 of file vort.py.

3.15.3.35 tuple vort.vtx = self.head

Definition at line 89 of file vort.py.

3.15.3.36 tuple vort.vtx_c = Vortex(-1-i2,v_arr_c_coords[i2][0],v_arr_c_coords[i2][1],True,sign=v_arr_c_sign[i2])

Definition at line 239 of file vort.py.

3.15.3.37 tuple vort.vtx_p = Vortex(i1,v_arr_p_coords[i1][0],v_arr_p_coords[i1][1],True,sign=v_arr_p_sign[i1])

Definition at line 235 of file vort.py.

3.15.3.38 tuple vort.vtx_pos = self.vtx_uid(uid_i)

Definition at line 197 of file vort.py.

3.15.3.39 tuple vort.vtx_pos_c = [ ]

Definition at line 260 of file vort.py.

3.15.3.40 tuple vort.vtx_pos_p = [ ]

Definition at line 259 of file vort.py.

3.15.3.41 tuple vort.xDim = int(c.getfloat('Params','xDim'))

Definition at line 30 of file vort.py.

3.15.3.42 tuple vort.xMax = (c.getfloat('Params','xMax'))

Definition at line 37 of file vort.py.

3.15.3.43 tuple vort.yDim = int(c.getfloat('Params','yDim'))

Definition at line 31 of file vort.py.

3.15.3.44 tuple vort.yMax = (c.getfloat('Params','yMax'))

Definition at line 38 of file vort.py.

Chapter 4

File Documentation

4.1 bin/path.sh File Reference

4.2 path.sh

```
#!/bin/bash
export PATH=$PATH:/usr/local/cuda/bin:/usr/local/cuda/open64/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

4.3 bin/run.sh File Reference

4.4 run.sh

```
#!/bin/bash
i=0
count=0
declare -a JOBS=(-1 -1 -1 -1 -1 -1 -1 -1)
function run_gpue_tes {
    echo $1 >> test_file.txt
}
function run_gpue {
    sleep 1
    A=$(date '+%y/%m/%d/%H_%M_%S')
    if [ -d ./$A ]; then
        echo "Exists"
        A=$A-$i
        i=$((i+1))
    fi
    echo $A
    mkdir -p $A
    cp ./gpue ./$A; cp -r ./src ./$A; cp -r ./include ./$A; cp ./Makefile ./$A; cp -r ./py ./$A; cp -r ./bin ./$A; cp ./wfc_load ./$A; cp ./wfci_load ./$A;
    cd ./$A
    pwd >> result.log
    echo $1 >>result.log
    mail -s "#Started GPU Job# $A" lee.oriordan@oist.jp < result.log
    ./gpue $1 2>&1> result.log
    mkdir -p ./images
    python ./py/vis.py >> result.log
    cp *.png ./images
    cd ./images
    ls | grep wfc_evr | grep _abs | grep png | sort -k3 -t _ -n > list1.txt;mencoder mf://@list1.txt -mf
 w=1280:h=1024:fps=24:type=png -oac copy -ovc lavc -lavcopts
 vcodec=mpeg4:mbd=2:mv0:trell:v4mv:cbp:last_pred=3:predia=2:dia=2:vmax_b_frames=2:vb_strategy=1:precmp=2:cmp=2:subcmp=2
 wfc_${PWD##*/}.avi
    ls | grep wfc_evr | grep _diff | grep png | sort -k3 -t _ -n > list1.txt;mencoder mf://@list2.txt -mf
 w=1280:h=1024:fps=24:type=png -oac copy -ovc lavc -lavcopts
 vcodec=mpeg4:mbd=2:mv0:trell:v4mv:cbp:last_pred=3:predia=2:dia=2:vmax_b_frames=2:vb_strategy=1:precmp=2:cmp=2:subcmp=2
 wfc_${PWD##*/}_diff.avi
    rm -rf ./*.png
    python ./py/hist3d.py
    rm wfc*
    mail -s "#Completed GPU Job# $A" lee.oriordan@oist.jp < result.log
    cd ../../../../..
}

while read line ; do
    run_gpue "$line" &
    #echo "Running $line"
    JOBS[$count]=$!
    let count+=1
    sleep 1
    if [ $count -gt 7 ]; then
        wait
        count=0
    fi
done < ./bin/run_params.conf
```

4.5 bin/sanity_test.sh File Reference

Variables

- FILE
- do let POSITION if ["$i"!="0.0000000000000000e+00"]

4.5.1 Variable Documentation

4.5.1.1 FILE

Initial value:

```
=$1
COUNTER=0
POSITION=-1
ARR[0]=0
for i in $(cat $FILE)
```

Definition at line 20 of file sanity_test.sh.

Referenced by readIn(), readState(), writeOut(), writeOutDouble(), writeOutInt(), writeOutInt2(), writeOutParam(), and writeOutVortex().

4.5.1.2 do let POSITION if["$i"!="0.0000000000000000e+00"]

Definition at line 27 of file sanity_test.sh.

4.6 sanity_test.sh

```
00001 #
00002 # sanity_test.sh - GPUE: Split Operator based GPU solver for Nonlinear
00003 # Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 # Morgan, Neil Crowley.
00005
00006 # This library is free software; you can redistribute it and/or modify
00007 # it under the terms of the GNU Lesser General Public License as
00008 # published by the Free Software Foundation; either version 2.1 of the
00009 # License, or (at your option) any later version. This library is
00010 # distributed in the hope that it will be useful, but WITHOUT ANY
00011 # WARRANTY; without even the implied warranty of MERCHANTABILITY or
00012 # FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00013 # License for more details. You should have received a copy of the GNU
00014 # Lesser General Public License along with this library; if not, write
00015 # to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
00016 # Boston, MA 02111-1307 USA
00017 #
00018
00019 #!/bin/bash
00020 FILE=$1
00021 COUNTER=0
```

```
POSITION=-1
ARR[0]=0
for i in $(cat $FILE);
do
    let POSITION++
    if [ "$i" != "0.0000000000000000e+00" ];
    then
        ARR[$COUNTER]=$POSITION
        let COUNTER++
    fi

done
echo Non-zero elements $COUNTER
echo "Elements located at:"

for item in ${ARR[*]}
do
    printf "%s\n" $item
done
```

4.7 bin/upload_vids.sh File Reference

Functions

- do echo (if[[$(basename $(dirname $i))==’images’]];then cd $(dirname $i)/../bin;TITLE=$(head-n 1 run_↩ params.conf) SUMMARY=$(head-n 20../result.log) cd-google youtube post--category Tech $i--title"$TITLE"--summary"$SUMMARY"--access=unlisted $i fi)

Variables

- OLDPWD

4.7.1 Function Documentation

4.7.1.1 do echo (if];then cd $(dirname $i)/../bin; *TITLE[[$(basename $(dirname $i))==’images’]* = $(head-n 1 run_↩ params.conf) SUMMARY=$(head-n 20../result.log) cd-google youtube post--category Tec)

4.7.2 Variable Documentation

4.7.2.1 OLDPWD

Initial value:

```
=$(pwd)
for i in $(cat ./ogg.txt | grep wfc)
```

Definition at line 2 of file upload_vids.sh.

4.8 upload_vids.sh

```
00001 #!/bin/bash
00002 OLDPWD=$(pwd)
00003 for i in $(cat ./ogg.txt | grep wfc);
00004 do
00005     echo $(if [[ $(basename $(dirname $i))==’images’ ]];
00006         then
00007             cd $(dirname $i)/../bin;
00008             TITLE=$(head -n 1 run_params.conf)
00009             SUMMARY=$(head -n 20 ../result.log)
00010             cd -
00011             google youtube post --category Tech $i --title "$TITLE" --summary "$SUMMARY" --access=
      unlisted $i
00012         fi);
00013 done
00014
```

4.9 bin/zippit.sh File Reference

Functions

- for i in (cat manifest.txt)

Variables

- do echo Working on $i

4.9.1 Function Documentation

4.9.1.1 for i in (cat manifest. *txt*)

Referenced by conjugate(), and fInvSqRt().

Here is the caller graph for this function:

4.9.2 Variable Documentation

4.9.2.1 $HOME builds bin pigz p r $i

Definition at line 2 of file zippit.sh.

4.10 zippit.sh

```
00001 #!/bin/bash
00002 for i in $(cat manifest.txt); do echo 'Working on $i'; $HOME/builds/bin/pigz -
      p 24 -r $i; done
```

4.11 include/constants.h File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define PI 3.141592653589793
- #define HBAR 1.05457148e-34
- #define MU_N 5.05078324e-27
- #define MU_B 9.27400915e-24
- #define Q 1.602176565e-19
- #define MU_0 4∗PI∗1e-7
- #define EPSILON_0 8.854187817620e-12
- #define INV_RT_2 0.7071067811865475
- #define RT_2 1.4142135623730951

4.11.1 Macro Definition Documentation

4.11.1.1 #define EPSILON_0 8.854187817620e-12

Definition at line 27 of file constants.h.

4.11.1.2 #define HBAR 1.05457148e-34

Definition at line 22 of file constants.h.

Referenced by cMultDensity(), delta_define(), energyCalc(), initialise(), and optLatSetup().

4.11.1.3 #define INV_RT_2 0.7071067811865475

Definition at line 28 of file constants.h.

4.11.1.4 #define MU_0 4∗PI∗1e-7

Definition at line 26 of file constants.h.

4.11.1.5 #define MU_B 9.27400915e-24

Definition at line 24 of file constants.h.

4.11.1.6 #define MU_N 5.05078324e-27

Definition at line 23 of file constants.h.

4.11.1.7 #define PI 3.141592653589793

Definition at line 21 of file constants.h.

Referenced by cMultDensity(), evolve(), findVortex(), initialise(), main(), optLatSetup(), phaseTest(), and vortAngle().

4.11.1.8 #define Q 1.602176565e-19

Definition at line 25 of file constants.h.

4.11.1.9 #define RT_2 1.4142135623730951

Definition at line 29 of file constants.h.

4.12 constants.h

```
00001 /*
00002 * constants.h - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
00011 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
```

```
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details. You should have received a copy of the GNU
 * Lesser General Public License along with this library; if not, write
 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
 * Boston, MA 02111-1307 USA
 */
#ifndef CONSTANTS_H
#define CONSTANTS_H

#define PI 3.141592653589793
#define HBAR 1.05457148e-34 // m^2 kg/s
#define MU_N 5.05078324e-27 // J/T  Nuclear magneton
#define MU_B 9.27400915e-24 // J/T  Bohr magneton
#define Q 1.602176565e-19 // C  Elementary charge of proton
#define MU_0 4*PI*1e-7 // V*S/A*m or H/m or N/A^2  Vacuum permeability
#define EPSILON_0 8.854187817620e-12 // F/m  Vacuum permittivity
#define INV_RT_2 0.7071067811865475 // 1/sqrt(2)
#define RT_2 1.4142135623730951 // sqrt(2)

#endif
```

4.13 include/ds.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Include dependency graph for ds.h: This graph shows which files directly or indirectly include this file:

Classes

- struct Param
- struct Array

Typedefs

- typedef struct Param Param
- typedef struct Array Array

Functions

- void initArr (Array ∗arr, size_t initLen)
- void appendData (Array ∗arr, char ∗t, double d)
- void freeArray (Array ∗arr)
- Param newParam (char ∗t, double d)

4.13.1 Class Documentation

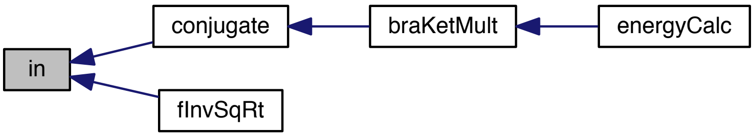
4.13.1.1 struct Param

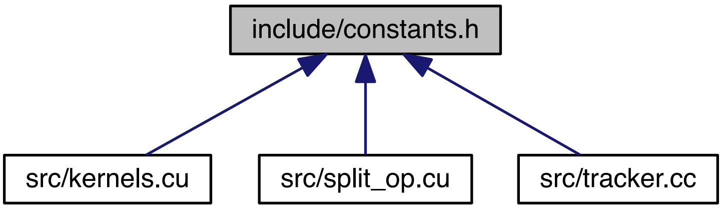
Definition at line 25 of file ds.h.

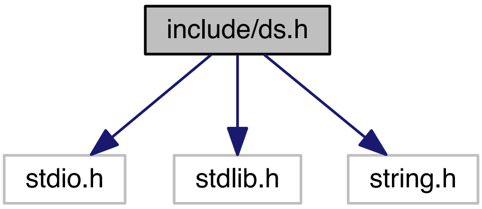
Collaboration diagram for Param:

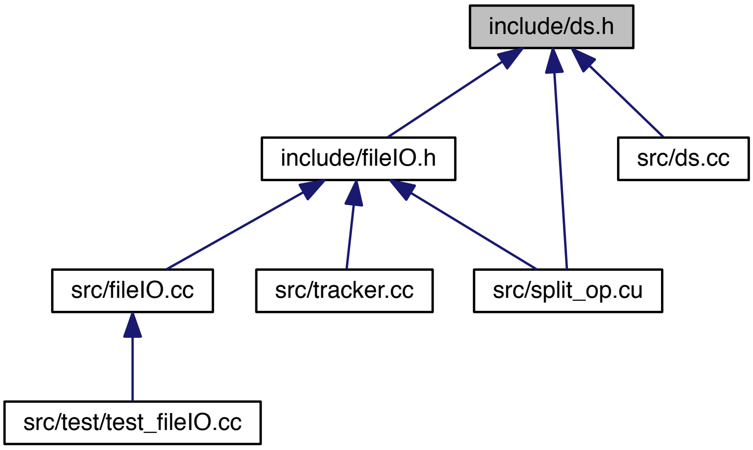
Class Members

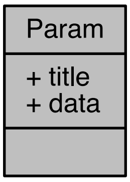
double	data	


in
conjugate
braKetMult
energyCalc
fInvSqRt


include/constants.h
src/kernels.cu
src/split_op.cu
src/tracker.cc


include/ds.h
stdio.h
stdlib.h
string.h


include/ds.h
include/fileIO.h
src/ds.cc
src/fileIO.cc
src/tracker.cc
src/split_op.cu
src/test/test_fileIO.cc


Param
+ title
+ data

char	title[32]	

4.13.1.2 struct Array

Definition at line 31 of file ds.h.

Collaboration diagram for Array:

Class Members

Param ∗	array	
size_t	length	
size_t	used	

4.13.2 Typedef Documentation

4.13.2.1 typedef struct Array Array

Definition at line 36 of file ds.h.

4.13.2.2 typedef struct Param Param

Definition at line 29 of file ds.h.

4.13.3 Function Documentation

4.13.3.1 void appendData (**Array** ∗ *arr,* char ∗ *t,* double *d*)

Definition at line 27 of file ds.cc.

References Array::array, Array::length, newParam(), overlap::p, and Array::used.

Referenced by evolve(), initialise(), optLatSetup(), and parseArgs().

```
00027                                               {
00028     Param p = newParam(t,d);
00029     if(arr->used == arr->length){
00030         arr->length *= 2;
00031         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
      Param));
00032     }
00033     arr->array[arr->used] = p;
00034     arr->used = arr->used + 1;
00035 }
```

Here is the call graph for this function:

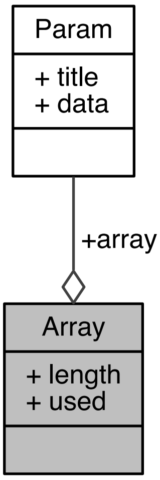
Here is the caller graph for this function:

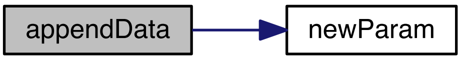
4.13.3.2 void freeArray (**Array** ∗ *arr*)

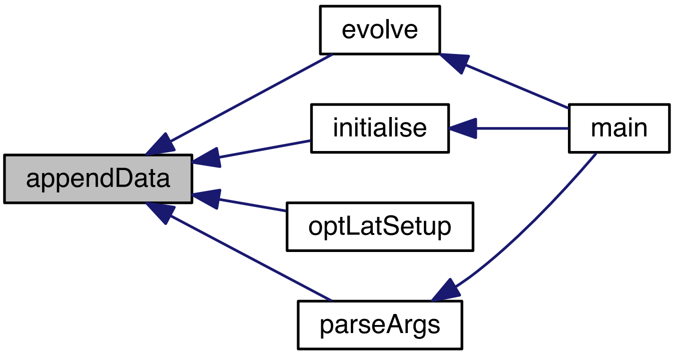
Definition at line 37 of file ds.cc.

References Array::array, Array::length, and Array::used.

```
00037                        {
00038     free(arr->array);
00039     arr->array = NULL;
00040     arr->used = 0;
00041     arr->length = 0;
00042 }
```


Param
+ title
+ data
+array
Array
+ length
+ used


appendData
newParam


evolve
initialise
main
appendData
optLatSetup
parseArgs

4.13.3.3 void initArr (Array * *arr,* size_t *initLen*)

Definition at line 21 of file ds.cc.

References Array::array, Array::length, and Array::used.

Referenced by main().

```
00021                                        {
00022     arr->array = (Param*) malloc(initLen*sizeof(Param));
00023     arr->used = 0;
00024     arr->length = initLen;
00025 }
```

Here is the caller graph for this function:

4.13.3.4 Param newParam (char * *t,* double *d*)

Definition at line 44 of file ds.cc.

References Param::data, overlap::p, and Param::title.

Referenced by appendData().

```
00044                                 {
00045     Param p;
00046     strcpy(p.title,t);
00047     p.data = d;
00048     return p;
00049 }
```

Here is the caller graph for this function:

4.14 ds.h

```
/*
* ds.h - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#ifndef DS_H
#define DS_H
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

struct Param{
    char title[32];
    double data;
};
typedef struct Param Param;

struct Array{
    Param *array;
    size_t length;
    size_t used;
};
typedef struct Array Array;

void initArr(Array *arr, size_t initLen);
void appendData(Array *arr, char* t, double d);
void freeArray(Array *arr);
Param newParam(char* t,double d);
#endif
```

4.15 include/fileIO.h File Reference

```
#include "../include/ds.h"
#include "../include/tracker.h"
```

Include dependency graph for fileIO.h: This graph shows which files directly or indirectly include this file:

Functions

- void hdfWriteDouble (int xDim, double ∗op, long incr, char ∗dset)
- void hdfWriteComplex (int xDim, double2 ∗wfc, long incr, char ∗dset)
- double2 ∗ readIn (char ∗, char ∗, int, int)
- void writeOut (char ∗, char ∗, double2 ∗, int, int)
- void writeOutDouble (char ∗, char ∗, double ∗, int, int)
- void writeOutInt (char ∗, char ∗, int ∗, int, int)
- void writeOutInt2 (char ∗, char ∗, int2 ∗, int, int)
- void writeOutVortex (char ∗, char ∗, struct Vortex ∗, int, int)
- void writeOutParam (char ∗, Array, char ∗)
- int readState (char ∗)

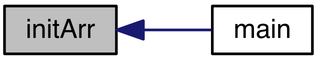
4.15.1 Function Documentation

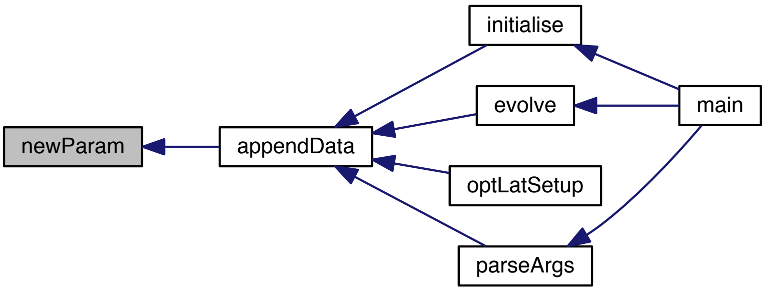
4.15.1.1 void hdfWriteComplex (int *xDim,* double2 ∗ *wfc,* long *incr,* char ∗ *dset*)

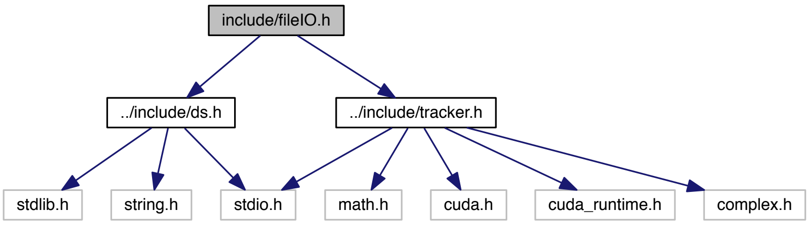
Definition at line 46 of file fileIO.cc.

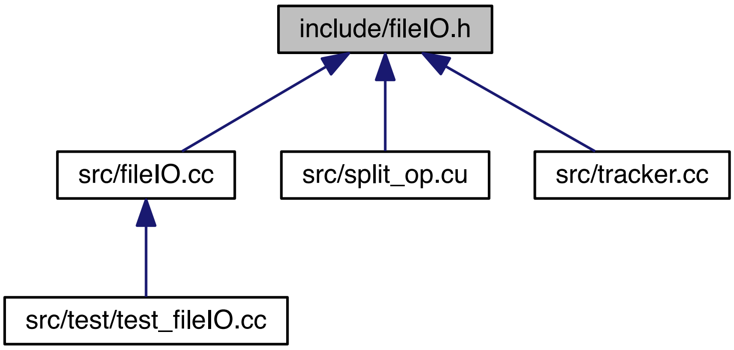
References x, xDim, and y.

```
00046                                                                              {
00047
00048     typedef struct d2{
00049         double x;
00050         double y;
00051     }d2;
00052     hid_t file_id;
00053     hsize_t dims[2];
00054     dims[0]=xDim;
00055     dims[1]=xDim;
00056     herr_t status;
00057     double2 tmp;
00058     hid_t complex_id = H5Tcreate(H5T_COMPOUND, sizeof(tmp));
00059     H5Tinsert (complex_id, "real", HOFFSET(d2,x), H5T_NATIVE_DOUBLE);
00060     H5Tinsert (complex_id, "imaginary", HOFFSET(d2,y), H5T_NATIVE_DOUBLE);
00061
00062     char dataset[32];
00063     strcpy(dataset,"/");
00064     strcat(dataset,dset);
00065     if(incr==0){
00066         file_id = H5Fcreate("GPUE.h5",H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
00067     }
00068     else{
00069         file_id = H5Fopen( "GPUE.h5", H5F_ACC_RDWR, H5P_DEFAULT );
00070     }
00071         status = H5LTmake_dataset( file_id, dset, 2, dims, complex_id, wfc );
00072
00073         status = H5Fclose(file_id);
00074 }
```


initArr
main


initialise
evolve
main
newParam
appendData
optLatSetup
parseArgs


include/fileIO.h
../include/ds.h
../include/tracker.h
stdlib.h
string.h
stdio.h
math.h
cuda.h
cuda_runtime.h
complex.h


include/fileIO.h
src/fileIO.cc
src/split_op.cu
src/tracker.cc
src/test/test_fileIO.cc

4.15.1.2 void hdfWriteDouble (int *xDim,* double ∗ *op,* long *incr,* char ∗ *dset*)

Definition at line 27 of file fileIO.cc.

References xDim.

```
00027                                                                  {
00028     hid_t file_id;
00029     hsize_t dims[2];
00030     dims[0]=xDim;
00031     dims[1]=xDim;
00032     herr_t status;
00033     char dataset[32];
00034     strcpy(dataset,"/");
00035     strcat(dataset,dset);
00036     if(incr==0){
00037         file_id = H5Fcreate("GPUE.h5",H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
00038     }
00039     else{
00040         file_id = H5Fopen( "GPUE.h5", H5F_ACC_RDWR, H5P_DEFAULT );
00041     }
00042         status = H5LTmake_dataset( file_id, dset, 2, dims, H5T_NATIVE_DOUBLE, op );
00043 
00044         status = H5Fclose(file_id);
00045 }
```

4.15.1.3 double2∗ readIn (char ∗ , char ∗ , int , int)

Definition at line 76 of file fileIO.cc.

References vis_ev::f, FILE, vis::i, and yDim.

Referenced by main().

```
00076                                                               {
00077     FILE *f;
00078     f = fopen(fileR,"r");
00079     int i = 0;
00080     double2 *arr = (double2*) malloc(sizeof(double2)*xDim*yDim);
00081     double line;
00082     while(fscanf(f,"%lE",&line) > 0){
00083         arr[i].x = line;
00084         ++i;
00085     }
00086     fclose(f);
00087     f = fopen(fileI,"r");
00088     i = 0;
00089     while(fscanf(f,"%lE",&line) > 0){
00090         arr[i].y = line;
00091         ++i;
00092     }
00093     fclose(f);
00094     return arr;
00095 }
```

Here is the caller graph for this function:

4.15.1.4 int readState (char ∗)

Definition at line 166 of file fileIO.cc.

References vis_ev::f, and FILE.

```
00166                           {
00167     FILE *f;
00168     f = fopen(name,"r");
00169     fclose(f);
00170     return 0;
00171 }
```

4.15.1.5 void writeOut (char ∗ , char ∗ , double2 ∗ , int , int)

Definition at line 109 of file fileIO.cc.

Referenced by evolve(), initialise(), and main().

```
00109                                                                             {
00110     FILE *f;
00111     sprintf (buffer, "%s_%d", file, step);
00112     f = fopen (buffer,"w");
00113     int i;
00114     for (i = 0; i < length; i++)
00115         fprintf (f, "%.16e\n",data[i].x);
00116     fclose (f);
00117
00118     sprintf (buffer, "%si_%d", file, step);
00119     f = fopen (buffer,"w");
00120     for (i = 0; i < length; i++)
00121         fprintf (f, "%.16e\n",data[i].y);
00122     fclose (f);
00123 }
```

Here is the caller graph for this function:

4.15.1.6 void writeOutDouble (char ∗ , char ∗ , double ∗ , int , int)

Definition at line 125 of file fileIO.cc.

References vis_ev::f, FILE, and vis::i.

Referenced by evolve(), initialise(), and main().

```
00125                                                                                  {
00126     FILE *f;
00127     sprintf (buffer, "%s_%d", file, step);
00128     f = fopen (buffer,"w");
00129     int i;
00130     for (i = 0; i < length; i++)
00131         fprintf (f, "%.16e\n",data[i]);
00132     fclose (f);
00133 }
```

Here is the caller graph for this function:

4.15.1.7 void writeOutInt (char ∗ , char ∗ , int ∗ , int , int)

Definition at line 135 of file fileIO.cc.

References vis_ev::f, FILE, and vis::i.

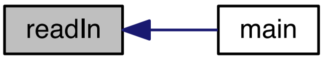
```
00135                                                                            {
00136     FILE *f;
00137     sprintf (buffer, "%s_%d", file, step);
00138     f = fopen (buffer,"w");
00139     int i;
00140     for (i = 0; i < length; i++)
00141         fprintf (f, "%d\n",data[i]);
00142     fclose (f);
00143 }
```

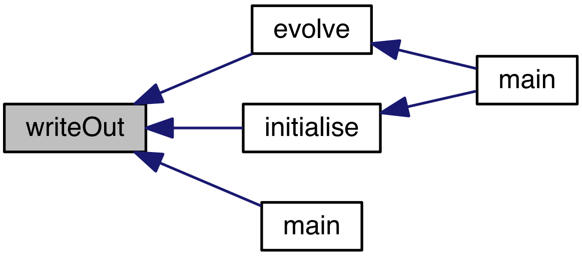
4.15.1.8 void writeOutInt2 (char ∗ , char ∗ , int2 ∗ , int , int)

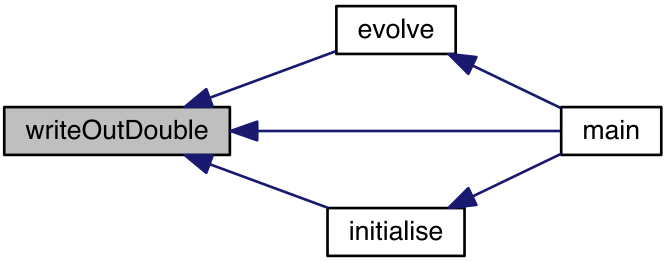
Definition at line 145 of file fileIO.cc.

References vis_ev::f, FILE, vis::i, x, and y.

```
00145                                                                               {
00146     FILE *f;
```


readIn
main


writeOut
evolve
initialise
main
main


writeOutDouble
evolve
main
initialise

```
00147     sprintf (buffer, "%s_%d", file, step);
00148     f = fopen (buffer,"w");
00149     int i;
00150     for (i = 0; i < length; i++)
00151         fprintf (f, "%d,%d\n",data[i].x,data[i].y);
00152     fclose (f);
00153 }
```

4.15.1.9 void writeOutParam (char ∗ , Array , char ∗)

Definition at line 97 of file fileIO.cc.

References Array::array, Param::data, vis_ev::f, FILE, vis::i, Param::title, and Array::used.

Referenced by evolve(), and main().

```
00097                                                       {
00098     FILE *f;
00099     sprintf(buffer, "%s", file);
00100     f = fopen(file,"w");
00101     fprintf(f,"[Params]\n");
00102     for (int i = 0; i < arr.used; ++i){
00103         fprintf(f,"%s=",arr.array[i].title);
00104         fprintf(f,"%e\n",arr.array[i].data);
00105     }
00106     fclose(f);
00107 }
```

Here is the caller graph for this function:

4.15.1.10 void writeOutVortex (char ∗ , char ∗ , struct Vortex ∗ , int , int)

Definition at line 155 of file fileIO.cc.

References Vortex::coords, vis_ev::f, FILE, vis::i, Vortex::sign, and Vortex::wind.

Referenced by evolve().

```
00155                                                                                         {
00156     FILE *f;
00157     sprintf (buffer, "%s_%d", file, step);
00158     f = fopen (buffer,"w");
00159     int i;
00160     fprintf (f, "#X,Y,WINDING,SIGN\n");
00161     for (i = 0; i < length; i++)
00162         fprintf (f, "%d,%d,%d,%d\n",data[i].coords.x,data[i].coords.y,
      data[i].wind,data[i].sign);
00163     fclose (f);
00164 }
```

Here is the caller graph for this function:

4.16 fileIO.h

```
00001 /*
00002 * fileIO.h - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
00011 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
00012 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00013 * License for more details. You should have received a copy of the GNU
00014 * Lesser General Public License along with this library; if not, write
00015 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
00016 * Boston, MA 02111-1307 USA
```

```
*/

#ifndef FILEIO_H
#define FILEIO_H
#include "../include/ds.h"
#include "../include/tracker.h"
void hdfWriteDouble(int xDim, double* op, long incr, char* dset);
void hdfWriteComplex(int xDim, double2* wfc, long incr, char* dset);
double2* readIn(char*, char*, int, int);
void writeOut(char*, char*, double2*, int, int );
void writeOutDouble(char*, char*, double*, int, int);
void writeOutInt(char*, char*, int*, int, int);
void writeOutInt2(char*, char*, int2*, int, int);
void writeOutVortex(char*, char*, struct Vortex*, int, int);
void writeOutParam(char*, Array, char*);
int readState(char*);
#endif
```

4.17 include/kernels.h File Reference

`#include <stdio.h>`

Include dependency graph for kernels.h: This graph shows which files directly or indirectly include this file:

Functions

- __device__ unsigned int getGid3d3d ()
- __device__ unsigned int getBid3d3d ()
- __device__ unsigned int getTid3d3d ()
- __device__ double complexMagnitudeSquared (double2)
- __global__ void cMult (cufftDoubleComplex ∗, cufftDoubleComplex ∗, cufftDoubleComplex ∗)
- __global__ void pinVortex (cufftDoubleComplex ∗, cufftDoubleComplex ∗, cufftDoubleComplex ∗)
- __global__ void cMultDensity (double2 ∗, double2 ∗, double2 ∗, double, double, double, int, int)
- __global__ void scalarDiv (double2 ∗, double, double2 ∗)

  Divides both components of vector type "in", by the value "factor".
- __global__ void scalarDiv1D (double2 ∗, double2 ∗)
- __global__ void scalarDiv2D (double2 ∗, double2 ∗)
- __global__ void scalarDiv_wfcNorm (double2 ∗, double, double2 ∗, double2 ∗)

  As above, but normalises for wfc.
- __global__ void reduce (double2 ∗, double ∗)
- __global__ void multipass (cufftDoubleComplex ∗, cufftDoubleComplex ∗, int)
- __global__ void angularOp (double, double, double2 ∗, double ∗, double2 ∗)
- __device__ double2 conjugate (double2 in)
- __device__ double2 realCompMult (double scalar, double2 comp)
- __global__ void energyCalc (double2 ∗wfc, double2 ∗op, double dt, double2 ∗energy, int gnd_state, int op_space, double sqrt_omegaz_mass)
- __device__ double2 braKetMult (double2 in1, double2 in2)
- __global__ void pSum (double ∗in1, double ∗output, int pass)

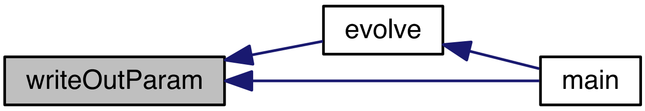
  Routine for parallel summation.

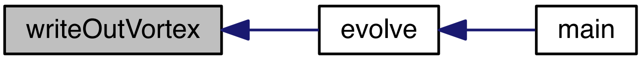
4.17.1 Function Documentation

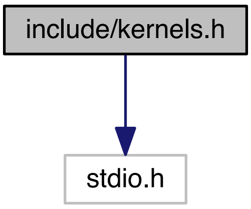
4.17.1.1 __global__ void angularOp (double , double , double2 ∗ , double ∗ , double2 ∗)

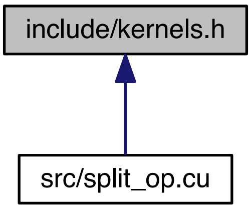
Definition at line 148 of file kernels.cu.

References getGid3d3d(), and result.


writeOutParam
evolve
main


writeOutVortex
evolve
main


include/kernels.h
stdio.h


include/kernels.h
src/split_op.cu

```
00148                                                                                    {
00149     unsigned int gid = getGid3d3d();
00150     double2 result;
00151     double op;
00152     op = exp( -omega*xpyypx[gid]*dt);
00153     result.x=wfc[gid].x*op;
00154     result.y=wfc[gid].y*op;
00155     out[gid]=result;
00156 }
```

Here is the call graph for this function:

4.17.1.2 __device__ double2 braKetMult (double2 *in1,* double2 *in2*) [inline]

Definition at line 83 of file kernels.cu.

References complexMultiply(), and conjugate().

Referenced by energyCalc().

```
00084 {
00085     return complexMultiply(conjugate(in1),in2);
00086 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.17.1.3 __global__ void cMult (cufftDoubleComplex ∗ , cufftDoubleComplex ∗ , cufftDoubleComplex ∗)

4.17.1.4 __global__ void cMultDensity (double2 ∗ , double2 ∗ , double2 ∗ , double , double , double , int , int)

Definition at line 99 of file kernels.cu.

References complexMagnitudeSquared(), HBAR, mass, observables::N, PI, result, x, and y.

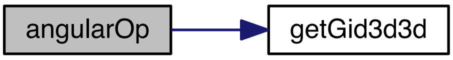
```
00099
                          {
00100     double2 result;
00101     double gDensity;
00102     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00103     gDensity = (0.5*N)*complexMagnitudeSquared(in2[tid])*4*
      HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*
      HBAR));
00104
00105     if(gstate == 0){
00106         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR) );
00107         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00108         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00109     }
00110     else{
00111         double2 tmp;
00112         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(
      dt/HBAR));
00113         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(
      dt/HBAR));
00114
00115         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00116         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00117     }
00118     out[tid] = result;
00119 }
```

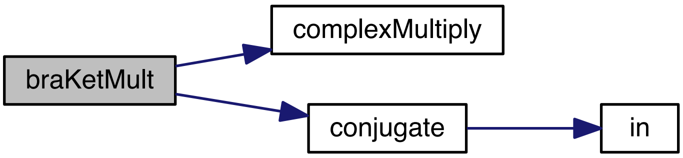
Here is the call graph for this function:

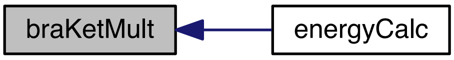
4.17.1.5 __device__ double complexMagnitudeSquared (double2)

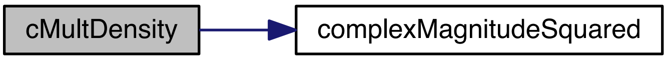
Definition at line 69 of file kernels.cu.

Referenced by cMultDensity(), and energyCalc().


angularOp
getGid3d3d


braKetMult
complexMultiply
conjugate
in


braKetMult
energyCalc


cMultDensity
complexMagnitudeSquared

```
00069                                                              {
00070     return in.x*in.x + in.y*in.y;
00071 }
```

Here is the caller graph for this function:

4.17.1.6 __device__ double2 conjugate (double2 *in*)

Definition at line 51 of file kernels.cu.

References in(), and result.

Referenced by braKetMult().

```
00051                                        {
00052     double2 result = in;
00053     result.y = -result.y;
00054     return result;
00055 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.17.1.7 __global__ void energyCalc (double2 * *wfc,* double2 * *op,* double *dt,* double2 * *energy,* int *gnd_state,* int *op_space,* double *sqrt_omegaz_mass*)

Definition at line 188 of file kernels.cu.

References braKetMult(), complexMagnitudeSquared(), dt, gDenConst, getGid3d3d(), HBAR, realCompMult(), result, and x.

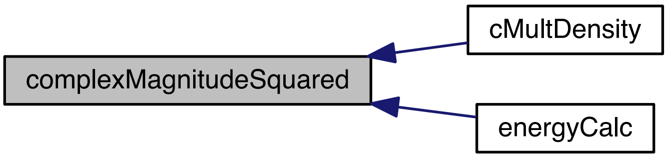
```
00188
                                {
00189     unsigned int gid = getGid3d3d();
00190     double hbar_dt = HBAR/dt;
00191     double g_local = 0.0;
00192     double2 result;
00193     double opLocal;
00194     if(op_space)
00195         g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(
      wfc[gid]);
00196     if(!gnd_state){
00197         opLocal = -log(op[gid].x + g_local)*hbar_dt;
00198     }
00199     else{
00200         opLocal = cos(op[gid].x + g_local)*hbar_dt;
00201     }
00202     result = braKetMult(wfc[gid], realCompMult(opLocal,
      wfc[gid]));
00203     //printf("oplocal=%e    Resx=%e Resy=%e\n",opLocal,result.x,result.y);
00204     energy[gid].x += result.x;
00205     energy[gid].y += result.y;
00206 }
```

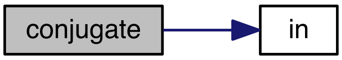
Here is the call graph for this function:

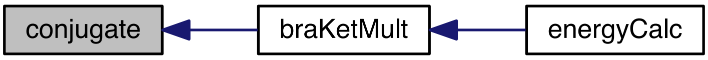
4.17.1.8 __device__ unsigned int getBid3d3d ( )

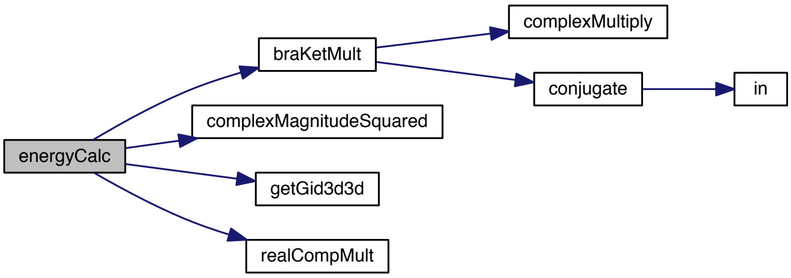
Definition at line 41 of file kernels.cu.

```
00041                                    {
00042     return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00043 }
```


complexMagnitudeSquared
cMultDensity
energyCalc


conjugate
in


conjugate
braKetMult
energyCalc


energyCalc
braKetMult
complexMultiply
conjugate
in
complexMagnitudeSquared
getGid3d3d
realCompMult

4.17.1.9 __device__ unsigned int getGid3d3d ( )

Definition at line 10 of file gpu_functions.cu.

Referenced by angularOp(), cMult(), energyCalc(), multipass(), pSum(), pSumT(), scalarDiv(), scalarDiv_wfcNorm(), scalVecMult_d2d(), scalVecMult_d2d2(), scalVecMult_dd(), scalVecMult_ii(), vecVecMult_d2d(), vecVecMult_d2d2(), vecVecMult_dd(), and vecVecMult_ii().

```
00010                                       {
00011     int gid = blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
      threadIdx.y ) + blockIdx.x ) + threadIdx.x;
00012     return gid;
00013 }
```

Here is the caller graph for this function:

4.17.1.10 __device__ unsigned int getTid3d3d ( )

Definition at line 47 of file kernels.cu.

```
00047                                       {
00048     return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) )  + threadIdx.y )  +
      threadIdx.x;
00049 }
```

4.17.1.11 __global__ void multipass (cufftDoubleComplex ∗ , cufftDoubleComplex ∗ , int)

4.17.1.12 __global__ void pinVortex (cufftDoubleComplex ∗ , cufftDoubleComplex ∗ , cufftDoubleComplex ∗)

4.17.1.13 __global__ void pSum (double ∗ *in1*, double ∗ *output*, int *pass*)

Routine for parallel summation.

Can be looped over from host.

Definition at line 234 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
00234                                                               {
00235         unsigned int tid = threadIdx.x;
00236         unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
00237         unsigned int gid = getGid3d3d();
00238         extern __shared__ double sdata2[];
00239         for(int i = blockDim.x>>1; i > 0; i>>=1){
00240                 if(tid < blockDim.x>>1){
00241                         sdata2[tid] += sdata2[tid + i];
00242                 }
00243                 __syncthreads();
00244         }
00245         if(tid==0){
00246                 output[bid] = sdata2[0];
00247         }
00248 }
```

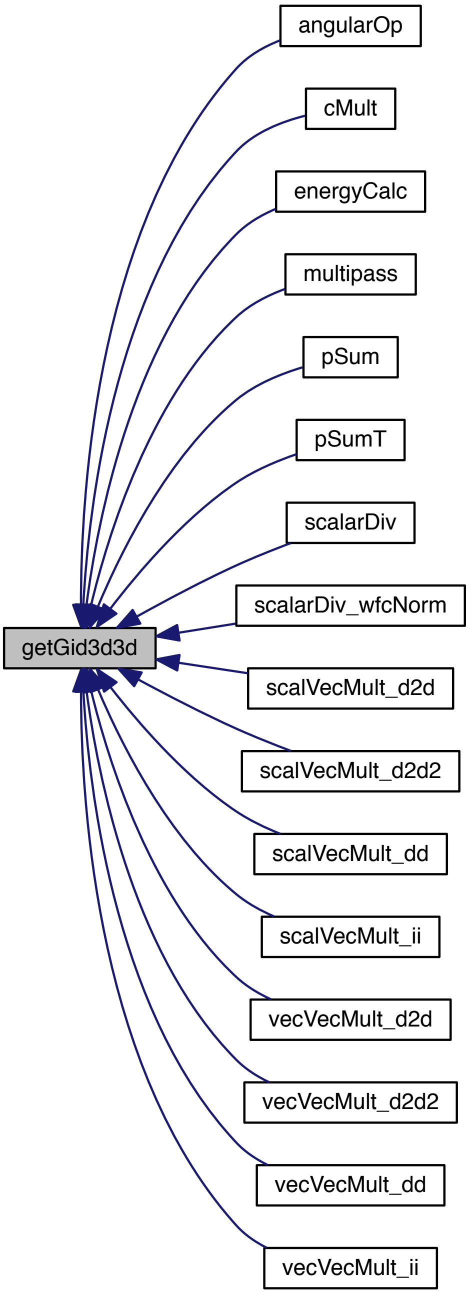
Here is the call graph for this function:

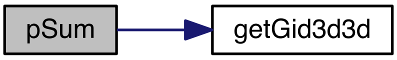
4.17.1.14 __host__ __device__ double2 realCompMult (double *scalar*, double2 *comp*)

Definition at line 180 of file gpu_functions.cu.

References result.

Referenced by energyCalc(), scalVecMult_d2d(), and vecVecMult_d2d().


angularOp
cMult
energyCalc
multipass
pSum
pSumT
scalarDiv
scalarDiv_wfcNorm
getGid3d3d
scalVecMult_d2d
scalVecMult_d2d2
scalVecMult_dd
scalVecMult_ii
vecVecMult_d2d
vecVecMult_d2d2
vecVecMult_dd
vecVecMult_ii


pSum
getGid3d3d

```
00180                                                                 {
00181     double2 result;
00182     result.x = rl*cmp1.x;
00183     result.y = rl*cmp1.y;
00184     return result;
00185 }
```

Here is the caller graph for this function:

4.17.1.15 __global__ void reduce (double2 ∗ , double ∗)

4.17.1.16 __global__ void scalarDiv (double2 ∗ *in,* double *factor,* double2 ∗ *out*)

Divides both components of vector type "in", by the value "factor".

Results given with "out"

Definition at line 125 of file kernels.cu.

References getGid3d3d(), and result.

```
00125                                                                    {
00126     double2 result;
00127     //extern __shared__ double2 tmp_in[];
00128     unsigned int gid = getGid3d3d();
00129     result.x = (in[gid].x*factor);
00130     result.y = (in[gid].y*factor);
00131     out[gid] = result;
00132 }
```

Here is the call graph for this function:

4.17.1.17 __global__ void scalarDiv1D (double2 ∗ , double2 ∗)

4.17.1.18 __global__ void scalarDiv2D (double2 ∗ , double2 ∗)

4.17.1.19 __global__ void scalarDiv_wfcNorm (double2 ∗ , double , double2 ∗ , double2 ∗)

As above, but normalises for wfc.

Definition at line 137 of file kernels.cu.

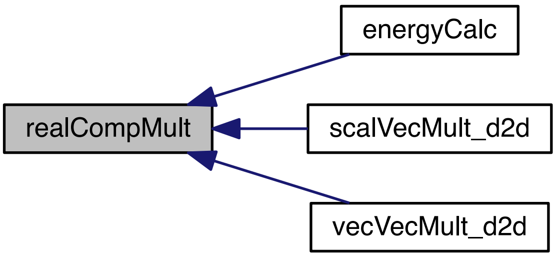
References getGid3d3d(), result, x, and y.

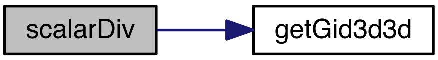
```
00137                                                                                        {
00138     unsigned int gid = getGid3d3d();
00139     double2 result;
00140     double norm = sqrt((pSum[0].x + pSum[0].y)*dr);
00141     result.x = (in[gid].x/norm);
00142     result.y = (in[gid].y/norm);
00143     out[gid] = result;
00144 }
```

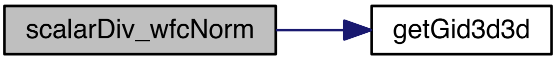
Here is the call graph for this function:

4.18 kernels.h

```
00001 /*
00002 * kernels.h - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
```


realCompMult
energyCalc
scalVecMult_d2d
vecVecMult_d2d


scalarDiv
getGid3d3d


scalarDiv_wfcNorm
getGid3d3d

```
 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
 * License for more details. You should have received a copy of the GNU
 * Lesser General Public License along with this library; if not, write
 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
 * Boston, MA 02111-1307 USA
 */

#ifndef KERNELS_H
#define KERNELS_H
#include<stdio.h>
/* CUDA function declarations */
__device__ unsigned int getGid3d3d();
__device__ unsigned int getBid3d3d();
__device__ unsigned int getTid3d3d();
__device__ double complexMagnitudeSquared(double2);
__device__ double complexMagnitudeSquared(double2);
__global__ void cMult(cufftDoubleComplex*, cufftDoubleComplex*, cufftDoubleComplex*);
__global__ void pinVortex(cufftDoubleComplex*, cufftDoubleComplex*, cufftDoubleComplex*);
__global__ void cMultDensity(double2*, double2*, double2*, double, double,double, int, int);
__global__ void scalarDiv(double2*, double, double2*);
__global__ void scalarDiv1D(double2*, double2*);
__global__ void scalarDiv2D(double2*, double2*);
__global__ void scalarDiv_wfcNorm(double2*, double, double2*, double2*);
__global__ void reduce(double2*, double*);
__global__ void multipass(cufftDoubleComplex*, cufftDoubleComplex*, int);
__global__ void angularOp(double, double, double2*, double*, double2*);


//#####################################################################
//

__device__ double2 conjugate(double2 in);
__device__ double2 realCompMult(double scalar, double2 comp);
__global__ void energyCalc(double2 *wfc, double2 *op, double dt, double2 *energy, int gnd_state, int op_space, double sqrt_omegaz_mass);
inline __device__ double2 braKetMult(double2 in1, double2 in2);
//template<typename T> __global__ void pSumT(T* in1, T* output, int pass);
__global__ void pSum(double* in1, double* output, int pass);
//template<double> __global__ void pSumT(double* in1, double* output, int pass);

#endif
```

4.19 include/minions.h File Reference

```
#include <cuda.h>
#include <stdio.h>
#include <math.h>
#include <cuda_runtime.h>
#include "tracker.h"
```

Include dependency graph for minions.h: This graph shows which files directly or indirectly include this file:

Functions

- double psi2 (double2)
- double minValue (double ∗, int)
- double maxValue (double ∗, int)
- double sumAvg (double ∗in, int len)
- double fInvSqRt (double)

  id magic hackery
- void coordSwap (struct Vortex ∗vCoords, int src, int dest)
- double complexMag (double2 in)
- double complexMag2 (double2 in)
- double2 complexMult (double2 in1, double2 in2)
- double2 complexScale (double2 comp, double scale)
- double2 conj (double2 c)
- double2 complexDiv (double2 num, double2 den)

4.19.1 Function Documentation

4.19.1.1 double2 complexDiv (double2 *num*, double2 *den*)

Definition at line 101 of file minions.cc.

References hist3d::c, complexMag2(), complexMult(), complexScale(), and conj().

Referenced by findVortex(), and phaseTest().

```
00101                                             {
00102     double2 c = conj(den);
00103     return complexScale(complexMult(num,c),(1.0/
     complexMag2(den)));
00104 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.19.1.2 double complexMag (double2 *in*)

Definition at line 73 of file minions.cc.

Referenced by findVortex(), and phaseTest().

```
00073                                {
00074     return sqrt(in.x*in.x + in.y*in.y);
00075 }
```

Here is the caller graph for this function:

4.19.1.3 double complexMag2 (double2 *in*)

Definition at line 77 of file minions.cc.

Referenced by complexDiv().

```
00077                                 {
00078     return in.x*in.x + in.y*in.y;
00079 }
```

Here is the caller graph for this function:

4.19.1.4 double2 complexMult (double2 *in1*, double2 *in2*)

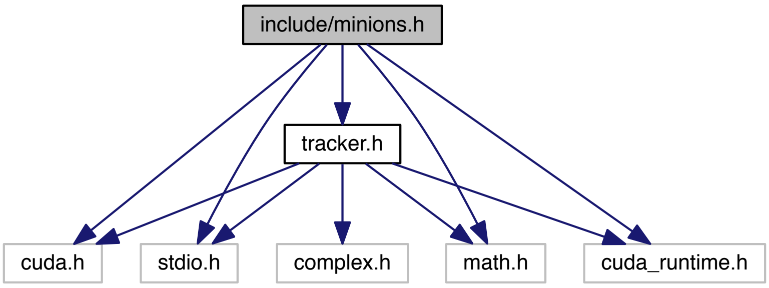
Definition at line 81 of file minions.cc.

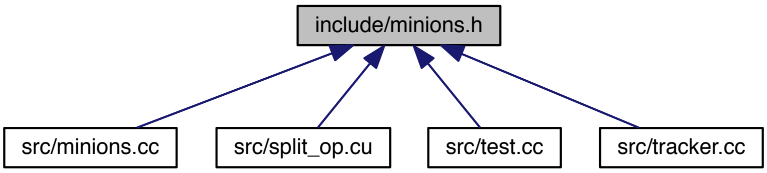
References result.

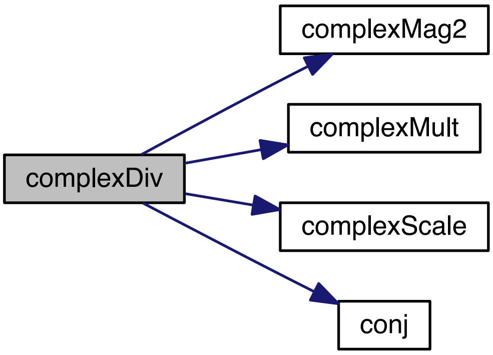
Referenced by complexDiv(), and main().

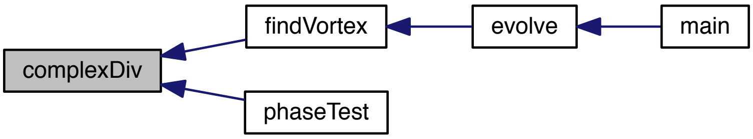
```
00081                                              {
00082     double2 result;
00083     result.x = (in1.x*in2.x - in1.y*in2.y);
00084     result.y = (in1.x*in2.y + in1.y*in2.x);
00085     return result;
00086 }
```

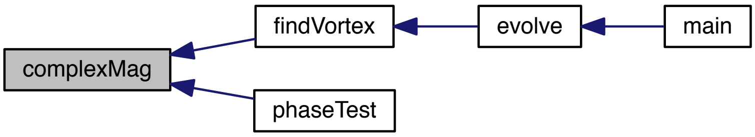
Here is the caller graph for this function:

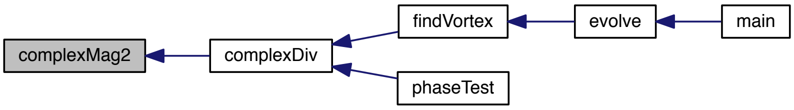

include/minions.h
tracker.h
cuda.h
stdio.h
complex.h
math.h
cuda_runtime.h

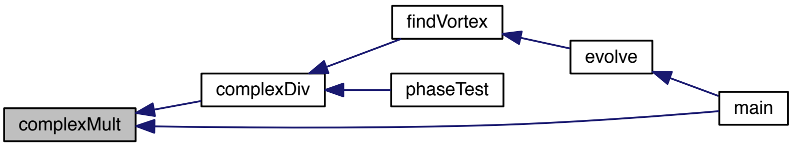

include/minions.h
src/minions.cc
src/split_op.cu
src/test.cc
src/tracker.cc


complexMag2
complexMult
complexDiv
complexScale
conj


findVortex
evolve
main
complexDiv
phaseTest


complexMag
findVortex
evolve
main
phaseTest


complexMag2
complexDiv
findVortex
evolve
main
phaseTest


findVortex
evolve
complexDiv
phaseTest
main
complexMult

4.19.1.5 double2 complexScale (double2 *comp*, double *scale*)

Definition at line 88 of file minions.cc.

References result.

Referenced by complexDiv(), findVortex(), and phaseTest().

```
00088                                                   {
00089     double2 result;
00090     result.x = comp.x*scale;
00091     result.y = comp.y*scale;
00092     return result;
00093 }
```

Here is the caller graph for this function:

4.19.1.6 double2 conj (double2 *c*)

Definition at line 95 of file minions.cc.

```
00095                          {
00096     double2 result = c;
00097     result.y = -result.y;
00098     return result;
00099 }
```

4.19.1.7 void coordSwap (struct **Vortex** ∗ *vCoords*, int *src*, int *dest*)

Definition at line 67 of file minions.cc.

Referenced by main(), and vortArrange().

```
00067                                                            {
00068     struct Vortex d = vCoords[dest];
00069     vCoords[dest] = vCoords[src];
00070     vCoords[src] = d;
00071 }
```

Here is the caller graph for this function:

4.19.1.8 double fInvSqRt (double)

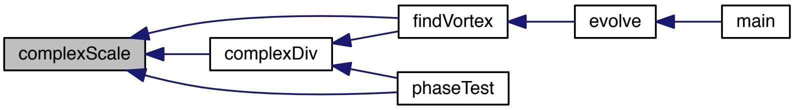
id magic hackery

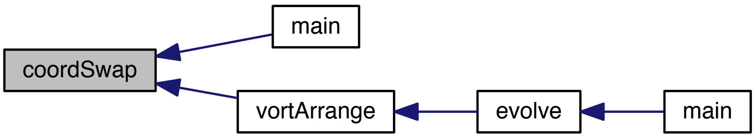
Definition at line 52 of file minions.cc.

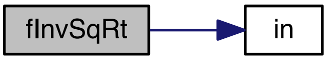
References in(), and l.

```
00052                             {
00053     long long l;
00054     double in05, calc;
00055     const double threehalfs = 1.5;
00056 
00057     in05 = in*0.5;
00058     calc=in;
00059     l = * (long long*) &calc;
00060     l = 0x5fe6eb50c7b537a9LL - (l >> 1);
00061     calc = *(double *) &l;
00062     calc = calc*( 1.5 - (in05*calc*calc) );
00063 
00064     return calc;
00065 }
```

Here is the call graph for this function:


complexScale
complexDiv
findVortex
phaseTest
evolve
main


coordSwap
main
vortArrange
evolve
main


fInvSqRt
in

4.19.1.9 double maxValue (double ∗ , int)

Definition at line 25 of file minions.cc.

References vis::i.

Referenced by findOLMaxima().

```
00025                                       {
00026     double max = grid[0];
00027     for (unsigned int i=1;i<len-1;++i){
00028         if(max<grid[i])
00029             max=grid[i];
00030     }
00031     return max;
00032 }
```

Here is the caller graph for this function:

4.19.1.10 double minValue (double ∗ , int)

Definition at line 34 of file minions.cc.

References vis::i.

Referenced by vortAngle().

```
00034                                       {
00035     double min = grid[0];
00036     for (unsigned int i=1;i<len-1;++i){
00037         if(min>grid[i])
00038             min=grid[i];
00039     }
00040     return min;
00041 }
```

Here is the caller graph for this function:

4.19.1.11 double psi2 (double2)

Definition at line 21 of file minions.cc.

Referenced by evolve().

```
00021                        {
00022     return in.x*in.x + in.y*in.y;
00023 }
```

Here is the caller graph for this function:

4.19.1.12 double sumAvg (double ∗ *in,* int *len*)

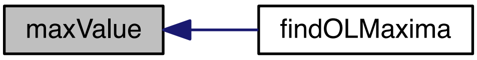
Definition at line 43 of file minions.cc.

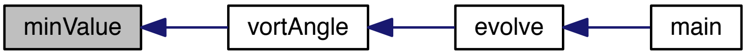
References vis::i.

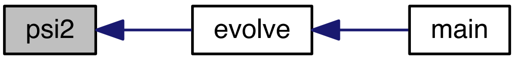
Referenced by evolve().

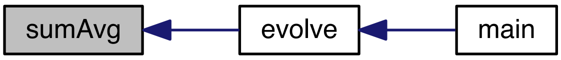
```
00043                                    {
00044     double avg = 0.0;
00045 
00046     for (unsigned int i=0; i<len; ++i){
00047         avg += in[i];
00048     }
00049     return avg/len;
00050 }
```

Here is the caller graph for this function:


maxValue
findOLMaxima


minValue
vortAngle
evolve
main


psi2
evolve
main


sumAvg
evolve
main

4.20 minions.h

```
/*
* minions.h - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#ifndef MINIONS_H
#define MINIONS_H

#include <cuda.h>
#include <stdio.h>
#include <math.h>
#include <cuda_runtime.h>
#include "tracker.h"

/* Returns |x|^2 of the double2 arg*/
double psi2(double2);

/* Returns the minimumi and maximum values in the array*/
double minValue(double*,int);
double maxValue(double*,int);

/* Computes average of the array*/
double sumAvg(double* in, int len);

double fInvSqRt(double);
//float fInvSqRt(float);

void coordSwap(struct Vortex *vCoords, int src, int dest);

double complexMag(double2 in);
double complexMag2(double2 in);
double2 complexMult(double2 in1, double2 in2);
double2 complexScale(double2 comp, double scale);
double2 conj(double2 c);
double2 complexDiv(double2 num, double2 den);
#endif
```

4.21 include/split_op.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <cufft.h>
#include <ctype.h>
#include <getopt.h>
```

Include dependency graph for split_op.h: This graph shows which files directly or indirectly include this file:

Functions

- int isError (int, char ∗)
- void writeOut (char ∗, char ∗, double2 ∗, int, int)
- void parSum (double2 ∗, double2 ∗, int, int, int)

- void optLatSetup (int2 centre, double ∗V, struct Vortex ∗vArray, int num_vortices, double theta_opt, double intensity, double ∗v_opt, double ∗x, double ∗y)
- double energy_angmom (double ∗Energy, double ∗Energy_gpu, double2 ∗V_op, double2 ∗K_op, double dx, double dy, double2 ∗gpuWfc, int gState)

    Calculates energy and angular momentum of current state.

Variables

- struct Params ∗ paramS
- cudaError_t err
- cufftResult result
- int ang_mom = 0
- int gpe = 0
- double mass
- double a_s
- double omegaX
- double omegaY
- double omegaZ
- double xi
- double dt
- double gdt
- int xDim
- int yDim
- int read_wfc
- int print
- int write_it
- long gsteps
- long esteps
- long atoms
- double ∗ x
- double ∗ y
- double ∗ xp
- double ∗ yp
- double ∗ px
- double ∗ py
- double dx
- double dy
- double xMax
- double yMax
- cufftHandle plan_2d
- cufftHandle plan_1d
- cufftDoubleComplex ∗ wfc
- cufftDoubleComplex ∗ wfc0
- cufftDoubleComplex ∗ wfc_backup
- cufftDoubleComplex ∗ GK
- cufftDoubleComplex ∗ GV_half
- cufftDoubleComplex ∗ GV
- cufftDoubleComplex ∗ EK
- cufftDoubleComplex ∗ EV
- cufftDoubleComplex ∗ EV_opt
- cufftDoubleComplex ∗ GxPy
- cufftDoubleComplex ∗ GyPx
- cufftDoubleComplex ∗ ExPy
- cufftDoubleComplex ∗ EyPx

- cufftDoubleComplex * EappliedField
- double * Energy
- double * Energy_gpu
- double * r
- double * Phi
- double * V
- double * V_opt
- double * K
- double * xPy
- double * yPx
- double * xPy_gpu
- double * yPx_gpu
- cufftDoubleComplex * wfc_gpu
- cufftDoubleComplex * K_gpu
- cufftDoubleComplex * V_gpu
- cufftDoubleComplex * par_sum
- cudaStream_t streamA
- cudaStream_t streamB
- cudaStream_t streamC
- cudaStream_t streamD
- double interaction
- double laser_power
- dim3 grid
- int threads
- double l

4.21.1 Function Documentation

4.21.1.1 double energy_angmom (double * *Energy*, double * *Energy_gpu*, double2 * *V_op*, double2 * *K_op*, double *dx*, double *dy*, double2 * *gpuWfc*, int *gState*)

Calculates energy and angular momentum of current state.

Definition at line 628 of file split_op.cu.

References vis::i, result, xDim, and yDim.

```
00628
                                        {
00629     double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
00630     double result=0;
00631
00632     for (int i=0; i < xDim*yDim; ++i){
00633         Energy[i] = 0.0;
00634     }
00635
00636
00637 /*  cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);
00638
00639     energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
00640     result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
00641     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00642
00643     energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
00644     result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
00645     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00646
00647     err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00648
00649     for(int i=0; i<xDim*yDim; i++){
00650         result += energy[i].x;
00651         //printf("En=%E\n",result*dx*dy);
00652     }
00653     return result*dx*dy;
00654 */
00655
00656 }
```

4.21.1.2 int isError (int , char ∗)

Definition at line 42 of file split_op.cu.

References result.

```
00042                             {
00043     if(result!=0){printf("Error has occurred for method %s with return type %d\n",
      c,result);
00044         exit(result);
00045     }
00046     return result;
00047 }
```

4.21.1.3 void optLatSetup (int2 *centre,* double ∗ *V,* struct Vortex ∗ *vArray,* int *num_vortices,* double *theta_opt,* double *intensity,* double ∗ *v_opt,* double ∗ *x,* double ∗ *y*)

4.21.1.4 void parSum (double2 ∗ , double2 ∗ , int , int , int)

Definition at line 578 of file split_op.cu.

References dx, dy, threads, and yDim.

Referenced by evolve().

```
00578                                                                                   {
00579         int grid_tmp = xDim*yDim;
00580         int block = grid_tmp/threads;
00581         int thread_tmp = threads;
00582         int pass = 0;
00583         while((double)grid_tmp/threads > 1.0){
00584             if(grid_tmp == xDim*yDim){
00585                 multipass<<<block,threads,threads*sizeof(double2)>>>(&gpuWfc[0],&gpuParSum[0],pass);
00586             }
00587             else{
00588                 multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass
      );
00589             }
00590             grid_tmp /= threads;
00591             block = (int) ceil((double)grid_tmp/threads);
00592             pass++;
00593         }
00594         thread_tmp = grid_tmp;
00595         multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00596         scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
00597 }
```

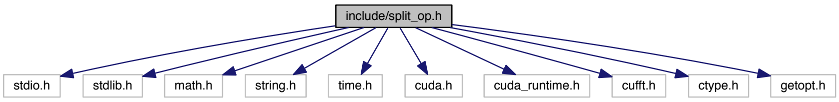
Here is the caller graph for this function:

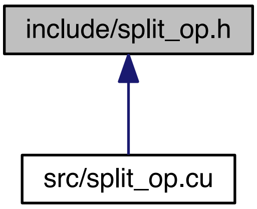
4.21.1.5 void writeOut (char ∗ , char ∗ , double2 ∗ , int , int)

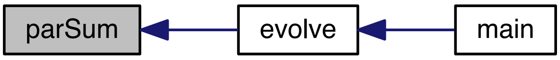
Definition at line 109 of file fileIO.cc.

References vis_ev::f, FILE, vis::i, x, and y.

```
00109                                                                               {
00110     FILE *f;
00111     sprintf (buffer, "%s_%d", file, step);
00112     f = fopen (buffer,"w");
00113     int i;
00114     for (i = 0; i < length; i++)
00115         fprintf (f, "%.16e\n",data[i].x);
00116     fclose (f);
00117
00118     sprintf (buffer, "%si_%d", file, step);
00119     f = fopen (buffer,"w");
00120     for (i = 0; i < length; i++)
00121         fprintf (f, "%.16e\n",data[i].y);
00122     fclose (f);
00123 }
```


include/split_op.h
stdio.h
stdlib.h
math.h
string.h
time.h
cuda.h
cuda_runtime.h
cufft.h
ctype.h
getopt.h


include/split_op.h
src/split_op.cu


parSum
evolve
main

4.21.2 Variable Documentation

4.21.2.1 double a_s

Definition at line 50 of file split_op.h.

Referenced by evolve(), and initialise().

4.21.2.2 int ang_mom = 0

Definition at line 46 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.3 long atoms

Definition at line 58 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.4 double dt

Definition at line 54 of file split_op.h.

Referenced by delta_define(), energyCalc(), evolve(), initialise(), optLatSetup(), and parseArgs().

4.21.2.5 double dx

Definition at line 59 of file split_op.h.

Referenced by delta_define(), evolve(), initialise(), main(), optLatSetup(), parSum(), and sigVOL().

4.21.2.6 double dy

Definition at line 59 of file split_op.h.

Referenced by evolve(), initialise(), main(), and parSum().

4.21.2.7 cufftDoubleComplex ∗ EappliedField

Definition at line 65 of file split_op.h.

Referenced by initialise().

4.21.2.8 cufftDoubleComplex ∗ EK

Definition at line 65 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.9 double∗ Energy

Definition at line 66 of file split_op.h.

Referenced by initialise().

4.21.2.10 double ∗ Energy_gpu

Definition at line 66 of file split_op.h.

Referenced by initialise().

4.21.2.11 cudaError_t err

Definition at line 42 of file split_op.h.

Referenced by main().

4.21.2.12 long esteps

Definition at line 58 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.13 cufftDoubleComplex ∗ EV

Definition at line 65 of file split_op.h.

Referenced by evolve(), initialise(), and main().

4.21.2.14 cufftDoubleComplex ∗ EV_opt

Definition at line 65 of file split_op.h.

Referenced by delta_define(), evolve(), initialise(), and optLatSetup().

4.21.2.15 cufftDoubleComplex ∗ ExPy

Definition at line 65 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.16 cufftDoubleComplex ∗ EyPx

Definition at line 65 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.17 double gdt

Definition at line 54 of file split_op.h.

Referenced by evolve(), initialise(), and parseArgs().

4.21.2.18 cufftDoubleComplex ∗ GK

Definition at line 65 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.19 int gpe = 0

Definition at line 47 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.20 dim3 grid

Definition at line 79 of file split_op.h.

Referenced by initialise().

4.21.2.21 long gsteps

Definition at line 58 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.22 cufftDoubleComplex $*$ GV

Definition at line 65 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.23 cufftDoubleComplex $*$ GV_half

Definition at line 65 of file split_op.h.

4.21.2.24 cufftDoubleComplex $*$ GxPy

Definition at line 65 of file split_op.h.

4.21.2.25 cufftDoubleComplex $*$ GyPx

Definition at line 65 of file split_op.h.

4.21.2.26 double interaction

Definition at line 75 of file split_op.h.

Referenced by evolve(), and parseArgs().

4.21.2.27 double $*$ K

Definition at line 66 of file split_op.h.

Referenced by initialise().

4.21.2.28 cufftDoubleComplex $*$ K_gpu

Definition at line 69 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.29 double l

Definition at line 83 of file split_op.h.

Referenced by fInvSqRt(), initialise(), main(), and parseArgs().

4.21.2.30 double laser_power

Definition at line 76 of file split_op.h.

Referenced by parseArgs().

4.21.2.31 double mass

Definition at line 50 of file split_op.h.

Referenced by cMultDensity(), evolve(), and initialise().

4.21.2.32 double omegaX

Definition at line 50 of file split_op.h.

Referenced by evolve(), main(), and parseArgs().

4.21.2.33 double omegaY

Definition at line 50 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.34 double omegaZ

Definition at line 50 of file split_op.h.

Referenced by evolve(), initialise(), and parseArgs().

4.21.2.35 cufftDoubleComplex ∗ par_sum

Definition at line 69 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.36 struct Params∗ paramS

Definition at line 35 of file split_op.cu.

4.21.2.37 double ∗ Phi

Definition at line 66 of file split_op.h.

Referenced by initialise().

4.21.2.38 cufftHandle plan_1d

Definition at line 62 of file split_op.h.

Referenced by evolve(), and initialise().

4.21.2.39 cufftHandle plan_2d

Definition at line 62 of file split_op.h.

Referenced by evolve(), and initialise().

4.21.2.40 int print

Definition at line 57 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.41 double ∗ px

Definition at line 59 of file split_op.h.

4.21.2.42 double ∗ py

Definition at line 59 of file split_op.h.

4.21.2.43 double ∗ r

Definition at line 66 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.44 int read_wfc

Definition at line 57 of file split_op.h.

Referenced by main(), and parseArgs().

4.21.2.45 cufftResult result

Definition at line 43 of file split_op.h.

Referenced by angularOp(), cMult(), cMultDensity(), complexMult(), complexMultiply(), complexScale(), conj(), conjugate(), energy_angmom(), energyCalc(), evolve(), initialise(), isError(), phaseTest(), realCompMult(), scalarDiv(), scalarDiv_wfcNorm(), sepAvg(), and vortSepAvg().

4.21.2.46 cudaStream_t streamA

Definition at line 72 of file split_op.h.

4.21.2.47 cudaStream_t streamB

Definition at line 72 of file split_op.h.

4.21.2.48 cudaStream_t streamC

Definition at line 72 of file split_op.h.

4.21.2.49 cudaStream_t streamD

Definition at line 72 of file split_op.h.

4.21.2.50 int threads

Definition at line 80 of file split_op.h.

Referenced by initialise(), and parSum().

4.21.2.51 double ∗ V

Definition at line 66 of file split_op.h.

Referenced by delta_define(), and initialise().

4.21.2.52 cufftDoubleComplex ∗ V_gpu

Definition at line 69 of file split_op.h.

Referenced by evolve(), initialise(), and main().

4.21.2.53 double ∗ V_opt

Definition at line 66 of file split_op.h.

Referenced by evolve(), initialise(), and main().

4.21.2.54 cufftDoubleComplex∗ wfc

Definition at line 65 of file split_op.h.

Referenced by evolve(), initialise(), and main().

4.21.2.55 cufftDoubleComplex ∗ wfc0

Definition at line 65 of file split_op.h.

4.21.2.56 cufftDoubleComplex ∗ wfc_backup

Definition at line 65 of file split_op.h.

Referenced by initialise().

4.21.2.57 cufftDoubleComplex∗ wfc_gpu

Definition at line 69 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.58 int write_it

Definition at line 57 of file split_op.h.

Referenced by evolve(), and parseArgs().

4.21.2.59 double∗ x

Definition at line 59 of file split_op.h.

Referenced by cMultDensity(), energyCalc(), evolve(), hdfWriteComplex(), initialise(), main(), matTrans(), scalar←Div_wfcNorm(), writeOut(), and writeOutInt2().

4.21.2.60 int xDim

Definition at line 57 of file split_op.h.

Referenced by delta_define(), energy_angmom(), evolve(), hdfWriteComplex(), hdfWriteDouble(), initialise(), main(), matTrans(), olPos(), optLatSetup(), parseArgs(), and vortPos().

4.21.2.61 double xi

Definition at line 51 of file split_op.h.

Referenced by evolve().

4.21.2.62 double xMax

Definition at line 59 of file split_op.h.

Referenced by initialise().

4.21.2.63 double ∗ xp

Definition at line 59 of file split_op.h.

Referenced by initialise().

4.21.2.64 double ∗ xPy

Definition at line 66 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.65 double ∗ xPy_gpu

Definition at line 66 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.66 double ∗ y

Definition at line 59 of file split_op.h.

Referenced by cMultDensity(), findVortex(), hdfWriteComplex(), initialise(), main(), matTrans(), olPos(), phaseTest(), scalarDiv_wfcNorm(), writeOut(), and writeOutInt2().

4.21.2.67 int yDim

Definition at line 57 of file split_op.h.

Referenced by delta_define(), energy_angmom(), evolve(), initialise(), main(), optLatSetup(), parseArgs(), parSum(), and readIn().

4.21.2.68 double yMax

Definition at line 59 of file split_op.h.

Referenced by initialise().

4.21.2.69 double ∗ yp

Definition at line 59 of file split_op.h.

Referenced by initialise().

4.21.2.70 double ∗ yPx

Definition at line 66 of file split_op.h.

Referenced by initialise(), and main().

4.21.2.71 double ∗ yPx_gpu

Definition at line 66 of file split_op.h.

Referenced by initialise(), and main().

4.22 split_op.h

```
00001 /*
00002 * split_op.h - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
00011 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
00012 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00013 * License for more details. You should have received a copy of the GNU
00014 * Lesser General Public License along with this library; if not, write
00015 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
00016 * Boston, MA 02111-1307 USA
00017 */
00018
00019 #ifndef SPLIT_OP_H
00020 #define SPLIT_OP_H
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <math.h>
00025 #include <string.h>
00026 #include <time.h>
00027 #include <cuda.h>
00028 #include <cuda_runtime.h>
00029 #include <cufft.h>
00030 #include <ctype.h>
00031 #include <getopt.h>
00032 #ifdef __linux
00033     #include<omp.h>
00034 #elif __APPLE__
00035     //printf("OpenMP support disabled due to Clang/LLVM being behind the trend.",);
```

```
#endif

/* Keep track of all params for reading/writing to file*/
extern struct Params *paramS;

/* Error variable & return variables */
cudaError_t err;
cufftResult result;

/* Define operating modes */
int ang_mom = 0;
int gpe = 0;

/* Allocating global variables */
double mass, a_s, omegaX, omegaY, omegaZ;
double xi; //Healing length minimum value defined at central density.

/* Evolution timestep */
double dt, gdt;

/* Grid dimensions vector. xyz are dim length, w is total grid size (x*y*z) */
int xDim, yDim, read_wfc, print, write_it;
long  gsteps, esteps, atoms;
double *x,*y,*xp,*yp,*px,*py,dx,dy,xMax,yMax;

/* CuFFT plans for forward and inverse. May only need to use 1 for both */
cufftHandle plan_2d, plan_1d;

/* Arrays for storing wavefunction, momentum and position op, etc */
cufftDoubleComplex *wfc, *wfc0, *wfc_backup, *GK, *GV_half, *GV, *EK, *EV, *EV_opt, *GxPy, *GyPx, *ExPy, *EyPx, *EappliedField;
double *Energy, *Energy_gpu, *r, *Phi, *V, *V_opt, *K, *xPy, *yPx, *xPy_gpu, *yPx_gpu;

/* CUDA data buffers for FFT */
cufftDoubleComplex *wfc_gpu, *K_gpu, *V_gpu, *par_sum;

/* CUDA streams */
cudaStream_t streamA, streamB, streamC, streamD;

/* Scaling the interaction */
double interaction;
double laser_power;

/* Define global dim3 and threads for grid and thread dim calculation */
dim3 grid;
int threads;

/* */
double l;
/* Function declarations */
/*
 * arg1 = Function result code from CUDA CUFFT calls.
 * arg2 = String data for name of function called. Prints value to stdout.
 */
int isError(int, char*); //Checks to see if an error has occurred.
void writeOut(char*, char*, double2*, int, int); //Writes out to file

void parSum(double2* , double2* , int , int , int );
void optLatSetup(int2 centre, double* V, struct Vortex *vArray, int num_vortices, double theta_opt, double intensity, double* v_opt, double *x, double *y);

double energy_angmom(double* Energy, double* Energy_gpu, double2 *V_op, double2 *K_op, double dx, double dy, double2 *gpuWfc, int gState);
#endif
```

4.23 include/tracker.h File Reference

```
#include <math.h>
#include <stdio.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <complex.h>
```

Include dependency graph for tracker.h: This graph shows which files directly or indirectly include this file:

Classes

- struct Vortex

Functions

- int findVortex (int ∗, double2 ∗, double, int, double ∗, int)
- void vortPos (int ∗marker, struct Vortex ∗vLocation, int xDim, double2 ∗wfc)

  Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.
- void olPos (int ∗marker, int2 ∗vLocation, int xDim)

  Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.
- struct Vortex ∗ vortPosDelta (int ∗cMarker, int2 ∗pMarker, double ∗x, double tolerance, int numVortices, int xDim)
- struct Vortex vortCentre (struct Vortex ∗cArray, int length, int xDim)
- double vortAngle (struct Vortex ∗vortCoords, struct Vortex central, int numVort)
- double vortSepAvg (struct Vortex ∗vArray, struct Vortex centre, int length)
- double sigVOL (int2 ∗vArr, int2 ∗opLatt, double ∗x, int numVort)
- int findOLMaxima (int ∗marker, double ∗V, double radius, int xDim, double ∗x)
- void vortArrange (struct Vortex ∗vCoordsC, struct Vortex ∗vCoordsP, int length)
- int phaseTest (int2 vLoc, double2 ∗wfc, int xDim)

4.23.1 Class Documentation

4.23.1.1 struct Vortex

Definition at line 31 of file tracker.h.

Collaboration diagram for Vortex:

Class Members

int2	coords	
int	sign	
int	wind	

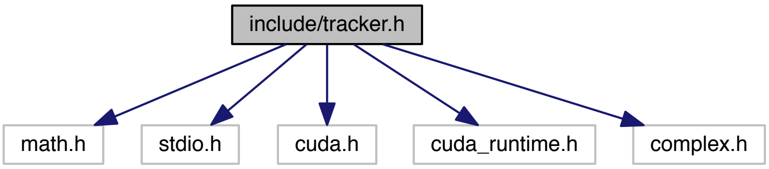
4.23.2 Function Documentation

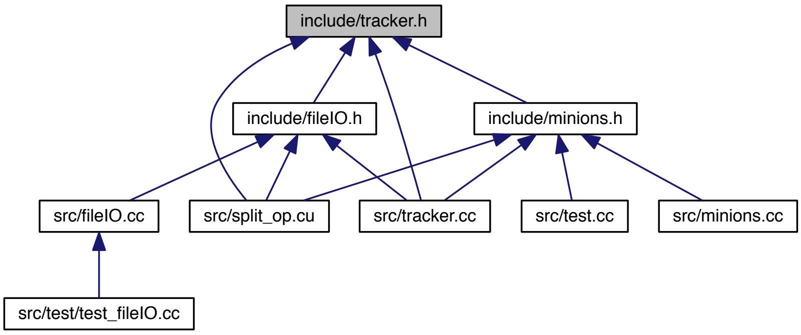
4.23.2.1 int findOLMaxima (int ∗ *marker,* double ∗ *V,* double *radius,* int *xDim,* double ∗ *x*)

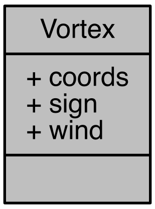
Definition at line 40 of file tracker.cc.

References vis::i, and maxValue().

```
00040                                                                                  {
00041     double gridValues[9];
00042     int2 mIndex[1024];
00043     int2 index;
00044     int i,j,found;
00045     found=0;
00046     for (i=1; i<xDim-1; ++i ){
00047         for(j=1; j<xDim-1;++j){
00048             if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00049                 gridValues[0] = Vopt[(i-1)*xDim + (j-1)];
00050                 gridValues[1] = Vopt[(i-1)*xDim + j];
00051                 gridValues[2] = Vopt[(i-1)*xDim + (j+1)];
00052                 gridValues[3] = Vopt[i*xDim + (j-1)];
00053                 gridValues[4] = Vopt[i*xDim + j];
00054                 gridValues[5] = Vopt[i*xDim + (j+1)];
00055                 gridValues[6] = Vopt[(i+1)*xDim + (j-1)];
00056                 gridValues[7] = Vopt[(i+1)*xDim + j];
```


include/tracker.h
math.h
stdio.h
cuda.h
cuda_runtime.h
complex.h


include/tracker.h
include/fileIO.h
include/minions.h
src/fileIO.cc
src/split_op.cu
src/tracker.cc
src/test.cc
src/minions.cc
src/test/test_fileIO.cc


Vortex
+ coords
+ sign
+ wind

```
00057                 gridValues[8] = Vopt[(i+1)*xDim + (j+1)];
00058                 if(fabs((gridValues[4]-maxValue(gridValues,9))/gridValues[4]) <= 1e-7){
00059                     //printf ("%d,%d\n",i,j);
00060                     (marker)[i*xDim + j] = 1;
00061                     index.x=i;
00062                     index.y=j;
00063                     mIndex[found] = index;
00064                     ++found;
00065                 }
00066             }
00067         }
00068     }
00069     return found;
00070 }
```

Here is the call graph for this function:

4.23.2.2 int findVortex (int ∗ , double2 ∗ , double , int , double ∗ , int)

Definition at line 110 of file tracker.cc.

References complexDiv(), complexMag(), complexScale(), vis::i, PI, and y.

Referenced by evolve().

```
00110                                                                                             {
00111         double2 *g = (double2*) malloc(sizeof(double2)*4);
00112         double *phiDelta = (double*) malloc(sizeof(double)*4);
00113     int i,j,found;
00114     int cond_x, cond_y;
00115     cond_x = 0; cond_y = 0;
00116     found = 0;
00117     long rnd_value = 0;
00118     double sum = 0.0;
00119         for ( i=0; i < xDim-1; ++i ){
00120                 for( j=0; j < xDim-1; ++j ){
00121                         if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00122                                 g[0] = complexScale( complexDiv(
      wfc[i*xDim + j],           wfc[(i+1)*xDim + j] ) ,    (complexMag(
      wfc[(i+1)*xDim + j]) / complexMag( wfc[i*xDim + j] )));
00123                                 g[1] = complexScale( complexDiv(
      wfc[(i+1)*xDim + j],       wfc[(i+1)*xDim + (j+1)] ) ,    (complexMag(
      wfc[(i+1)*xDim + (j+1)]) / complexMag( wfc[(i+1)*xDim + j] )));
00124                                 g[2] = complexScale( complexDiv(
      wfc[(i+1)*xDim + (j+1)], wfc[i*xDim + (j+1)] ) ,   (complexMag(
      wfc[i*xDim + (j+1)]) / complexMag( wfc[(i+1)*xDim + (j+1)] )));
00125                                 g[3] = complexScale( complexDiv(
      wfc[i*xDim + (j+1)],       wfc[i*xDim + j] ) ,        (complexMag(
      wfc[i*xDim + j])      / complexMag( wfc[i*xDim + (j+1)] )));
00126
00127                 for (int k=0; k<4; ++k){
00128                     phiDelta[k] = atan2( g[k].y, g[k].x );
00129                     if(phiDelta[k] <= -PI){
00130                         phiDelta[k] += 2*PI;
00131                     }
00132                 }
00133                 sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00134                 rnd_value = lround(sum/(2*PI));
00135                                 if( sum >= 1.9*PI && cond_x <= 0 && cond_y <= 0){
00136                     marker[i*xDim + j] = rnd_value;
00137                     ++found;
00138                     sum = 0.0;
00139                     cond_x = 2; cond_y = 2;
00140                                 }
00141                 else if( sum <= -1.9*PI && cond_x <= 0 && cond_y <= 0 )  {
00142                     marker[i*xDim + j] = -rnd_value;
00143                     ++found;
00144                     sum = 0.0;
00145                     cond_x = 2; cond_y = 2;
00146
00147                 }
00148                 --cond_x;
00149                 --cond_y;
00150                         }
00151                 }
00152         }
00153     return found;
00154 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.23.2.3 void olPos (int ∗ *marker,* int2 ∗ *vLocation,* int *xDim*)

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

Definition at line 158 of file tracker.cc.

References vort::counter, vis::i, xDim, and y.

```
00158                                                   {
00159     int i,j;
00160     unsigned int counter=0;
00161     for(i=0; i<xDim; ++i){
00162         for(j=0; j<xDim; ++j){
00163             if((marker)[i*xDim + j] == 1){
00164                 (olLocation)[ counter ].x=i;
00165                 (olLocation)[ counter ].y=j;
00166                 ++counter;
00167             }
00168         }
00169     }
00170 }
```

4.23.2.4 int phaseTest (int2 *vLoc,* double2 ∗ *wfc,* int *xDim*)

Definition at line 172 of file tracker.cc.

References complexDiv(), complexMag(), complexScale(), PI, result, and y.

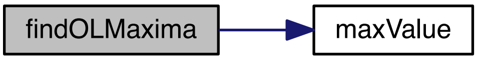
```
00172                                                {
00173     int result = 0;
00174     double2 gridValues[4];
00175     double phiDelta[4];
00176     double sum=0.0;
00177     int i=vLoc.x, j=vLoc.y;
00178     gridValues[0] = complexScale( complexDiv(wfc[i*xDim + j],
      wfc[(i+1)*xDim + j]), (complexMag(wfc[(i+1)*xDim + j])/
      complexMag(wfc[i*xDim + j])));
00179         gridValues[1] = complexScale( complexDiv(wfc[(i+1)*
      xDim + j],wfc[(i+1)*xDim + (j+1)]), (complexMag(wfc[(i+1)*
      xDim + (j+1)])/complexMag(wfc[(i+1)*xDim + j])));
00180         gridValues[2] = complexScale( complexDiv(wfc[(i+1)*
      xDim + (j+1)],wfc[i*xDim + (j+1)]), (complexMag(wfc[i*
      xDim + (j+1)])/complexMag(wfc[(i+1)*xDim + (j+1)])));
00181         gridValues[3] = complexScale( complexDiv(wfc[i*
      xDim + (j+1)],wfc[i*xDim + j]), (complexMag(wfc[i*xDim + j])/
      complexMag(wfc[i*xDim + (j+1)])));
00182
00183     for (int k=0; k<4; ++k){
00184         phiDelta[k] = atan2(gridValues[k].y,gridValues[k].x);
00185                 if(phiDelta[k] <= -PI){
00186                     phiDelta[k] += 2*PI;
00187         }
00188     }
00189     sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00190     if(sum >=1.8*PI){
00191         result = 1;
00192     }
00193     free(gridValues); free(phiDelta);
00194     return result;
00195 }
```

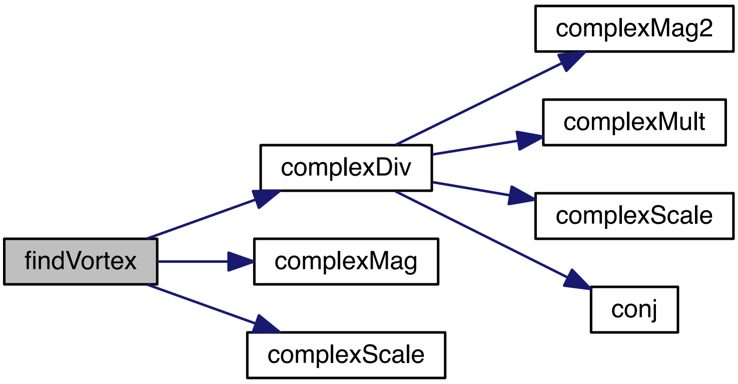
Here is the call graph for this function:

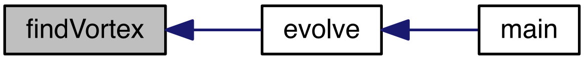
4.23.2.5 double sigVOL (int2 ∗ *vArr,* int2 ∗ *opLatt,* double ∗ *x,* int *numVort*)

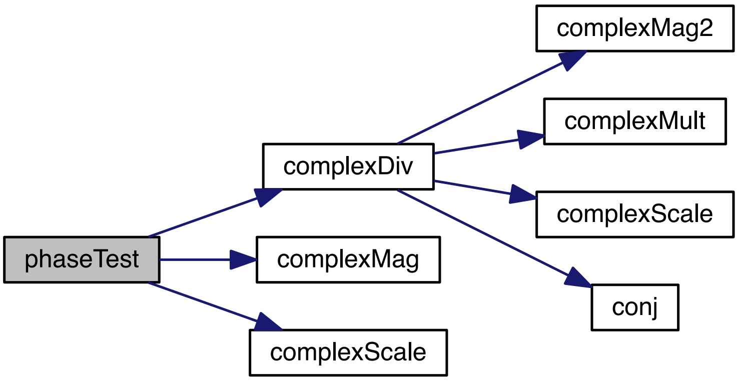
4.23.2.6 double vortAngle (struct Vortex ∗ *vortCoords,* struct Vortex *central,* int *numVort*)

Definition at line 255 of file tracker.cc.


findOLMaxima
maxValue


complexMag2
complexMult
complexDiv
complexScale
findVortex
complexMag
conj
complexScale


findVortex
evolve
main


complexMag2
complexMult
complexDiv
complexScale
phaseTest
complexMag
conj
complexScale

References Vortex::coords, vis::i, minValue(), PI, and Vortex::sign.

Referenced by evolve().

```
00255                                                                                     {
00256     int location;
00257     double sign=1.0;
00258     double minValue=2*512*512;//(pow(central.x - vortCoords[0].x,2) + pow(central.y -
       vortCoords[0].y,2));
00259     for (int i=0; i < numVort; ++i){
00260         if (minValue > (pow(central.x - vortCoords[i].coords.x,2) + pow(central.
      coords.y - vortCoords[i].coords.y,2)) && abs(central.coords.x - vortCoords[i].
      coords.x) > 1e-4 && abs(central.coords.y - vortCoords[i].coords.y) > 1e-4){
00261             minValue = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.
      coords.y - vortCoords[i].coords.y,2));
00262             location = i;
00263         }
00264     }
00265     return PI/2 + atan((vortCoords[location].coords.y - central.coords.y) / (vortCoords[
      location].coords.x - central.coords.x));
00266
00267     //return PI/2 + fmod(atan2(vortCoords[location].y-central.y, vortCoords[location].x - central.x),
       PI/3);
00268     //return PI/2 - sign*acos( ( (central.x - vortCoords[location].x)*(central.x - vortCoords[location].x)
       ) / ( minValue*(central.x - vortCoords[location].x) ) );
00269 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.23.2.7 void vortArrange (struct **Vortex** ∗ *vCoordsC,* struct **Vortex** ∗ *vCoordsP,* int *length*)

Definition at line 217 of file tracker.cc.

References Vortex::coords, coordSwap(), vort::dist(), and vis::i.

Referenced by evolve().

```
00217                                                                                    {
00218     int dist, dist_t;
00219     int i, j, index;
00220     for ( i = 0; i < length; ++i ){
00221         dist = 0x7FFFFFFF; //arbitrary big value
00222         index = i;
00223         for ( j = i; j < length ; ++j){
00224             dist_t = ( (vCoordsP[i].coords.x - vCoordsC[j].coords.x)*(vCoordsP[i].
      coords.x - vCoordsC[j].coords.x) + (vCoordsP[i].coords.y - vCoordsC[j].
      coords.y)*(vCoordsP[i].coords.y - vCoordsC[j].coords.y) );
00225             if(dist > dist_t ){
00226                 dist = dist_t;
00227                 index = j;
00228             }
00229         }
00230         coordSwap(vCoordsC,index,i);
00231     }
00232 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

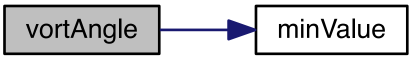
4.23.2.8 struct Vortex vortCentre (struct **Vortex** ∗ *cArray,* int *length,* int *xDim*)

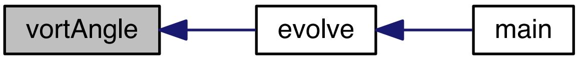
Definition at line 236 of file tracker.cc.

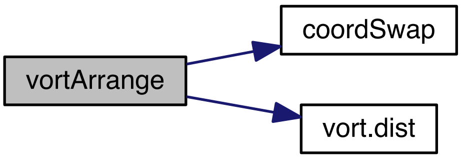
References Vortex::coords, vort::counter, and vis::i.

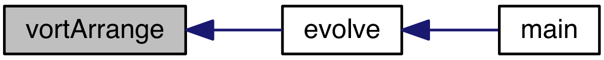
Referenced by evolve().

```
00236                                                                         {
00237     int i, j, counter=0;
00238     int valX, valY;
00239     double valueTest, value = 0.0;
```


vortAngle
minValue


vortAngle
evolve
main


vortArrange
coordSwap
vort.dist


vortArrange
evolve
main

```
00240     valX = (cArray)[0].coords.x - ((xDim/2)-1);
00241     valY = (cArray)[0].coords.y - ((xDim/2)-1);
00242     value = sqrt( valX*valX + valY*valY );//Calcs the sqrt(x^2+y^2) from central position. try to minimise
       this value
00243     for ( i=1; i<length; ++i ){
00244         valX = (cArray)[i].coords.x - ((xDim/2)-1);
00245         valY = (cArray)[i].coords.y - ((xDim/2)-1);
00246         valueTest = sqrt(valX*valX + valY*valY);
00247         if(value > valueTest){
00248             value = valueTest;
00249             counter = i;
00250         }
00251     }
00252     return (cArray)[counter];
00253 }
```

Here is the caller graph for this function:

4.23.2.9 void vortPos (int ∗ *marker,* struct Vortex ∗ *vLocation,* int *xDim,* double2 ∗ *wfc*)

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

Definition at line 198 of file tracker.cc.

References Vortex::coords, vort::counter, vis::i, Vortex::sign, Vortex::wind, and xDim.

Referenced by evolve().

```
00198                                                                           {
00199     int i,j;
00200     unsigned int counter=0;
00201     for(i=0; i<xDim; ++i){
00202         for(j=0; j<xDim; ++j){
00203             if( abs((marker)[i*xDim + j]) >= 1){
00204                 (vLocation)[ counter ].coords.x=i;
00205                 (vLocation)[ counter ].coords.y=j;
00206                 (vLocation)[ counter ].sign = ( signbit(abs(marker[i*xDim + j])) == 0 ) ? 1 : -1;
00207                 (vLocation)[ counter ].wind = abs(marker[i*xDim + j]);
00208                 ++counter;
00209             }
00210         }
00211     }
00212 }
```

Here is the caller graph for this function:

4.23.2.10 struct Vortex∗ vortPosDelta (int ∗ *cMarker,* int2 ∗ *pMarker,* double ∗ *x,* double *tolerance,* int *numVortices,* int *xDim*)

4.23.2.11 double vortSepAvg (struct Vortex ∗ *vArray,* struct Vortex *centre,* int *length*)

Definition at line 26 of file tracker.cc.

References Vortex::coords, and result.

Referenced by evolve(), and optLatSetup().

```
00026                                                                          {
00027     double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00028     double min = 0.0;
00029     int index=0;
00030     min = sqrt( pow(centre.coords.x - vArray[0].coords.x,2) + pow(centre.
      coords.y - vArray[0].coords.y,2));
00031     for (int j=1; j<length; ++j){
00032         if(min > sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.
      coords.y - vArray[j].coords.y,2)) && sqrt( pow(centre.coords.x - vArray[j].
      coords.x,2) + pow(centre.coords.y - vArray[j].coords.y,2)) > 1e-7){
00033             min = sqrt(pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.
      coords.y - vArray[j].coords.y,2));
00034             index = j;
00035         }
00036     }
00037     return min;
00038 }
```

Here is the caller graph for this function:

4.24 tracker.h

```
/*
* tracker.h - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#ifndef TRACKER_H
#define TRACKER_H
#ifdef __linux
    #include<omp.h>
#elif __APPLE__
#endif
#include<math.h>
#include<stdio.h>
#include<cuda.h>
#include<cuda_runtime.h>
#include<complex.h>

struct Vortex{
    int2 coords;
    int sign;
    int wind;
};


int findVortex(int*,double2*, double, int, double*, int);
//void vortPos(int *marker, int2 *vLocation, int xDim, double2* wfc);
void vortPos(int *marker, struct Vortex *vLocation, int xDim, double2* wfc);
void olPos(int *marker, int2 *vLocation, int xDim);
struct Vortex* vortPosDelta(int *cMarker, int2 *pMarker, double* x, double tolerance, int numVortices, int xDim);
struct Vortex vortCentre(struct Vortex *cArray, int length, int xDim);
double vortAngle(struct Vortex *vortCoords, struct Vortex central, int numVort);
double vortSepAvg(struct Vortex *vArray, struct Vortex centre, int length);
double sigVOL(int2 *vArr, int2 *opLatt, double *x, int numVort);
int findOLMaxima(int *marker, double *V, double radius, int xDim, double* x);
void vortArrange(struct Vortex *vCoordsC, struct Vortex *vCoordsP, int length);
int phaseTest(int2 vLoc, double2* wfc, int xDim);
#endif
```

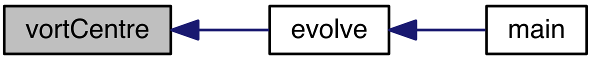
4.25 py/hist3d.py File Reference

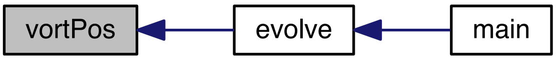
Namespaces

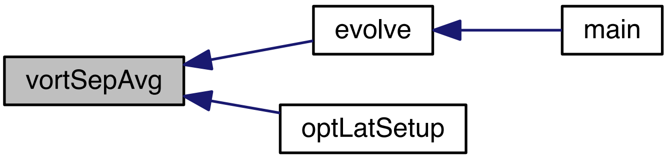
- hist3d

Functions

- def hist3d.plot_xyz_histogram
- def hist3d.plot_hist_pcolor


vortCentre
evolve
main


vortPos
evolve
main


vortSepAvg
evolve
main
optLatSetup

Variables

- tuple hist3d.c = ConfigParser.ConfigParser()
- tuple hist3d.xDim = int(c.getfloat(’Params’,’xDim’))
- tuple hist3d.yDim = int(c.getfloat(’Params’,’yDim’))
- tuple hist3d.gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
- tuple hist3d.evMaxVal = int(c.getfloat(’Params’,’esteps’))
- tuple hist3d.incr = int(c.getfloat(’Params’,’print_out’))
- tuple hist3d.sep = (c.getfloat(’Params’,’dx’))
- tuple hist3d.dx = (c.getfloat(’Params’,’dx’))
- tuple hist3d.dt = (c.getfloat(’Params’,’dt’))
- tuple hist3d.xMax = (c.getfloat(’Params’,’xMax’))
- tuple hist3d.yMax = (c.getfloat(’Params’,’yMax’))
- tuple hist3d.num_vort = int(c.getfloat(’Params’,’Num_vort’))

4.26 hist3d.py

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from numpy import genfromtxt
import math as m
import ConfigParser

c = ConfigParser.ConfigParser()
c.readfp(open(r’Params.dat’))

xDim = int(c.getfloat(’Params’,’xDim’))
yDim = int(c.getfloat(’Params’,’yDim’))
gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
evMaxVal = int(c.getfloat(’Params’,’esteps’))
incr = int(c.getfloat(’Params’,’print_out’))
sep = (c.getfloat(’Params’,’dx’))
dx = (c.getfloat(’Params’,’dx’))
dt = (c.getfloat(’Params’,’dt’))
xMax = (c.getfloat(’Params’,’xMax’))
yMax = (c.getfloat(’Params’,’yMax’))
num_vort = int(c.getfloat(’Params’,’Num_vort’))

sep=1.0
def plot_xyz_histogram(start,fin,incr, barcolor):
    fig = plt.figure()
    ax = Axes3D(fig)
    data =[]
    for i in range(start, fin, incr):
        v_arr=genfromtxt(’vort_lsq_’ + str(i) + ’.csv’,delimiter=’,’ )
        datatmp=[]
        count=0

        for i1 in range(0,v_arr.size/2):
            for i2 in range(i1,v_arr.size/2):
                datatmp.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2  +  abs(v_arr[i1][1]*sep - v_arr[i2][1]*sep)**2 ))
                count = count + 1
        hist=np.histogram(datatmp,bins=np.arange(1.0,m.sqrt(xDim**2 + yDim**2),1.0))
        data.append(hist[:][0])
    """ Takes in a matrix (see structure above) and generate a pseudo-3D histogram by overlaying close, semitransparent bars. """
    for time, occurrence in zip(range(len(data)), data):
        dist = range(len(occurrence))
        barband = range(-45, 45, 5)
        #for modifier in barband:
        ax.bar(dist, occurrence, zs=time, zdir=’y’, color=np.random.rand(3,1), alpha=0.8)
            #ax.bar(current, occurrence, zs=duration+(float(modifier)/100), zdir=’y’, color=np.random.rand(3,1), alpha=0.6)

    ax.set_xlabel(’Dist’)
    ax.set_ylabel(’Time’)
    ax.set_zlabel(’Occurrances’)

    plt.savefig("HIST_N.pdf")
    plt.show()

def plot_hist_pcolor(start,fin,incr, barcolor):
    fig = plt.figure()

    data =[]
    for i in range(start, fin, incr):
        v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',' )
        datatmp=[]
        count=0

        for i1 in range(0,v_arr.size/2):
            for i2 in range(i1,v_arr.size/2):
                m_tmp = m.sqrt(abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2  +  abs(v_arr[i1][1]*sep - v_arr[i2][1]*sep)**2 )
                datatmp.append( m_tmp )
                count = count + 1
        hist=np.histogram(datatmp,bins=np.arange(0.0,240.0,0.1))
        data.append(hist[:][0])

      #  print data
        ax = fig.add_subplot(111)
        ax.imshow(data)
    plt.gca().invert_yaxis()
        ax.set_aspect('auto')
#        plt.jet()
    fig.savefig("HIST_PCOLOR.pdf")

#plot_xyz_histogram(0,100000,100,'b')
#plot_hist_pcolor(0,100000,100,'b')

```

4.27 py/hist_it.py File Reference

Namespaces

- hist_it

4.28 hist_it.py

```
#
# vis.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
```

4.29 py/image_gen.py File Reference

Namespaces

- image_gen

4.30 image_gen.py

4.31 py/observables.py File Reference

Namespaces

- observables

Functions

- def observables.kinertrum
- def observables.dens_struct_fact
- def observables.energy_total
- def observables.energy_kinetic
- def observables.energy_potential
- def observables.ang_mom
- def observables.expec_val_monopole
- def observables.expec_val_quadrupole
- def observables.expec_val_

Variables

- tuple observables.c = ConfigParser.ConfigParser()
- tuple observables.xDim = int(c.getfloat(’Params’,’xDim’))
- tuple observables.yDim = int(c.getfloat(’Params’,’yDim’))
- tuple observables.gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
- tuple observables.evMaxVal = int(c.getfloat(’Params’,’esteps’))
- tuple observables.incr = int(c.getfloat(’Params’,’print_out’))
- tuple observables.sep = (c.getfloat(’Params’,’dx’))
- tuple observables.dx = (c.getfloat(’Params’,’dx’))
- tuple observables.dkx = (c.getfloat(’Params’,’dpx’))
- tuple observables.dt = (c.getfloat(’Params’,’dt’))
- tuple observables.xMax = (c.getfloat(’Params’,’xMax’))
- tuple observables.yMax = (c.getfloat(’Params’,’yMax’))
- tuple observables.num_vort = int(c.getfloat(’Params’,’Num_vort’))
- tuple observables.N = int(c.getfloat(’Params’,’atoms’))
- tuple observables.data = numpy.ndarray(shape=(xDim,yDim))
- tuple observables.x = np.asarray(open(’x_0’).read().splitlines(),dtype=’f8’)
- tuple observables.y = np.asarray(open(’y_0’).read().splitlines(),dtype=’f8’)
- tuple observables.kx = np.asarray(open(’px_0’).read().splitlines(),dtype=’f8’)
- tuple observables.ky = np.asarray(open(’py_0’).read().splitlines(),dtype=’f8’)

4.32 observables.py

```
#
# observables.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012-2014, Lee J. O’Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
import os
from numpy import genfromtxt
import math as m
import matplotlib as mpl
import numpy as np
import scipy as sp
import numpy.matlib
mpl.use(’Agg’)
import multiprocessing as mp
from multiprocessing import Pool
from multiprocessing import Process
from matplotlib.ticker import ScalarFormatter
import matplotlib.pyplot as plt
import ConfigParser
import random as r
from decimal import *
from scipy.spatial import Delaunay

getcontext().prec = 4
c = ConfigParser.ConfigParser()
c.readfp(open(r’Params.dat’))

xDim = int(c.getfloat(’Params’,’xDim’))
yDim = int(c.getfloat(’Params’,’yDim’))
gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
evMaxVal = int(c.getfloat(’Params’,’esteps’))
incr = int(c.getfloat(’Params’,’print_out’))
sep = (c.getfloat(’Params’,’dx’))
dx = (c.getfloat(’Params’,’dx’))
dkx = (c.getfloat(’Params’,’dpx’))
dt = (c.getfloat(’Params’,’dt’))
xMax = (c.getfloat(’Params’,’xMax’))
yMax = (c.getfloat(’Params’,’yMax’))
try:
    num_vort = int(c.getfloat(’Params’,’Num_vort’))
except:
    print ’!num_vort undefined!’
N = int(c.getfloat(’Params’,’atoms’))

data = numpy.ndarray(shape=(xDim,yDim))

x=np.asarray(open(’x_0’).read().splitlines(),dtype=’f8’)
y=np.asarray(open(’y_0’).read().splitlines(),dtype=’f8’)
kx=np.asarray(open(’px_0’).read().splitlines(),dtype=’f8’)
ky=np.asarray(open(’py_0’).read().splitlines(),dtype=’f8’)

#Kinetic energy spectrum = kinertrum
def kinertrum(Psi, dx):
    kxm, kym = np.meshgrid(px,py)
    kMax = np.max(np.max(kx))

    n_r = np.abs(Psi)**2
    cPsi = np.conj(Psi)
    phi = np.angle(Psi)

    ph1 = np.unwrap(phi, axis=0)
    ph2 = np.unwrap(phi, axis=1)

    vel_ph1_x, vel_ph1_y = np.gradient(np1,dx,dy)
    vel_ph2_x, vel_ph2_y = np.gradient(np2,dx,dy)

    v_x = (hbar/mass)*vel_ph1_x;
    v_y = (hbar/mass)*vel_ph2_y;

    u_x = np.multiply(np.abs(Psi),v_x)
    u_y = np.multiply(np.abs(Psi),v_y)

    F_x = (1.0/(2*3.14159))*np.fft.fftn(u_x)
    F_y = (1.0/(2*3.14159))*np.fft.fftn(u_y)

    uc_kx = ( np.multiply(np.multiply(kxm,kxm),F_x) + np.multiply(np.multiply(kxm,kym),F_y))
    uc_ky = ( np.multiply(np.multiply(kym,kxm),F_x) + np.multiply(np.multiply(kym,kym),F_y))

    ui_kx = F_x - uc_kx
    ui_ky = F_y - uc_ky

    uc_x = np.fft.ifftn(uc_kx)
    uc_y = np.fft.ifftn(uc_ky)
    ui_x = np.fft.ifftn(ui_kx)
    ui_y = np.fft.ifftn(ui_ky)

    Ec = 0.5*np.real(np.square(uc_x) + np.square(uc_y))
    Ei = 0.5*np.real(np.square(ui_x) + np.square(ui_y))

    k_bins=np.arange(0,max(np.sqrt(kx**2 + ky**2)),np.sqrt(dkx**2 + dky**2))
    num_bins = len(k_bins)

    for i1 in np.arange(0,num_bins-1):
        iX = np.where(k >=k_bins[i1] & k<k_bins[i1+1])
        Ei_kx = np.sum(np.sum(np.abs(ui_kx[iX]**2*k[iX]))
        Ei_ky = np.sum(np.sum(np.abs(ui_ky[iX]**2*k[iX]))
        Ei_k[i1] = (Ei_kx + Ei_ky)/len(iX)
    np.savetxt(’Ek_i’ + str(i) + ’.csv’,E_k,delimiter=’,’)


def dens_struct_fact(dataName, initValue, finalValue,incr):
    n_k=np.zeros(finalValue/incr)
    n_k_t=np.zeros((finalValue/incr,xDim,yDim),dtype=np.complex128)
    for i in range(initValue,incr*(finalValue/incr),incr):
        if os.path.exists(dataName + ’_’ + str(i)):
            real=open(dataName + ’_’ + str(i)).read().splitlines()
            img=open(dataName + ’i_’ + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
            a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
            a = a_r[:] + 1j*a_i[:]
            n = np.abs(a)**2
            sf = np.fft.fftshift(np.fft.fft2(np.reshape(n,(xDim,yDim))))
            n_k_t[i/incr][:][:] = sf[:][:];
            n_k[i/incr]=(abs(np.sum(np.sum(sf))*dkx**2))

            fig, ax = plt.subplots()
            f = plt.imshow(np.log10(abs(sf)),cmap=plt.get_cmap(’gnuplot2’))
            cbar = fig.colorbar(f)
            plt.gca().invert_yaxis()
            plt.savefig("struct_" + str(i/incr) + ".png",vmin=0,vmax=12,dpi=200)
            plt.close()
            print i/incr

    np.savetxt(’Struct’ + ’.csv’,n_k,delimiter=’,’)
    plt.plot(range(initValue,finalValue,incr),n_k)
    sp.io.savemat(’Struct_t.mat’,mdict={’n_k_t’,n_k_t})
    plt.savefig("Struct.pdf",dpi=200)
    plt.close()

def energy_total(dataName, initValue, finalValue, increment):
    print ’energy’

def energy_kinetic(dataName, initValue, finalValue, increment):
    px1 = np.fft.fftshift(px)
    py1 = np.fft.fftshift(py)
    dk=[]
    dk2[:] = (px1[:]**2 + py1[:]**2)
    Lz = np.zeros( (finalValue/incr))
    for i in range(initValue,incr*(finalValue/incr),incr):
        if os.path.exists(dataName + ’_’ + str(i)):
            real=open(dataName + ’_’ + str(i)).read().splitlines()
            img=open(dataName + ’i_’ + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
            a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
            a = a_r[:] + 1j*a_i[:]
            wfcp = np.fft.fft2(np.reshape(a,(xDim,yDim)))
            conjwfcp = np.conj(wfcp)
            E_k = np.zeros(len(px1))
            for ii in range(0,len(px1)):
                E_k[ii] = np.sum( np.sum( np.multiply(wfcp,conjwfcp) )  )*dk2[ii]

        np.savetxt(’E_k_’ + str(i) + ’.csv’,E_k,delimiter=’,’)
        print i

def energy_potential(dataName, initValue, finalValue, increment):
    print ’energy’

def ang_mom(dataName, initValue, finalValue, incr, ev_type, imgdpi):
    xm, ym = np.meshgrid(x,y)
    pxm, pym = np.meshgrid(px,py)
    dx2=dx**2
    Lz = np.zeros( (finalValue/incr))
    for i in range(initValue,incr*(finalValue/incr),incr):
        if os.path.exists(dataName + ’_’ + str(i)):
            real=open(dataName + ’_’ + str(i)).read().splitlines()
            img=open(dataName + ’i_’ + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
            a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
            a = a_r[:] + 1j*a_i[:]
            wfc = np.reshape(a,(xDim,yDim))
            conjwfc = np.conj(wfc)

            wfc_ypx = np.multiply(ym,np.fft.ifft(np.multiply(pxm,np.fft.fft(wfc,axis=1)),axis=1))
            wfc_xpy = np.multiply(xm,np.fft.ifft(np.multiply(pym,np.fft.fft(wfc,axis=0)),axis=0))
            result = np.sum( np.sum( np.multiply(conjwfc,wfc_xpy - wfc_ypx) ) )*dx2
        else:
            print "Skipped " + dataName + "_"+ str(i)
            result = np.nan

        print i, incr
        Lz[(i/incr)] = np.real(result)
    type=""
    if ev_type == 0:
        type = "gnd"
    else:
        type = "ev"
    np.savetxt('Lz.csv',Lz,delimiter=',')

    plt.plot(Lz)
    plt.savefig("Lz_"+type+".pdf",dpi=imgdpi)
    plt.axis('off')
    plt.savefig("Lz_"+type+"_axis0.pdf",bbox_inches='tight',dpi=imgdpi)
    plt.close()

def expec_val_monopole(dataName, initValue, finalValue, incr):
    x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
    y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
#   px=open('px_0')
#   py=open('py_0')
    xm, ym = np.meshgrid(x, y)
    result = []
    for i in range(initValue,finalValue,incr):
        if not os.path.exists(dataName):
            real=open(dataName + '_' + str(i)).read().splitlines()
            img=open(dataName + 'i_' + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
            a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
            a = a_r[:] + 1j*a_i[:]
            wfc = np.reshape(a,(xDim,yDim))
            conjwfc = np.conj(wfc)

            d1 = np.multiply( np.square(xm) + np.square(ym), wfc )
            d2 = np.multiply( conjwfc, d1)
            result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
        print str(100*float(i)/finalValue) + '%'
    np.savetxt('monopole.csv',result,delimiter=',')
    plt.plot(range(initValue,finalValue,incr),result)
    plt.savefig("Monopole.png",dpi=200)
    plt.close()

def expec_val_quadrupole(dataName, initValue, finalValue, incr):
    x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
    y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
#   px=open('px_0')
#   py=open('py_0')
    xm, ym = np.meshgrid(x, y)
    result = []
    for i in range(initValue,finalValue,incr):
        if not os.path.exists(dataName):
            real=open(dataName + '_' + str(i)).read().splitlines()
            img=open(dataName + 'i_' + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
            a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
            a = a_r[:] + 1j*a_i[:]
            wfc = np.reshape(a,(xDim,yDim))
            conjwfc = np.conj(wfc)

            d1 = np.multiply( np.square(xm) - np.square(ym), wfc )
            d2 = np.multiply( conjwfc, d1)
            result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
        print str(100*float(i)/finalValue) + '%'
    np.savetxt('quadrupole.csv',result,delimiter=',')
    plt.plot(range(initValue,finalValue,incr),result)
    plt.savefig("Quadrupole.png",dpi=200)
    plt.close()

def expec_val_(quant_name, quantity, dataName, initValue, finalValue, incr):
    x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
    y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
#   px=open('px_0')
#   py=open('py_0')
    xm, ym = np.meshgrid(x, y)
    result = []
    for i in range(initValue,finalValue,incr):
        if not os.path.exists(dataName):
            real=open(dataName + '_' + str(i)).read().splitlines()
            img=open(dataName + 'i_' + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
            a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
            a = a_r[:] + 1j*a_i[:]
            wfc = np.reshape(a,(xDim,yDim))
            conjwfc = np.conj(wfc)

            d1 = np.multiply( quantity, wfc )
            d2 = np.multiply( conjwfc, d1)
            result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
        print str(100*float(i)/finalValue) + '%'
    np.savetxt(quant_name + '.csv',result,delimiter=',')
    plt.plot(range(initValue,finalValue,incr),result)
    plt.savefig(quant_name + ".pdf",dpi=200)
    plt.close()

if __name__ == '__main__':
    dens_struct_fact('wfc_ev', 0, evMaxVal, 500)
    exit()
    energy_kinetic('wfc_ev', 0, evMaxVal, 200)
#   ang_mom('wfc_0_ramp', 0, gndMaxVal, incr, 0, 200)
    ang_mom('wfc_ev', 0, evMaxVal, incr, 1, 200)
    expec_val_monopole('wfc_ev',0,evMaxVal,incr)
    expec_val_quadrupole('wfc_ev',0,evMaxVal,incr)
```

4.33 py/overlap.py File Reference

Namespaces

- overlap

Functions

- def overlap.overlap
- def overlap.densitydiff

Variables

- tuple overlap.c = ConfigParser.ConfigParser()
- tuple overlap.xDim = int(c.getfloat('Params','xDim'))
- tuple overlap.yDim = int(c.getfloat('Params','yDim'))
- tuple overlap.gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple overlap.evMaxVal = int(c.getfloat('Params','esteps'))
- tuple overlap.incr = int(c.getfloat('Params','print_out'))
- tuple overlap.sep = (c.getfloat('Params','dx'))
- tuple overlap.dx = (c.getfloat('Params','dx'))
- tuple overlap.dt = (c.getfloat('Params','dt'))
- tuple overlap.xMax = (c.getfloat('Params','xMax'))
- tuple overlap.yMax = (c.getfloat('Params','yMax'))
- tuple overlap.num_vort = int(c.getfloat('Params','Num_vort'))
- tuple overlap.data = numpy.ndarray(shape=(xDim,yDim))
- tuple overlap.real = open("wfc_ev_" + str(0))
- tuple overlap.img = open("wfc_evi_" + str(0))
- tuple overlap.a_r = numpy.asanyarray(real,dtype='f8')
- tuple overlap.a_i = numpy.asanyarray(img,dtype='f8')
- list overlap.wfc0 = a_r[:]
- tuple overlap.rho0 = abs(np.reshape(wfc0,(xDim,yDim)))
- float overlap.norm_coef = 1.0
- list overlap.evImgList = [ ]
- list overlap.ev_proc = [ ]
- tuple overlap.val = evImgList.pop()
- tuple overlap.p = ev_proc.pop()

4.34 overlap.py

```
#
# vis.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O’Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
import os
from numpy import genfromtxt
import math as m
import matplotlib as mpl
import numpy as np
import numpy.matlib
mpl.use(’Agg’)
import multiprocessing as mp
from multiprocessing import Pool
from multiprocessing import Process
from matplotlib.ticker import ScalarFormatter
import matplotlib.pyplot as plt
import ConfigParser
import random as r
from decimal import *

getcontext().prec = 4
c = ConfigParser.ConfigParser()
c.readfp(open(r’Params.dat’))

xDim = int(c.getfloat(’Params’,’xDim’))
yDim = int(c.getfloat(’Params’,’yDim’))
gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
evMaxVal = int(c.getfloat(’Params’,’esteps’))
incr = int(c.getfloat(’Params’,’print_out’))
sep = (c.getfloat(’Params’,’dx’))
dx = (c.getfloat(’Params’,’dx’))
dt = (c.getfloat(’Params’,’dt’))
xMax = (c.getfloat(’Params’,’xMax’))
yMax = (c.getfloat(’Params’,’yMax’))
num_vort = int(c.getfloat(’Params’,’Num_vort’))

data = numpy.ndarray(shape=(xDim,yDim))

print "##Index" + ’\t’ + ’Value’ + ’\t’ +  "Overlap"
def overlap(dataName,value,norm_coef):
    real=open(dataName + ’_’ + str(value)).read().splitlines()
    img=open(dataName + ’i_’ + str(value)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
    a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
    a = a_r[:] + 1j*a_i[:]
    b = np.vdot(wfc0,a)
    s = np.sum(b)
    print str(value) + ’\t’ +  str(s) + ’\t’ +  str(norm_coef*abs(s)**2)

def densitydiff(dataName,value,rho0):
    real=open(dataName + ’_’ + str(value)).read().splitlines()
    img=open(dataName + ’i_’ + str(value)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
    a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
    a = a_r[:] + 1j*a_i[:]
    b = reshape(abs(a)**2,(xDim,yDim))
    c = rho0 - b

    fig, ax = plt.subplots()
    f = plt.imshow(c)
    cbar = fig.colorbar(f)
    #getcontext().prec = 5
    plt.title(’wfc(t=0) - wfc(t=’ + str(value*dt) + ’)’)
    plt.gca().set_xlabel(’x ’+ str((dx)))
    plt.gca().set_ylabel(’y ’+ str(dx))
    plt.gca().invert_yaxis()
    plt.jet()
    plt.savefig(dataName+"r_"+str(value)+"_diff.png",dpi=imgdpi)
    plt.close()

if __name__ == ’__main__’:
    real=open("wfc_ev_" + str(0)).read().splitlines()
    img=open("wfc_evi_" + str(0)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
    a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
    wfc0 = a_r[:] + 1j*a_i[:]
    rho0 = abs(np.reshape(wfc0,(xDim,yDim)))**2
    norm_coef = 1.0/abs(np.sum(np.vdot(wfc0,wfc0)))**2
    print(norm_coef)
    evImgList=[]
    for i in range(0,1000000,500):
        evImgList.append(i)
    ev_proc = []
    while evImgList:
        val=evImgList.pop()
        ev_proc.append(Process(target=densitydiff, args=("wfc_ev",val,rho0))
#       ev_proc.append(Process(target=overlap, args=("wfc_ev",val,norm_coef)))
#       ev_proc.append(Process(target=hist_gen,args=("hist_ev",i,128)))

    while 1:
        if (mp.cpu_count()/8) > len(mp.active_children()):
            try:
                p=ev_proc.pop()
                p.start()
            except:
                print "Failed to execute ", p
```

4.35 py/py_upload.py File Reference

Namespaces

- py_upload

Functions

- def py_upload.get_authenticated_service
- def py_upload.initialize_upload
- def py_upload.resumable_upload

Variables

- int py_upload.MAX_RETRIES = 10
- tuple py_upload.RETRIABLE_EXCEPTIONS
- list py_upload.RETRIABLE_STATUS_CODES = [500, 502, 503, 504]
- string py_upload.CLIENT_SECRETS_FILE = "client_secrets.json"
- string py_upload.YOUTUBE_UPLOAD_SCOPE = "https://www.googleapis.com/auth/youtube.upload"
- string py_upload.YOUTUBE_API_SERVICE_NAME = "youtube"
- string py_upload.YOUTUBE_API_VERSION = "v3"
- string py_upload.MISSING_CLIENT_SECRETS_MESSAGE
- tuple py_upload.parser = OptionParser()
- string py_upload.default = "Test Title"
- string py_upload.help = "Video description"

4.36 py_upload.py

```
#!/usr/bin/python

import httplib
import httplib2
import os
import random
import sys
import time

from apiclient.discovery import build
from apiclient.errors import HttpError
from apiclient.http import MediaFileUpload
from oauth2client.file import Storage
from oauth2client.client import flow_from_clientsecrets
from oauth2client.tools import run
from optparse import OptionParser


# Explicitly tell the underlying HTTP transport library not to retry, since
# we are handling retry logic ourselves.
httplib2.RETRIES = 1

# Maximum number of times to retry before giving up.
MAX_RETRIES = 10

# Always retry when these exceptions are raised.
RETRIABLE_EXCEPTIONS = (httplib2.HttpLib2Error, IOError, httplib.NotConnected,
  httplib.IncompleteRead, httplib.ImproperConnectionState,
  httplib.CannotSendRequest, httplib.CannotSendHeader,
  httplib.ResponseNotReady, httplib.BadStatusLine)

# Always retry when an apiclient.errors.HttpError with one of these status
# codes is raised.
RETRIABLE_STATUS_CODES = [500, 502, 503, 504]

# CLIENT_SECRETS_FILE, name of a file containing the OAuth 2.0 information for
# this application, including client_id and client_secret. You can acquire an
# ID/secret pair from the API Access tab on the Google APIs Console
#   http://code.google.com/apis/console#access
# For more information about using OAuth2 to access Google APIs, please visit:
#   https://developers.google.com/accounts/docs/OAuth2
# For more information about the client_secrets.json file format, please visit:
#   https://developers.google.com/api-client-library/python/guide/aaa_client_secrets
# Please ensure that you have enabled the YouTube Data API for your project.
CLIENT_SECRETS_FILE = "client_secrets.json"

# A limited OAuth 2 access scope that allows for uploading files, but not other
# types of account access.
YOUTUBE_UPLOAD_SCOPE = "https://www.googleapis.com/auth/youtube.upload"
YOUTUBE_API_SERVICE_NAME = "youtube"
YOUTUBE_API_VERSION = "v3"

# Helpful message to display if the CLIENT_SECRETS_FILE is missing.
MISSING_CLIENT_SECRETS_MESSAGE = """
WARNING: Please configure OAuth 2.0

To make this sample run you will need to populate the client_secrets.json file
found at:

   %s

with information from the APIs Console
https://code.google.com/apis/console#access

For more information about the client_secrets.json file format, please visit:
https://developers.google.com/api-client-library/python/guide/aaa_client_secrets
""" % os.path.abspath(os.path.join(os.path.dirname(__file__),
                                   CLIENT_SECRETS_FILE))

def get_authenticated_service():
  flow = flow_from_clientsecrets(CLIENT_SECRETS_FILE, scope=YOUTUBE_UPLOAD_SCOPE,
    message=MISSING_CLIENT_SECRETS_MESSAGE)

  storage = Storage("%s-oauth2.json" % sys.argv[0])
  credentials = storage.get()

  if credentials is None or credentials.invalid:
    credentials = run(flow, storage)

  return build(YOUTUBE_API_SERVICE_NAME, YOUTUBE_API_VERSION,
    http=credentials.authorize(httplib2.Http()))


def initialize_upload(options):
  youtube = get_authenticated_service()

  tags = None
  if options.keywords:
    tags = options.keywords.split(",")

  insert_request = youtube.videos().insert(
    part="snippet,status",
    body=dict(
      snippet=dict(
        title=options.title,
        description=options.description,
        tags=tags,
        categoryId=options.category
      ),
      status=dict(
        privacyStatus=options.privacyStatus
      )
    ),
    # chunksize=-1 means that the entire file will be uploaded in a single
    # HTTP request. (If the upload fails, it will still be retried where it
    # left off.) This is usually a best practice, but if you’re using Python
    # older than 2.6 or if you’re running on App Engine, you should set the
    # chunksize to something like 1024 * 1024 (1 megabyte).
    media_body=MediaFileUpload(options.file, chunksize=-1, resumable=True)
  )

  resumable_upload(insert_request)


def resumable_upload(insert_request):
  response = None
  error = None
  retry = 0
  while response is None:
    try:
      print "Uploading file..."
      status, response = insert_request.next_chunk()
      if ’id’ in response:
        print "’%s’ (video id: %s) was successfully uploaded." % (
          options.title, response[’id’])
      else:
        exit("The upload failed with an unexpected response: %s" % response)
    except HttpError, e:
      if e.resp.status in RETRIABLE_STATUS_CODES:
        error = "A retriable HTTP error %d occurred:\n%s" % (e.resp.status,
                                                             e.content)
      else:
        raise
    except RETRIABLE_EXCEPTIONS, e:
      error = "A retriable error occurred: %s" % e

    if error is not None:
      print error
      retry += 1
      if retry > MAX_RETRIES:
        exit("No longer attempting to retry.")

      max_sleep = 2 ** retry
      sleep_seconds = random.random() * max_sleep
      print "Sleeping %f seconds and then retrying..." % sleep_seconds
      time.sleep(sleep_seconds)


if __name__ == ’__main__’:
  parser = OptionParser()
  parser.add_option("--file", dest="file", help="Video file to upload")
  parser.add_option("--title", dest="title", help="Video title",
    default="Test Title")
  parser.add_option("--description", dest="description",
    help="Video description",
    default="Test Description")
  parser.add_option("--category", dest="category",
    help="Numeric video category. " +
      "See https://developers.google.com/youtube/v3/docs/videoCategories/list",
    default="22")
  parser.add_option("--keywords", dest="keywords",
    help="Video keywords, comma separated", default="")
  parser.add_option("--privacyStatus", dest="privacyStatus",
    help="Video privacy status: public, private or unlisted",
    default="public")
  (options, args) = parser.parse_args()

  if options.file is None or not os.path.exists(options.file):
    exit("Please specify a valid file using the --file= parameter.")
  else:
    initialize_upload(options)
```

4.37 py/run.py File Reference

Namespaces

- run

4.38 run.py

4.39 py/stats.py File Reference

Namespaces

- stats

Functions

- def stats.lsFit

Variables

- tuple stats.c = ConfigParser.ConfigParser()
- tuple stats.incr = int(c.getfloat('Params','print_out'))
- tuple stats.xDim = int(c.getfloat('Params','xDim'))
- tuple stats.yDim = int(c.getfloat('Params','yDim'))

4.40 stats.py

```
#
# stats.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
import os
from numpy import genfromtxt
import math as m
#import matplotlib as mpl
import numpy as np
import numpy.matlib
#mpl.use('Agg')
#import multiprocessing as mp
#from multiprocessing import Pool
#from multiprocessing import Process
#from matplotlib.ticker import ScalarFormatter
#import matplotlib.pyplot as plt
import ConfigParser
import random as r
from decimal import *

#getcontext().prec = 4
c = ConfigParser.ConfigParser()
c.readfp(open(r'Params.dat'))

incr = int(c.getfloat('Params','print_out'))
xDim = int(c.getfloat('Params','xDim'))
yDim = int(c.getfloat('Params','yDim'))

def lsFit(start,end,incr):
    L = np.matrix([
            [0,0,1],
            [1,0,1],
            [0,1,1],
            [1,1,1]
            ])
    LSQ = np.linalg.inv(np.transpose(L)*L)*np.transpose(L)
    for i in range(start,end,incr):
        v_arr=genfromtxt('vort_arr_' + str(i),delimiter=',' )
        real=open('wfc_ev_' + str(i)).read().splitlines()
        img=open('wfc_evi_' + str(i)).read().splitlines()
        a_r = np.asanyarray(real,dtype='f8') #64-bit double
        a_i = np.asanyarray(img,dtype='f8') #64-bit double
        a = a_r[:] + 1j*a_i[:]
        wfc = (np.reshape(a,(xDim,yDim)))

        indX = [row[0] for row in v_arr]
        indY = [row[1] for row in v_arr]
        wind = [row[2] for row in v_arr]
        sign = [row[3] for row in v_arr]
        data=[]
        for ii in range(0,len(indX)):
            p=np.matrix([[0],[0],[0],[0]],dtype=np.complex)
            p[0]=(wfc[indX[ii], indY[ii]])
            p[1]=(wfc[indX[ii]+1, indY[ii]])
            p[2]=(wfc[indX[ii], indY[ii]+1])
            p[3]=(wfc[indX[ii]+1, indY[ii]+1])
            rc = LSQ * np.real(p)
            ic = LSQ * np.imag(p)

            A=np.squeeze([row[0:2] for row in [rc,ic]])
            B=-np.squeeze([row[2] for row in [rc,ic]])
            r=np.linalg.lstsq(A,B)[0]
            data.append([indX[ii]+r[0],indY[ii]+r[1],sign[ii]])

#       f = plt.imshow(abs(wfc)**2)
#       plt.jet()
#       plt.gca().invert_yaxis()
#       plt.hold(True)
#       X = [row[0] for row in data]
#       Y = [row[1] for row in data]
#       plt.scatter(Y,X,s=0.2,marker='.',c='red',lw=0)
#       plt.scatter(indY,indX,s=0.2,marker='.',c='yellow',lw=0)
#       plt.savefig("fig.png",dpi=1200)
#       plt.close()
        np.savetxt('vort_lsq_'+str(i)+'.csv',data,delimiter=',')
```

4.41 py/track.py File Reference

Namespaces

- track

Variables

- tuple track.img = cv.LoadImage("foo2.jpg",cv.CV_LOAD_IMAGE_GRAYSCALE)
- tuple track.eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
- tuple track.temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

4.42 track.py

```
import cv
img= cv.LoadImage("foo2.jpg",cv.CV_LOAD_IMAGE_GRAYSCALE)
eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
for (x,y) in cv.GoodFeaturesToTrack(img, eig_image, temp_image, 300, 0.1, 1.0, None, 3, True):
  print "good feature at", x,y
```

4.43 py/track_circles.py File Reference

Namespaces

- track_circles

Variables

- tuple track_circles.img = cv.LoadImage("wfc_1000.png",cv.CV_LOAD_IMAGE_GRAYSCALE)
- tuple track_circles.eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
- tuple track_circles.temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
- tuple track_circles.circles = cv.CreateMat(img.width,1,cv.CV_32FC3)
- tuple track_circles.c = numpy.asarray(circles)

4.44 track_circles.py

```
import cv, numpy
img= cv.LoadImage("wfc_1000.png",cv.CV_LOAD_IMAGE_GRAYSCALE)
eig_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)
temp_image = cv.CreateImage(cv.GetSize(img), cv.IPL_DEPTH_32F, 1)

circles=cv.CreateMat(img.width,1,cv.CV_32FC3)
cv.HoughCircles(img,circles,cv.CV_HOUGH_GRADIENT,2,10, 200,100)
c=numpy.asarray(circles)
for (x) in c:
    print x
```

4.45 py/vis.py File Reference

Namespaces

- vis

Functions

- def vis.delaunay
- def vis.voronoi
- def vis.laplacian
- def vis.struct_fact
- def vis.opPot
- def vis.hist_gen
- def vis.image_gen
- def vis.image_gen_single
- def vis.vort_traj
- def vis.scaleAxis
- def vis.overlap

Variables

- tuple vis.c = ConfigParser.ConfigParser()
- tuple vis.xDim = int(c.getfloat('Params','xDim'))
- tuple vis.yDim = int(c.getfloat('Params','yDim'))
- tuple vis.gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple vis.evMaxVal = int(c.getfloat('Params','esteps'))
- tuple vis.incr = int(c.getfloat('Params','print_out'))
- tuple vis.sep = (c.getfloat('Params','dx'))

- tuple vis.dx = (c.getfloat(’Params’,’dx’))
- tuple vis.dt = (c.getfloat(’Params’,’dt’))
- tuple vis.xMax = (c.getfloat(’Params’,’xMax’))
- tuple vis.yMax = (c.getfloat(’Params’,’yMax’))
- tuple vis.num_vort = int(c.getfloat(’Params’,’Num_vort’))
- tuple vis.data = numpy.ndarray(shape=(xDim,yDim))
- list vis.gndImgList = [ ]
- list vis.evImgList = [ ]
- list vis.gnd_proc = [ ]
- list vis.ev_proc = [ ]
- tuple vis.i = gndImgList.pop()
- vis.proc = gnd_proc+ev_proc
- tuple vis.p = proc.pop()

4.46 vis.py

```
#
# vis.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O’Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
import os
from numpy import genfromtxt
import math as m
import matplotlib as mpl
import matplotlib.tri as tri
import numpy as np
import scipy as sp
from scipy.spatial import Voronoi, voronoi_plot_2d
import numpy.matlib
mpl.use(’Agg’)
import multiprocessing as mp
from multiprocessing import Pool
from multiprocessing import Process
from matplotlib.ticker import ScalarFormatter
import matplotlib.pyplot as plt
import ConfigParser
import random as r
from decimal import *
import stats
import hist3d

getcontext().prec = 4
c = ConfigParser.ConfigParser()
c.readfp(open(r’Params.dat’))

xDim = int(c.getfloat(’Params’,’xDim’))
yDim = int(c.getfloat(’Params’,’yDim’))
gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
evMaxVal = int(c.getfloat(’Params’,’esteps’))
incr = int(c.getfloat(’Params’,’print_out’))
sep = (c.getfloat(’Params’,’dx’))
dx = (c.getfloat(’Params’,’dx’))
dt = (c.getfloat(’Params’,’dt’))
xMax = (c.getfloat(’Params’,’xMax’))
yMax = (c.getfloat(’Params’,’yMax’))
num_vort = int(c.getfloat(’Params’,’Num_vort’))

data = numpy.ndarray(shape=(xDim,yDim))

def delaunay(dataName,dataType,value):
    v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=’,’ )
    data = np.array([[row[0],row[1]] for row in v_arr])
    dln = sp.spatial.Delaunay(data)
    plt.triplot(data[:,0],data[:,1],dln.simplices.copy(),linewidth=0.5,color=’b’,marker=’.’)
    plt.xlim(300,700);plt.ylim(300,700);
    plt.savefig(’delaun_’ + str(value) + ’.png’,dpi=200)
    print ’Saved Delaunay @ t=’ + str(value)

def voronoi(dataName,dataType,value):
    v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=’,’ )
    data = [[row[0],row[1]] for row in v_arr]
    vor = Voronoi(data)
    voronoi_plot_2d(vor)
    plt.xlim(300,700);plt.ylim(300,700);
    plt.savefig(’voronoi_’ + str(value) + ’.png’,dpi=200)
    print ’Saved Voronoi @ t=’ + str(value)

def laplacian(density,name,imgdpi):
    gx,gy = np.gradient(density)
    g2x,gxgy = np.gradient(gx)
    gygx,g2y = np.gradient(gy)
    fig, ax = plt.subplots()
    #f = plt.quiver(gx,gy)
    f = plt.imshow((g2x**2 + g2y**2),cmap=plt.get_cmap(’spectral’))
    cbar = fig.colorbar(f)
    plt.savefig(name + "_laplacian.png",dpi=imgdpi)
    plt.close()
    f = plt.imshow((gxgy - gygx),cmap=plt.get_cmap(’spectral’))
    cbar = fig.colorbar(f)
    plt.savefig(name + "_dxdy.png",dpi=imgdpi)
    plt.close()

def struct_fact(density,name,imgdpi):
    fig, ax = plt.subplots()
    #f = plt.quiver(gx,gy)
    f = plt.imshow((np.abs(np.fft.fftshift(np.fft.fft2(density)))),cmap=plt.get_cmap(’prism’))
    cbar = fig.colorbar(f)
    cbar.set_clim(1e6,1e11)
    plt.jet()
    plt.savefig(name + "_struct_log10.png",dpi=imgdpi)
    plt.close()

def opPot(dataName,imgdpi):
    data = open(dataName).read().splitlines()
    a = numpy.asanyarray(data,dtype=’f8’)
    b = np.reshape(a,(xDim,yDim))
    fig, ax = plt.subplots()
    f = plt.imshow((b))
    plt.gca().invert_yaxis()
    cbar = fig.colorbar(f)
    plt.jet()
    plt.savefig(dataName + ".png",dpi=imgdpi)
    plt.close()

def hist_gen(name,value,num_bins):
    v_arr=genfromtxt(’vort_arr_’ + str(value),delimiter=’,’ )
    H=[]
    count=0

    for i1 in range(0,v_arr.size/2):
        for i2 in range(i1,v_arr.size/2):
            H.append(m.sqrt( abs(v_arr[i1][0]*sep - v_arr[i2][0]*sep)**2  +  abs(v_arr[i1][1]*sep - v_arr[i2][1]*sep)**2 ))
            count = count + 1
    plt.title(’Vortex lattice @ t=’ + str(value*dt))
    plt.ticklabel_format(style=’scientific’)
    plt.ticklabel_format(style=’scientific’,axis=’x’, scilimits=(0,0))
    h = plt.hist(H, bins=num_bins)
    plt.savefig(name + "_" + str(value) + ".pdf")
    plt.close()

def image_gen(dataName, initValue, finalValue, increment,imgdpi):
    for i in range(initValue,finalValue,increment):
        if not os.path.exists(dataName+"r_"+str(i)+"_abspsi2.png"):
            real=open(dataName + ’_’ + str(i)).read().splitlines()
            img=open(dataName + ’i_’ + str(i)).read().splitlines()
            a_r = numpy.asanyarray(real,dtype=’f8’) #64-bit double
            a_i = numpy.asanyarray(img,dtype=’f8’) #64-bit double
            a = a_r[:] + 1j*a_i[:]
            b = np.reshape(a,(xDim,yDim))
            f = plt.imshow(abs(b)**2)
            plt.jet()
            plt.gca().invert_yaxis()
            plt.savefig(dataName+"r_"+str(i)+"_abspsi2.png",dpi=imgdpi)
            plt.close()
            g = plt.imshow(np.angle(b))
            plt.gca().invert_yaxis()
            plt.savefig(dataName+"r_"+str(i)+"_phi.png",dpi=imgdpi)
            plt.close()
            f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))**2)
            plt.gca().invert_yaxis()
            plt.jet()
            plt.savefig(dataName+"p_"+str(i)+"_abspsi2.png",dpi=imgdpi)
            plt.close()
            g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
            plt.gca().invert_yaxis()
            plt.savefig(dataName+"p_"+str(i)+"_phi.png",dpi=imgdpi)
            plt.close()
            print "Saved figure: " + str(i) + ".png"
            plt.close()
        else:
            print "File(s) " + str(i) +".png already exist."

def image_gen_single(dataName, value, imgdpi,opmode):
    real=open(dataName + '_' + str(0)).read().splitlines()
    img=open(dataName + 'i_' + str(0)).read().splitlines()
    a1_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
    a1_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
    a1 = a1_r[:] + 1j*a1_i[:]
    b1 = np.reshape(a1,(xDim,yDim))

    if not os.path.exists(dataName+"r_"+str(value)+"_abspsi2.png"):
        real=open(dataName + '_' + str(value)).read().splitlines()
        img=open(dataName + 'i_' + str(value)).read().splitlines()
        a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
        a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
        a = a_r[:] + 1j*a_i[:]
        b = np.reshape(a,(xDim,yDim))

        #scaleAxis(b,dataName,"_abspsi2",value,imgdpi)
        if opmode & 0b100000 > 0:
            fig, ax = plt.subplots()
            #plt.rc('text',usetex=True)
            #plt.rc('font',family='serif')
            f = plt.imshow((abs(b)**2 - abs(b1)**2),cmap='gnuplot2',vmin=-6,vmax=6)
            plt.title(r'$\left(\rho( r,t ) - \rho( r,t_0 )\right),t=$' + str(value*dt))
            cbar = fig.colorbar(f)
            plt.gca().set_xlabel('x '+ str((dx)))
            plt.gca().set_ylabel('x '+ str(dx))
            plt.gca().invert_yaxis()
            plt.savefig(dataName+"r_"+str(value)+"_diffabspsi2.png",dpi=imgdpi)
            plt.close()
            #plt.rc('text',usetex=True)
            #plt.rc('font',family='serif')
            fig, ax = plt.subplots()
            f = plt.imshow((abs(b)**2),cmap='gnuplot2',vmin=0,vmax=8)
            plt.title('rho(r) @ t=' + str(value*dt))
            plt.title(r'$\log_{10}\rho \left( r,t \right),\,t=$' + str(value*dt))

            cbar = fig.colorbar(f)
            plt.gca().set_xlabel('x '+ str((dx)))
            plt.gca().set_ylabel('x '+ str(dx))
            plt.gca().invert_yaxis()
            plt.savefig(dataName+"r_"+str(value)+"_abspsi2.png",dpi=imgdpi)
            plt.axis('off')
            plt.savefig(dataName+"r_"+str(value)+"_abspsi2_axis0.pdf",bbox_inches='tight',dpi=imgdpi)
            plt.close()

        if opmode & 0b010000 > 0:
            fig, ax = plt.subplots()
            g = plt.imshow(np.angle(b))
            cbar = fig.colorbar(g)
            plt.gca().invert_yaxis()
            plt.title('theta(r) @ t=' + str(value*dt))
            plt.savefig(dataName+"r_"+str(value)+"_phi.png",dpi=imgdpi)
            plt.close()

        if opmode & 0b001000 > 0:
            fig, ax = plt.subplots()
            f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))**2)
            cbar = fig.colorbar(f)
            plt.gca().invert_yaxis()
            plt.jet()
            plt.title('rho(p) @ t=' + str(value*dt))
            plt.savefig(dataName+"p_"+str(value)+"_abspsi2.png",dpi=imgdpi)
            plt.close()

        if opmode & 0b000100 > 0:
            fig, ax = plt.subplots()
            g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
            cbar = fig.colorbar(g)
            plt.gca().invert_yaxis()
            plt.title('theta(p) @ t=' + str(value*dt))
            plt.savefig(dataName+"p_"+str(value)+"_phi.png",dpi=imgdpi)
            plt.close()

        if opmode & 0b000010 > 0:
            struct_fact(abs(b)**2,dataName+"_" + str(value),imgdpi)

        if opmode & 0b000001 > 0:
            laplacian(abs(b)**2,dataName+"_" + str(value),imgdpi)

        print "Saved figure: " + str(value) + ".png"
        plt.close()
    else:
        print "File(s) " + str(value) +".png already exist."

def vort_traj(name,imgdpi):
    evMaxVal_l = evMaxVal
    H=genfromtxt('vort_arr_0',delimiter=',' )
    count=0
    for i1 in range(incr,evMaxVal_l,incr):
        try:
            v_arr=genfromtxt('vort_lsq_' + str(i1) + '.csv',delimiter=',' )
            H=np.column_stack((H,v_arr))
        except:
            evMaxVal_l = i1
            break
    X=np.zeros((evMaxVal_l/incr),dtype=np.float64)
    Y=np.zeros((evMaxVal_l/incr),dtype=np.float64)
    H=np.reshape(H,([num_vort,2,evMaxVal_l/incr]),order='F')
    for i1 in range(0, num_vort):
        for i2 in range(0,evMaxVal_l/incr):
            X[i2]=(H[i1,0,i2]*dx) - xMax
            Y[i2]=(H[i1,1,i2]*dx) - yMax
        h = plt.plot(X,Y,color=(r.random(),r.random(),r.random(),0.85),linewidth=0.1)
    plt.axis('equal')
    plt.title('Vort(x,y) from t=0 to t='+str(evMaxVal_l*dt)+" s")

    plt.axis((-xMax/2.0, xMax/2.0, -yMax/2.0, yMax/2.0))
    plt.ticklabel_format(style='scientific')
    plt.ticklabel_format(style='scientific',axis='x', scilimits=(0,0))
    plt.ticklabel_format(style='scientific',axis='y', scilimits=(0,0))
    plt.savefig(name +".pdf")
    plt.close()
    print "Trajectories plotted."

def scaleAxis(data,dataName,label,value,imgdpi):
    fig, ax = plt.subplots()
    ax.xaxis.set_major_locator(ScaledLocator(dx=dx))
    ax.xaxis.set_major_formatter(ScaledLocator(dx=dx))
    f = plt.imshow(abs(data)**2)
    cbar = fig.colorbar(f)
    plt.gca().invert_yaxis()
    plt.jet()
    plt.savefig(dataName+"r_"+str(value)+"_"+label +".png",dpi=imgdpi)
    plt.close()

def overlap(dataName, initValue, finalValue, increment):
    real=open(dataName + '_' + str(0)).read().splitlines()
    img=open(dataName + 'i_' + str(0)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
    a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
    wfc0 = a_r[:] + 1j*a_i[:]
    for i in range(initValue,finalValue,increment):
        real=open(dataName + '_' + str(value)).read().splitlines()
        img=open(dataName + 'i_' + str(value)).read().splitlines()
        a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
        a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
        a = a_r[:] + 1j*a_i[:]
        b = np.dot(wfc0,a)
        print i, np.sum(b)

if __name__ == '__main__':
#   delaunay('vort_arr_',0)
#   try:
#       #stats.lsFit(0,evMaxVal,incr)
#       hist3d.plot_hist_pcolor(0,evMaxVal,incr,'b')
#   except:
#       print "Unhandled error occurred. Blame Lee."
    #vort_traj('traj_plot',200)
    opPot('V_opt_0',200)
    opPot('V_0',200)
    opPot('K_0',200)
    gndImgList=[]
    evImgList=[]
    for i in range(0,gndMaxVal,incr):
        gndImgList.append(i)
    for i in range(0,evMaxVal,incr):
        evImgList.append(i)
    gnd_proc = []
    ev_proc = []
    while gndImgList:
        i=gndImgList.pop()
        gnd_proc.append(Process(target=image_gen_single,args=("wfc_0_ramp",i,200,0b100000)))
        gnd_proc.append(Process(target=image_gen_single,args=("wfc_0_const",i,200,0b100000)))
    while evImgList:
        i=evImgList.pop()
        ev_proc.append(Process(target=image_gen_single,args=("wfc_ev",i,200,0b100000)))
        ev_proc.append(Process(target=delaunay,args=("vort_lsq_",'.csv',i)))
        ev_proc.append(Process(target=voronoi,args=("vort_lsq_",'.csv',i)))
#       ev_proc.append(Process(target=hist_gen,args=("hist_ev",i,128)))
    proc = gnd_proc + ev_proc
    while proc:
        if (mp.cpu_count()/2) > len(mp.active_children()):
            try:
                p=proc.pop()
                p.start()
            except:
                print "Failed to execute ", p
```

4.47 py/vis_ev.py File Reference

Namespaces

- vis_ev

Variables

- int vis_ev.xDim = 256
- int vis_ev.yDim = 256
- tuple vis_ev.data = numpy.ndarray(shape=(xDim,yDim))
- string vis_ev.s = "./wfc"
- tuple vis_ev.real = open(s + ’_’ + str(i))
- tuple vis_ev.img = open(s + ’i_’ + str(i))
- tuple vis_ev.a_r = numpy.asanyarray(real,dtype=’f8’)
- tuple vis_ev.a_i = numpy.asanyarray(img,dtype=’f8’)
- list vis_ev.a = a_r[:]
- tuple vis_ev.b = np.reshape(a,(xDim,yDim))
- tuple vis_ev.f = plt.imshow(abs(b)**2)

4.48 vis_ev.py

```
#
# vis_ev.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O’Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
import scipy
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import *
import numpy.matlib
xDim=256
yDim=256
data = numpy.ndarray(shape=(xDim,yDim))
s = "./wfc"
#figure(size=(xDim,yDim))
for i in range(0,50000,1000):
    real=open(s + '_' + str(i)).read().splitlines()
    img=open(s + ’i_’ + str(i)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
    a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
    a = a_r[:] + 1j*a_i[:]
    b = np.reshape(a,(xDim,yDim))
    f = plt.imshow(abs(b)**2)
    plt.jet()
#   plt.show()
    #view(0,0)
    plt.savefig("wfc_ev_"+str(i)+".png")#,size=(800,600))
#   close(gcf())
    print "Saved figure: " + str(i) + ".png"
del a, a_r, a_i
#contour3d(b, contours=4, transparent=True)
#imshow(abs(b)**2)
#data_tpot = scipy.io.loadmat(’/home/mlxd/workspace/Dev/Tpot.mat’)
#oct_a = data_tpot[’Pot’]
#contour3d(oct_a, contours=4, transparent=True)
#data_wfc = scipy.io.loadmat(’/home/mlxd/workspace/Dev/WFC_0.mat’)
#oct_b = data_wfc[’wfabs’]
#contour3d(oct_b, contours=4, transparent=True)
```

4.49 py/visual_ev.py File Reference

Namespaces

- visual_ev

Variables

- int visual_ev.xDim = 256
- int visual_ev.yDim = 256
- tuple visual_ev.data = numpy.ndarray(shape=(xDim,yDim))
- string visual_ev.s = "./wfc"
- tuple visual_ev.real = open(s + ’_’ + str(i))
- tuple visual_ev.img = open(s + ’i_’ + str(i))
- tuple visual_ev.a_r = numpy.asanyarray(real,dtype=’f8’)
- tuple visual_ev.a_i = numpy.asanyarray(img,dtype=’f8’)
- list visual_ev.a = a_r[:]
- tuple visual_ev.b = numpy.reshape(a,(xDim,yDim))

4.50 visual_ev.py

```
#
# visual_ev.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O’Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#

import numpy, scipy, mayavi
from mayavi.mlab import *
from scipy.io import *
import numpy.matlib
xDim=256
yDim=256
data = numpy.ndarray(shape=(xDim,yDim))
s = "./wfc"
```

```
figure(size=(xDim,yDim))
for i in range(0,50000,500):
    real=open(s + ’_’ + str(i)).read().splitlines()
    img=open(s + ’i_’ + str(i)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype=’f8’) #128-bit complex
    a_i = numpy.asanyarray(img,dtype=’f8’) #128-bit complex
    a = a_r[:] + 1j*a_i[:]
    b = numpy.reshape(a,(xDim,yDim))
    imshow(abs(b)**2)
    view(0,0)
    savefig("wfc_"+str(i)+".png")#,size=(800,600))
#    close(gcf())
del a, a_r, a_i
#contour3d(b, contours=4, transparent=True)
#imshow(abs(b)**2)
#data_tpot = scipy.io.loadmat(’/home/mlxd/workspace/Dev/Tpot.mat’)
#oct_a = data_tpot[’Pot’]
#contour3d(oct_a, contours=4, transparent=True)
#data_wfc = scipy.io.loadmat(’/home/mlxd/workspace/Dev/WFC_0.mat’)
#oct_b = data_wfc[’wfabs’]
#contour3d(oct_b, contours=4, transparent=True)
```

4.51 py/visual_gnd.py File Reference

Namespaces

- visual_gnd

Variables

- int visual_gnd.xDim = 256
- int visual_gnd.yDim = 256
- tuple visual_gnd.data = numpy.ndarray(shape=(xDim,yDim))
- string visual_gnd.s = "./wfc_0"
- tuple visual_gnd.real = open(s + ’_’ + str(i))
- tuple visual_gnd.img = open(s + ’i_’ + str(i))
- tuple visual_gnd.a_r = numpy.asanyarray(real,dtype=’f8’)
- tuple visual_gnd.a_i = numpy.asanyarray(img,dtype=’f8’)
- list visual_gnd.a = a_r[:]
- tuple visual_gnd.b = numpy.reshape(a,(xDim,yDim))

4.52 visual_gnd.py

```
#
# visual_gnd.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2012, Lee J. O’Riordan, Tadhg
# Morgan, Neil Crowley.

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
import numpy, scipy, mayavi
from mayavi.mlab import *
from scipy.io import *
import numpy.matlib
xDim=256
yDim=256
data = numpy.ndarray(shape=(xDim,yDim))
s = "./wfc_0"
figure(size=(xDim,yDim))
```

```
for i in range(0,50000,500):
    real=open(s + '_' + str(i)).read().splitlines()
    img=open(s + 'i_' + str(i)).read().splitlines()
    a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
    a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
    a = a_r[:] + 1j*a_i[:]
    b = numpy.reshape(a,(xDim,yDim))
    imshow(abs(b)**2)
    view(0,0)
    savefig("wfc_"+str(i)+".png")#,size=(800,600))
#    close(gcf())
del a, a_r, a_i
#contour3d(b, contours=4, transparent=True)
#imshow(abs(b)**2)
#data_tpot = scipy.io.loadmat('/home/mlxd/workspace/Dev/Tpot.mat')
#oct_a = data_tpot['Pot']
#contour3d(oct_a, contours=4, transparent=True)
#data_wfc = scipy.io.loadmat('/home/mlxd/workspace/Dev/WFC_0.mat')
#oct_b = data_wfc['wfabs']
#contour3d(oct_b, contours=4, transparent=True)
```

4.53 py/vort.py File Reference

Classes

- class vort.Vortex
- class vort.VtxList

Namespaces

- vort

Functions

- def vort.__init__
- def vort.update_uid
- def vort.update_on
- def vort.update_next
- def vort.dist
- def vort.__init__
- def vort.element
- def vort.vtx_uid
- def vort.max_uid
- def vort.add
- def vort.as_np
- def vort.write_out
- def vort.idx_min_dist
- def vort.remove
- def vort.swap_uid
- def vort.vort_decrease
- def vort.vort_increase
- def vort.do_the_thing

Variables

- tuple vort.c = ConfigParser.ConfigParser()
- tuple vort.xDim = int(c.getfloat('Params','xDim'))
- tuple vort.yDim = int(c.getfloat('Params','yDim'))
- tuple vort.gndMaxVal = int(c.getfloat('Params','gsteps'))

- tuple vort.evMaxVal = int(c.getfloat('Params','esteps'))
- tuple vort.incr = int(c.getfloat('Params','print_out'))
- tuple vort.dx = (c.getfloat('Params','dx'))
- tuple vort.dt = (c.getfloat('Params','dt'))
- tuple vort.xMax = (c.getfloat('Params','xMax'))
- tuple vort.yMax = (c.getfloat('Params','yMax'))
- tuple vort.r = m.sqrt((self.x - vtx.x)∗∗2 + (self.y - vtx.y)∗∗2)
- int vort.pos_l = 0
- vort.vtx = self.head
- int vort.pos = 0
- int vort.val = 0
- list vort.dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]
- list vort.data = [ ]
- int vort.i = 0
- int vort.counter = 0
- vort.ret_idx = counter
- tuple vort.current = self.element(pos-1)
- tuple vort.vtx_pos = self.vtx_uid(uid_i)
- tuple vort.max_uid = vorts_p.max_uid()
- tuple vort.v_arr_p = genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')

  v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')
- tuple vort.vorts_p = VtxList()
- tuple vort.vorts_c = VtxList()
- tuple vort.v_arr_c = genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
- tuple vort.v_arr_p_coords = np.array([[a for a in v][:2] for v in v_arr_p])
- tuple vort.v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])
- tuple vort.v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])
- tuple vort.v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])
- tuple vort.vtx_p = Vortex(i1,v_arr_p_coords[i1][0],v_arr_p_coords[i1][1],True,sign=v_arr_p_sign[i1])
- tuple vort.vtx_c = Vortex(-1-i2,v_arr_c_coords[i2][0],v_arr_c_coords[i2][1],True,sign=v_arr_c_sign[i2])
- tuple vort.index_r = vorts_c.idx_min_dist(vorts_p.element(i3))
- tuple vort.v0c = vorts_c.element(index_r[0])
- tuple vort.v0p = vorts_p.element(i3)
- tuple vort.v1c = vorts_c.element(index_r[0])
- list vort.uid_c = [[a for a in b][3] for b in vorts_c.as_np()]
- list vort.uid_p = [[a for a in b][3] for b in vorts_p.as_np()]
- tuple vort.dpc = set(uid_p)
- tuple vort.dcp = set(uid_c)
- list vort.vtx_pos_p = [ ]
- list vort.vtx_pos_c = [ ]
- tuple vort.vorts_c_update = sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])

4.53.1 Class Documentation

4.53.1.1 class vort::Vortex

Definition at line 41 of file vort.py.

Collaboration diagram for vort.Vortex:

4.53.1.2 class vort::VtxList

Definition at line 75 of file vort.py.

Collaboration diagram for vort.VtxList:

4.54 vort.py

```
###############################################################################
#
# vort.py - GPUE: Split Operator based GPU solver for Nonlinear
# Schrodinger Equation, Copyright (C) 2014, Lee J. O’Riordan

# This library is free software; you can redistribute it and/or modify
# it under the terms of the GNU Lesser General Public License as
# published by the Free Software Foundation; either version 2.1 of the
# License, or (at your option) any later version. This library is
# distributed in the hope that it will be useful, but WITHOUT ANY
# WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
# License for more details. You should have received a copy of the GNU
# Lesser General Public License along with this library; if not, write
# to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
# Boston, MA 02111-1307 USA
#
###############################################################################
import os
from numpy import genfromtxt
import math as m
import numpy as np
import copy as cp
import ConfigParser

###############################################################################
c = ConfigParser.ConfigParser()
c.readfp(open(r’Params.dat’))

xDim = int(c.getfloat(’Params’,’xDim’))
yDim = int(c.getfloat(’Params’,’yDim’))
gndMaxVal = int(c.getfloat(’Params’,’gsteps’))
evMaxVal = int(c.getfloat(’Params’,’esteps’))
incr = int(c.getfloat(’Params’,’print_out’))
dx = (c.getfloat(’Params’,’dx’))
dt = (c.getfloat(’Params’,’dt’))
xMax = (c.getfloat(’Params’,’xMax’))
yMax = (c.getfloat(’Params’,’yMax’))

###############################################################################
class Vortex:
###############################################################################
###############################################################################
    def __init__(self,uid,x,y,isOn,sign=1):
###############################################################################
        self.uid = uid
        self.x = x
        self.y = y
        self.sign = sign
        self.isOn = isOn
        self.next = None

###############################################################################
    def update_uid(self,uid):
###############################################################################
        self.uid = uid

###############################################################################
    def update_on(self,isOn):
###############################################################################
        self.isOn = isOn

###############################################################################
    def update_next(self,next):
###############################################################################
        self.next = next

###############################################################################
    def dist(self,vtx):
###############################################################################
        r = m.sqrt((self.x - vtx.x)**2 + (self.y - vtx.y)**2)
        return r

###############################################################################
class VtxList:
###############################################################################
###############################################################################
    def __init__(self):
###############################################################################
        self.head = None
        self.tail = None
        self.length = 0

###############################################################################
```

```
    def element(self,pos):
################################################################################
        pos_l = 0
        if pos < self.length:
            vtx = self.head
            while pos_l < pos:
                pos_l = pos_l +1
                vtx = vtx.next
        else:
            print "Out of bounds"
            exit(-1)
        return vtx

################################################################################
    def vtx_uid(self,uid):
################################################################################
        vtx = self.head
        pos = 0
        while vtx.uid != uid:
            vtx = vtx.next
            pos = pos +1
        return [vtx,pos]

################################################################################
    def max_uid(self):
################################################################################
        val = 0
        vtx = self.head
        val = vtx.uid
        pos = 0
        #while pos < self.length:
        while True:
            vtx = vtx.next
            if(vtx == None):
                break
            if vtx.uid > val:
                val = vtx.uid
            pos = pos +1
        return [val,pos]

################################################################################
    def add(self,Vtx,index=None):
################################################################################
        if self.length == 0:
            self.head = Vtx
            self.tail = Vtx
            self.length = 1
        elif index == None:
            self.tail.next = Vtx
            self.tail = Vtx
            self.length = self.length +1
        else:
            Vtx.next = self.element(index)
            self.element(index-1).next = Vtx
            self.length = self.length + 1

################################################################################
    def as_np(self):
################################################################################
        dtype = [(’x’,float),(’y’,float),(’sign’,int),(’uid’,int),(’isOn’,int)]
        data =[]# np.array([],dtype=dtype)
        i = 0
        vtx = self.head
        while vtx != None:
            data.append([vtx.x, vtx.y, vtx.sign, vtx.uid, vtx.isOn])
            vtx = vtx.next
            i = i+1
        return (data)

################################################################################
    def write_out(self,time,data):
################################################################################
        np.savetxt(’vort_ord_’+str(time)+’.csv’,data,fmt=’%10.5f,%10.5f,%i,%i,%i’,delimiter=’,’)

################################################################################
    def idx_min_dist(self,vortex, isSelf=False):
################################################################################
        counter = 0
        ret_idx = counter
        vtx = self.head
        if vtx != None:
            r = vtx.dist(vortex)
            while vtx.next != None:
                vtx = vtx.next
                counter = counter +1
                if r > vtx.dist(vortex):
                    r = vtx.dist(vortex)
                    ret_idx = counter
        return (ret_idx,r)

##############################################################################
    def remove(self,pos):
##############################################################################
        if self.length > 1 and pos > 1:
            current = self.element(pos-1).next
            self.element(pos - 1).next = current.next
            current.next = None
            self.length = self.length - 1
            return current
        elif pos == 0:
            current = self.head
            self.head = self.head.next
            self.length = self.length - 1
            return current
        else:
            self.head = None
            self.length = 0
            return None

##############################################################################
    def swap_uid(self,uid_i,uid_f):
##############################################################################
        vtx_pos = self.vtx_uid(uid_i)
        self.remove(pos_i)
        self.add(vtx,index=pos_f)

##############################################################################
    def vort_decrease(self,positions,vorts_p):
##############################################################################
        max_uid = vorts_p.max_uid()
        for i4 in positions:
            vtx = cp.copy(i4)
            vtx.update_on(False)
            vtx.update_next(None)
            self.add(vtx)

##############################################################################
    def vort_increase(self,positions,vorts_p):
##############################################################################
        counter = 1
        max_uid = vorts_p.max_uid()
        for i4 in positions:
            self.element(i4).update_uid(max_uid[0] + counter)
            counter = counter+1

##############################################################################
def do_the_thing(start,fin,incr):
##############################################################################
    #v_arr_p=genfromtxt(’vort_lsq_’ + str(0) + ’.csv’,delimiter=’,’)
    v_arr_p=genfromtxt(’vort_lsq_’ + str(0) + ’.csv’,delimiter=’,’)
    for i in range(start+incr, fin+1, incr): #loop over samples in time
        vorts_p = VtxList()
        vorts_c = VtxList()
        #v_arr_c=genfromtxt(’vort_lsq_’ + str(i) + ’.csv’,delimiter=’,’ )
        v_arr_c=genfromtxt(’vort_lsq_’ + str(i) + ’.csv’,delimiter=’,’ )
        v_arr_p_coords = np.array([[a for a in v][:2] for v in v_arr_p])
        v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])
        v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])
        v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])
        for i1 in range(0,v_arr_p_coords.size/2): #loop over coordinates for a given time
            vtx_p = Vortex(i1,v_arr_p_coords[i1][0],v_arr_p_coords[i1][1],True,sign=v_arr_p_sign[i1])#,v_arr_p[i1][2])
            vorts_p.add(vtx_p)

        for i2 in range(0,v_arr_c_coords.size/2):
            vtx_c = Vortex(-1-i2,v_arr_c_coords[i2][0],v_arr_c_coords[i2][1],True,sign=v_arr_c_sign[i2])#,v_arr_p[i1][0])
            vorts_c.add(vtx_c)

        for i3 in range(0,vorts_p.length):
            index_r = vorts_c.idx_min_dist(vorts_p.element(i3))

            v0c = vorts_c.element(index_r[0]).sign
            v0p = vorts_p.element(i3).sign
            v1c = vorts_c.element(index_r[0]).uid
            if (index_r[1] < 7) and (vorts_c.element(index_r[0]).sign == vorts_p.element(i3).sign) and (vorts_c.element(index_r[0]).uid < 0):
            #if (index_r[1] < 2) and (vorts_c.element(index_r[0]).sign > 0) and (vorts_c.element(index_r[0]).uid < 0):
                vorts_c.element(index_r[0]).update_uid(vorts_p.element(i3).uid)
                vorts_c.element(index_r[0]).update_on(True)

        #You will never remember why this works
        uid_c = [[a for a in b][3] for b in vorts_c.as_np()]
        uid_p = [[a for a in b][3] for b in vorts_p.as_np()]

        dpc = set(uid_p).difference(set(uid_c))
        dcp = set(uid_c).difference(set(uid_p))
        vtx_pos_p=[]
        vtx_pos_c=[]
        for i5 in dpc:
            vtx_pos_p = np.append(vtx_pos_p,vorts_p.vtx_uid(i5)[0])
        for i6 in dcp:
            vtx_pos_c = np.append(vtx_pos_c,vorts_c.vtx_uid(i6)[1])
        if len(dpc or dcp) >= 1:
            vorts_c.vort_decrease(vtx_pos_p,vorts_p)
            vorts_c.vort_increase(vtx_pos_c,vorts_p)

        vorts_c_update=sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])
        vorts_c.write_out(i,np.asarray(vorts_c_update))
        print "[" + str(i) +"]", "Length of previous=" + str(len(v_arr_p_coords)), "Length of current=" +
str(len(vorts_c_update))
        v_arr_p=genfromtxt('vort_ord_' + str(i) + '.csv',delimiter=',' )

###############################################################################
###############################################################################
do_the_thing(0,200000,500)
```

4.55 src/ds.cc File Reference

```
#include "../include/ds.h"
```

Include dependency graph for ds.cc:

Functions

- void initArr (Array ∗arr, size_t initLen)
- void appendData (Array ∗arr, char ∗t, double d)
- void freeArray (Array ∗arr)
- Param newParam (char ∗t, double d)

4.55.1 Function Documentation

4.55.1.1 void appendData (Array ∗ *arr,* char ∗ *t,* double *d*)

Definition at line 27 of file ds.cc.

References Array::array, Array::length, newParam(), overlap::p, and Array::used.

Referenced by evolve(), initialise(), optLatSetup(), and parseArgs().

```
00027                                               {
00028     Param p = newParam(t,d);
00029     if(arr->used == arr->length){
00030         arr->length *= 2;
00031         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
      Param));
00032     }
00033     arr->array[arr->used] = p;
00034     arr->used = arr->used + 1;
00035 }
```

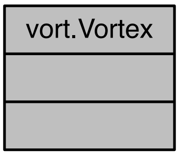
Here is the call graph for this function:

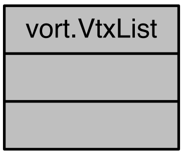
Here is the caller graph for this function:

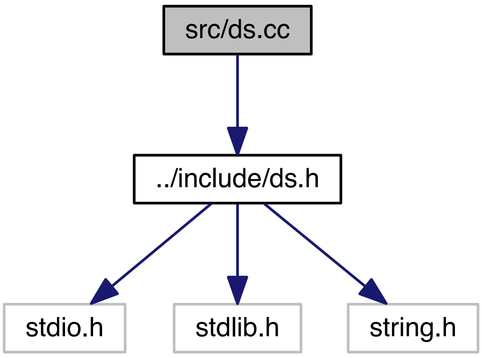
4.55.1.2 void freeArray (Array ∗ *arr*)

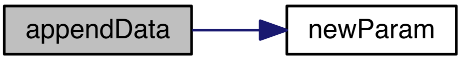
Definition at line 37 of file ds.cc.

References Array::array, Array::length, and Array::used.


vort.Vortex


vort.VtxList


src/ds.cc
../include/ds.h
stdio.h
stdlib.h
string.h


appendData
newParam

```
00037                            {
00038     free(arr->array);
00039     arr->array = NULL;
00040     arr->used = 0;
00041     arr->length = 0;
00042 }
```

4.55.1.3 void initArr (Array ∗ *arr,* size_t *initLen*)

Definition at line 21 of file ds.cc.

References Array::array, Array::length, and Array::used.

Referenced by main().

```
00021                                          {
00022     arr->array = (Param*) malloc(initLen*sizeof(Param));
00023     arr->used = 0;
00024     arr->length = initLen;
00025 }
```

Here is the caller graph for this function:

4.55.1.4 Param newParam (char ∗ *t,* double *d*)

Definition at line 44 of file ds.cc.

References Param::data, overlap::p, and Param::title.

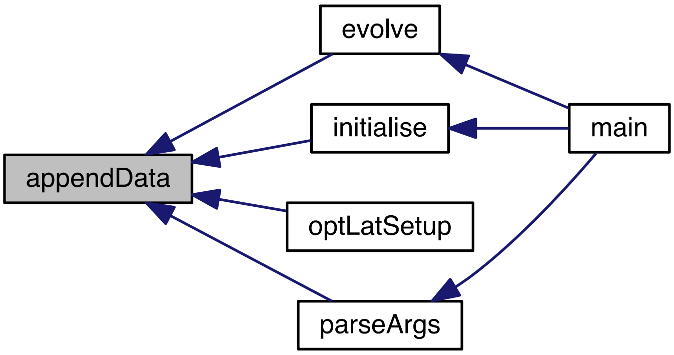
Referenced by appendData().

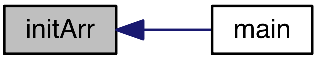
```
00044                                  {
00045     Param p;
00046     strcpy(p.title,t);
00047     p.data = d;
00048     return p;
00049 }
```

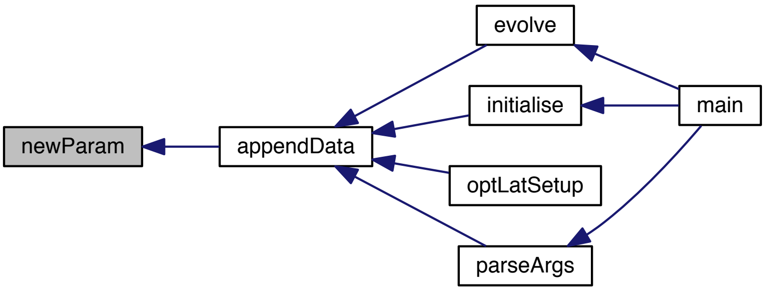
Here is the caller graph for this function:

4.56 ds.cc

```
00001 /*
00002 * ds.cc - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
00011 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
00012 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00013 * License for more details. You should have received a copy of the GNU
00014 * Lesser General Public License along with this library; if not, write
00015 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
00016 * Boston, MA 02111-1307 USA
00017 */
00018
00019 #include "../include/ds.h"
00020
00021 void initArr(Array *arr, size_t initLen){
00022     arr->array = (Param*) malloc(initLen*sizeof(Param));
00023     arr->used = 0;
00024     arr->length = initLen;
00025 }
00026
00027 void appendData(Array *arr, char* t, double d){
00028     Param p = newParam(t,d);
```


evolve
initialise
main
appendData
optLatSetup
parseArgs


initArr
main


evolve
initialise
main
newParam
appendData
optLatSetup
parseArgs

```
    if(arr->used == arr->length){
        arr->length *= 2;
        arr->array = (Param*)realloc(arr->array, arr->length*sizeof(Param));
    }
    arr->array[arr->used] = p;
    arr->used = arr->used + 1;
}

void freeArray(Array *arr){
    free(arr->array);
    arr->array = NULL;
    arr->used = 0;
    arr->length = 0;
}

Param newParam(char* t,double d){
    Param p;
    strcpy(p.title,t);
    p.data = d;
    return p;
}
```

4.57 src/fileIO.cc File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cuda_runtime.h>
#include <hdf5.h>
#include <hdf5_hl.h>
#include "../include/fileIO.h"
```

Include dependency graph for fileIO.cc: This graph shows which files directly or indirectly include this file:

Functions

- void hdfWriteDouble (int xDim, double ∗op, long incr, char ∗dset)
- void hdfWriteComplex (int xDim, double2 ∗wfc, long incr, char ∗dset)
- double2 ∗ readIn (char ∗fileR, char ∗fileI, int xDim, int yDim)
- void writeOutParam (char ∗buffer, Array arr, char ∗file)
- void writeOut (char ∗buffer, char ∗file, double2 ∗data, int length, int step)
- void writeOutDouble (char ∗buffer, char ∗file, double ∗data, int length, int step)
- void writeOutInt (char ∗buffer, char ∗file, int ∗data, int length, int step)
- void writeOutInt2 (char ∗buffer, char ∗file, int2 ∗data, int length, int step)
- void writeOutVortex (char ∗buffer, char ∗file, struct Vortex ∗data, int length, int step)
- int readState (char ∗name)

4.57.1 Function Documentation

4.57.1.1 void hdfWriteComplex (int *xDim,* double2 ∗ *wfc,* long *incr,* char ∗ *dset*)

Definition at line 46 of file fileIO.cc.

References x, xDim, and y.

```
00046                                                                      {
00047
00048     typedef struct d2{
00049         double x;
00050         double y;
00051     }d2;
00052     hid_t file_id;
00053     hsize_t dims[2];
00054     dims[0]=xDim;
00055     dims[1]=xDim;
00056     herr_t status;
00057     double2 tmp;
00058     hid_t complex_id = H5Tcreate(H5T_COMPOUND, sizeof(tmp));
00059     H5Tinsert (complex_id, "real", HOFFSET(d2,x), H5T_NATIVE_DOUBLE);
00060     H5Tinsert (complex_id, "imaginary", HOFFSET(d2,y), H5T_NATIVE_DOUBLE);
00061 
00062     char dataset[32];
00063     strcpy(dataset,"/");
00064     strcat(dataset,dset);
00065     if(incr==0){
00066         file_id = H5Fcreate("GPUE.h5",H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
00067     }
00068     else{
00069         file_id = H5Fopen( "GPUE.h5", H5F_ACC_RDWR, H5P_DEFAULT );
00070     }
00071         status = H5LTmake_dataset( file_id, dset, 2, dims, complex_id, wfc );
00072 
00073         status = H5Fclose(file_id);
00074 }
```

4.57.1.2 void hdfWriteDouble (int *xDim*, double ∗ *op*, long *incr*, char ∗ *dset*)

Definition at line 27 of file fileIO.cc.

References xDim.

```
00027                                                                 {
00028     hid_t file_id;
00029     hsize_t dims[2];
00030     dims[0]=xDim;
00031     dims[1]=xDim;
00032     herr_t status;
00033     char dataset[32];
00034     strcpy(dataset,"/");
00035     strcat(dataset,dset);
00036     if(incr==0){
00037         file_id = H5Fcreate("GPUE.h5",H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
00038     }
00039     else{
00040         file_id = H5Fopen( "GPUE.h5", H5F_ACC_RDWR, H5P_DEFAULT );
00041     }
00042         status = H5LTmake_dataset( file_id, dset, 2, dims, H5T_NATIVE_DOUBLE, op );
00043 
00044         status = H5Fclose(file_id);
00045 }
```

4.57.1.3 double2∗ readIn (char ∗ *fileR*, char ∗ *fileI*, int *xDim*, int *yDim*)

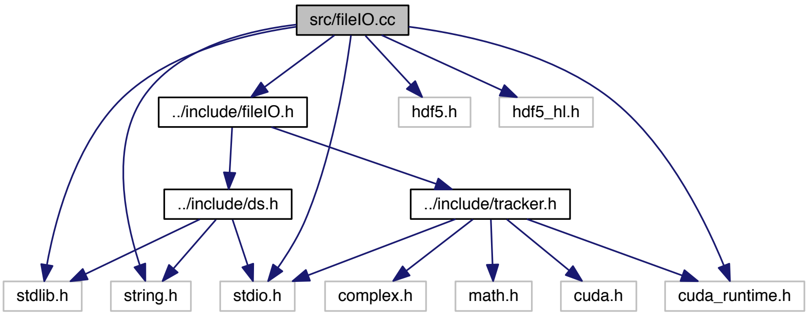
Definition at line 76 of file fileIO.cc.

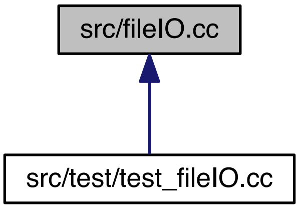
References vis_ev::f, FILE, vis::i, and yDim.

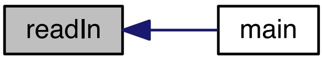
Referenced by main().

```
00076                                                              {
00077     FILE *f;
00078     f = fopen(fileR,"r");
00079     int i = 0;
00080     double2 *arr = (double2*) malloc(sizeof(double2)*xDim*yDim);
00081     double line;
00082     while(fscanf(f,"%lE",&line) > 0){
00083         arr[i].x = line;
00084         ++i;
00085     }
00086     fclose(f);
00087     f = fopen(fileI,"r");
00088     i = 0;
00089     while(fscanf(f,"%lE",&line) > 0){
00090         arr[i].y = line;
00091         ++i;
00092     }
00093     fclose(f);
00094     return arr;
00095 }
```

Here is the caller graph for this function:


src/fileIO.cc
../include/fileIO.h
hdf5.h
hdf5_hl.h
../include/ds.h
../include/tracker.h
stdlib.h
string.h
stdio.h
complex.h
math.h
cuda.h
cuda_runtime.h


src/fileIO.cc
src/test/test_fileIO.cc


readIn
main

4.57.1.4 int readState (char ∗ *name*)

Definition at line 166 of file fileIO.cc.

References vis_ev::f, and FILE.

```
00166                           {
00167     FILE *f;
00168     f = fopen(name,"r");
00169     fclose(f);
00170     return 0;
00171 }
```

4.57.1.5 void writeOut (char ∗ *buffer,* char ∗ *file,* double2 ∗ *data,* int *length,* int *step*)

Definition at line 109 of file fileIO.cc.

References vis_ev::f, FILE, vis::i, x, and y.

Referenced by evolve(), initialise(), and main().

```
00109                                                                            {
00110     FILE *f;
00111     sprintf (buffer, "%s_%d", file, step);
00112     f = fopen (buffer,"w");
00113     int i;
00114     for (i = 0; i < length; i++)
00115         fprintf (f, "%.16e\n",data[i].x);
00116     fclose (f);
00117 
00118     sprintf (buffer, "%si_%d", file, step);
00119     f = fopen (buffer,"w");
00120     for (i = 0; i < length; i++)
00121         fprintf (f, "%.16e\n",data[i].y);
00122     fclose (f);
00123 }
```

Here is the caller graph for this function:

4.57.1.6 void writeOutDouble (char ∗ *buffer,* char ∗ *file,* double ∗ *data,* int *length,* int *step*)

Definition at line 125 of file fileIO.cc.

References vis_ev::f, FILE, and vis::i.

Referenced by evolve(), initialise(), and main().

```
00125                                                                                 {
00126     FILE *f;
00127     sprintf (buffer, "%s_%d", file, step);
00128     f = fopen (buffer,"w");
00129     int i;
00130     for (i = 0; i < length; i++)
00131         fprintf (f, "%.16e\n",data[i]);
00132     fclose (f);
00133 }
```

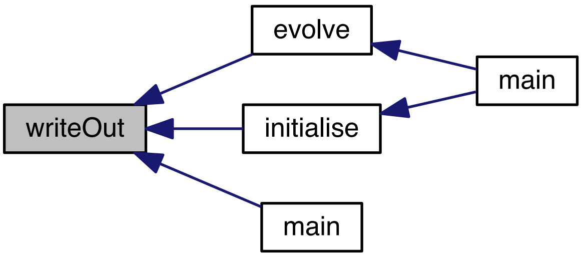
Here is the caller graph for this function:

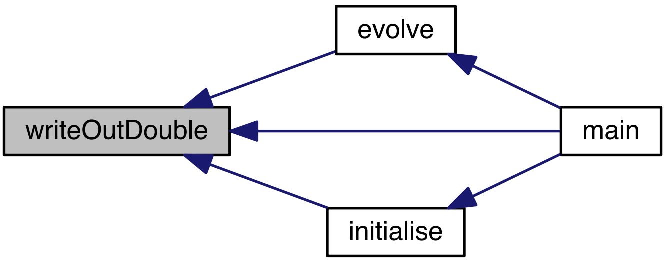
4.57.1.7 void writeOutInt (char ∗ *buffer,* char ∗ *file,* int ∗ *data,* int *length,* int *step*)

Definition at line 135 of file fileIO.cc.

References vis_ev::f, FILE, and vis::i.

```
00135                                                                           {
00136     FILE *f;
00137     sprintf (buffer, "%s_%d", file, step);
```


evolve
main
writeOut
initialise
main


evolve
writeOutDouble
main
initialise

```
00138     f = fopen (buffer,"w");
00139     int i;
00140     for (i = 0; i < length; i++)
00141         fprintf (f, "%d\n",data[i]);
00142     fclose (f);
00143 }
```

4.57.1.8 void writeOutInt2 (char ∗ *buffer,* char ∗ *file,* int2 ∗ *data,* int *length,* int *step*)

Definition at line 145 of file fileIO.cc.

References vis_ev::f, FILE, vis::i, x, and y.

```
00145                                                                                   {
00146     FILE *f;
00147     sprintf (buffer, "%s_%d", file, step);
00148     f = fopen (buffer,"w");
00149     int i;
00150     for (i = 0; i < length; i++)
00151         fprintf (f, "%d,%d\n",data[i].x,data[i].y);
00152     fclose (f);
00153 }
```

4.57.1.9 void writeOutParam (char ∗ *buffer,* Array *arr,* char ∗ *file*)

Definition at line 97 of file fileIO.cc.

References Array::array, Param::data, vis_ev::f, FILE, vis::i, Param::title, and Array::used.

Referenced by evolve(), and main().

```
00097                                                      {
00098     FILE *f;
00099     sprintf(buffer, "%s", file);
00100     f = fopen(file,"w");
00101     fprintf(f,"[Params]\n");
00102     for (int i = 0; i < arr.used; ++i){
00103         fprintf(f,"%s=",arr.array[i].title);
00104         fprintf(f,"%e\n",arr.array[i].data);
00105     }
00106     fclose(f);
00107 }
```

Here is the caller graph for this function:

4.57.1.10 void writeOutVortex (char ∗ *buffer,* char ∗ *file,* struct Vortex ∗ *data,* int *length,* int *step*)

Definition at line 155 of file fileIO.cc.

References Vortex::coords, vis_ev::f, FILE, vis::i, Vortex::sign, and Vortex::wind.

Referenced by evolve().

```
00155                                                                                              {
00156     FILE *f;
00157     sprintf (buffer, "%s_%d", file, step);
00158     f = fopen (buffer,"w");
00159     int i;
00160     fprintf (f, "#X,Y,WINDING,SIGN\n");
00161     for (i = 0; i < length; i++)
00162         fprintf (f, "%d,%d,%d,%d\n",data[i].coords.x,data[i].coords.y,data[i].
      wind,data[i].sign);
00163     fclose (f);
00164 }
```

Here is the caller graph for this function:

4.58 fileIO.cc

```
/*
* fileIO.c - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cuda_runtime.h>
#include <hdf5.h>
#include <hdf5_hl.h>
#include "../include/fileIO.h"

void hdfWriteDouble(int xDim, double* op, long incr, char* dset){
    hid_t file_id;
    hsize_t dims[2];
    dims[0]=xDim;
    dims[1]=xDim;
    herr_t status;
    char dataset[32];
    strcpy(dataset,"/");
    strcat(dataset,dset);
    if(incr==0){
        file_id = H5Fcreate("GPUE.h5",H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
    }
    else{
        file_id = H5Fopen( "GPUE.h5", H5F_ACC_RDWR, H5P_DEFAULT );
    }
        status = H5LTmake_dataset( file_id, dset, 2, dims, H5T_NATIVE_DOUBLE, op );

        status = H5Fclose(file_id);
}
void hdfWriteComplex(int xDim, double2* wfc, long incr, char* dset){

    typedef struct d2{
        double x;
        double y;
    }d2;
    hid_t file_id;
    hsize_t dims[2];
    dims[0]=xDim;
    dims[1]=xDim;
    herr_t status;
    double2 tmp;
    hid_t complex_id = H5Tcreate(H5T_COMPOUND, sizeof(tmp));
    H5Tinsert (complex_id, "real", HOFFSET(d2,x), H5T_NATIVE_DOUBLE);
    H5Tinsert (complex_id, "imaginary", HOFFSET(d2,y), H5T_NATIVE_DOUBLE);

    char dataset[32];
    strcpy(dataset,"/");
    strcat(dataset,dset);
    if(incr==0){
        file_id = H5Fcreate("GPUE.h5",H5F_ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
    }
    else{
        file_id = H5Fopen( "GPUE.h5", H5F_ACC_RDWR, H5P_DEFAULT );
    }
        status = H5LTmake_dataset( file_id, dset, 2, dims, complex_id, wfc );

        status = H5Fclose(file_id);
}

double2* readIn(char* fileR, char* fileI, int xDim, int yDim){
    FILE *f;
    f = fopen(fileR,"r");
    int i = 0;
    double2 *arr = (double2*) malloc(sizeof(double2)*xDim*yDim);
    double line;
    while(fscanf(f,"%lE",&line) > 0){
        arr[i].x = line;
        ++i;
    }
    fclose(f);
    f = fopen(fileI,"r");
    i = 0;
    while(fscanf(f,"%lE",&line) > 0){
        arr[i].y = line;
        ++i;
    }
    fclose(f);
    return arr;
}

void writeOutParam(char* buffer, Array arr, char *file){
    FILE *f;
    sprintf(buffer, "%s", file);
    f = fopen(file,"w");
    fprintf(f,"[Params]\n");
    for (int i = 0; i < arr.used; ++i){
        fprintf(f,"%s=",arr.array[i].title);
        fprintf(f,"%e\n",arr.array[i].data);
    }
    fclose(f);
}

void writeOut(char* buffer, char *file, double2 *data, int length, int step){
    FILE *f;
    sprintf (buffer, "%s_%d", file, step);
    f = fopen (buffer,"w");
    int i;
    for (i = 0; i < length; i++)
        fprintf (f, "%.16e\n",data[i].x);
    fclose (f);

    sprintf (buffer, "%si_%d", file, step);
    f = fopen (buffer,"w");
    for (i = 0; i < length; i++)
        fprintf (f, "%.16e\n",data[i].y);
    fclose (f);
}

void writeOutDouble(char* buffer, char *file, double *data, int length, int step){
    FILE *f;
    sprintf (buffer, "%s_%d", file, step);
    f = fopen (buffer,"w");
    int i;
    for (i = 0; i < length; i++)
        fprintf (f, "%.16e\n",data[i]);
    fclose (f);
}

void writeOutInt(char* buffer, char *file, int *data, int length, int step){
    FILE *f;
    sprintf (buffer, "%s_%d", file, step);
    f = fopen (buffer,"w");
    int i;
    for (i = 0; i < length; i++)
        fprintf (f, "%d\n",data[i]);
    fclose (f);
}

void writeOutInt2(char* buffer, char *file, int2 *data, int length, int step){
    FILE *f;
    sprintf (buffer, "%s_%d", file, step);
    f = fopen (buffer,"w");
    int i;
    for (i = 0; i < length; i++)
        fprintf (f, "%d,%d\n",data[i].x,data[i].y);
    fclose (f);
}

void writeOutVortex(char* buffer, char *file, struct Vortex *data, int length, int step){
    FILE *f;
    sprintf (buffer, "%s_%d", file, step);
    f = fopen (buffer,"w");
    int i;
    fprintf (f, "#X,Y,WINDING,SIGN\n");
    for (i = 0; i < length; i++)
        fprintf (f, "%d,%d,%d,%d\n",data[i].coords.x,data[i].coords.y,data[i].wind,data[i].sign);
    fclose (f);
}

int readState(char* name){
    FILE *f;
    f = fopen(name,"r");
    fclose(f);
    return 0;
}
```

4.59 src/gpu_functions.cu File Reference

Macros

- #define TILE_DIM 32
- #define BLOCK_ROW 4

Functions

- __device__ unsigned int getGid3d3d ()
- __global__ void scalVecMult_d2d (double2 ∗vecIn, double scalIn, double2 ∗vecOut)
- __global__ void scalVecMult_dd (double ∗vecIn, double scalIn, double ∗vecOut)
- __global__ void scalVecMult_ii (int ∗vecIn, int scalIn, int ∗vecOut)
- __global__ void scalVecMult_d2d2 (double2 ∗vecIn, double2 scalIn, double2 ∗vecOut)
- __global__ void vecVecMult_d2d2 (double2 ∗vec1In, double2 ∗vec2In, double2 ∗vecOut)
- __global__ void vecVecMult_d2d (double2 ∗vec1In, double ∗vec2In, double2 ∗vecOut)
- __global__ void vecVecMult_dd (double ∗vec1In, double ∗vec2In, double ∗vecOut)
- __global__ void vecVecMult_ii (int ∗vec1In, int ∗vec2In, int ∗vecOut)
- __global__ void matTrans (double2 ∗vecIn, double2 ∗vecOut)
- template<unsigned int blockSize>
  __global__ void sumVector_d (double ∗vecIn, double ∗vecOut, unsigned int n)
- template<unsigned int blockSize>
  __global__ void sumVector_d2 (double2 ∗vecIn, double2 ∗vecOut, unsigned int n)
- __host__ __device__ double2 compMagnitude (double2 cmp1)
- __host__ __device__ double2 realCompMult (double rl, double2 cmp)
- __host__ __device__ double2 compCompMult (double2 cmp1, double2 cmp2)
- __host__ __device__ double2 compSum (double2 cmp1, double2 cmp2)
- __host__ __device__ double2 conj (double2 cmp)

4.59.1 Macro Definition Documentation

4.59.1.1 #define BLOCK_ROW 4

Definition at line 3 of file gpu_functions.cu.

4.59.1.2 #define TILE_DIM 32

Definition at line 2 of file gpu_functions.cu.

Referenced by matTrans().

4.59.2 Function Documentation

4.59.2.1 __host__ __device__ double2 compCompMult (double2 *cmp1*, double2 *cmp2*)

Definition at line 186 of file gpu_functions.cu.

Referenced by scalVecMult_d2d2(), vecVecMult_d2d2(), and vecVecMult_ii().

```
00186                                                                      {
00187
00188 }
```

Here is the caller graph for this function:

4.59.2.2 __host__ __device__ double compMagnitude (double2 *cmp1*)

Definition at line 177 of file gpu_functions.cu.

```
00177                                                        {
00178     return sqrt(cmp1.x*cmp1.x + cmp1.y*cmp1.y);
00179 }
```

4.59.2.3 __host__ __device__ double2 compSum (double2 *cmp1,* double2 *cmp2*)

4.59.2.4 __host__ __device__ double2 conj (double2 *cmp*)

Definition at line 95 of file minions.cc.

References hist3d::c, and result.

Referenced by complexDiv().

```
00095                              {
00096     double2 result = c;
00097     result.y = -result.y;
00098     return result;
00099 }
```

Here is the caller graph for this function:

4.59.2.5 __device__ unsigned int getGid3d3d ( )

Definition at line 10 of file gpu_functions.cu.

Referenced by scalVecMult_d2d(), scalVecMult_d2d2(), scalVecMult_dd(), scalVecMult_ii(), vecVecMult_d2d(), vecVecMult_d2d2(), vecVecMult_dd(), and vecVecMult_ii().

```
00010                                   {
00011     int gid = blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
      threadIdx.y ) + blockIdx.x ) + threadIdx.x;
00012     return gid;
00013 }
```

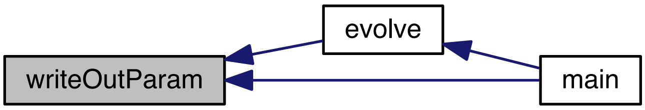
Here is the caller graph for this function:

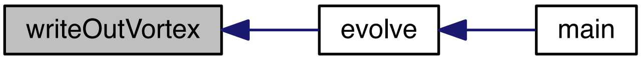
4.59.2.6 __global__ void matTrans (double2 ∗ *vecIn,* double2 ∗ *vecOut*)

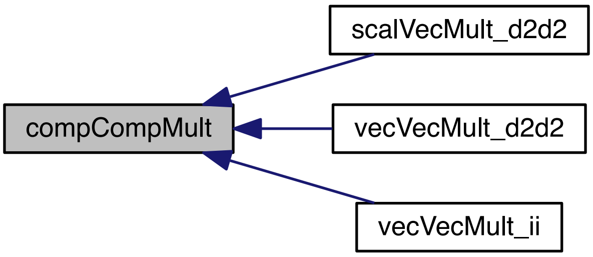
Definition at line 78 of file gpu_functions.cu.

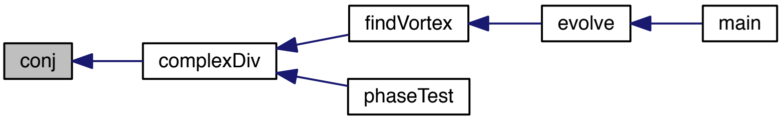
References TILE_DIM, x, xDim, and y.

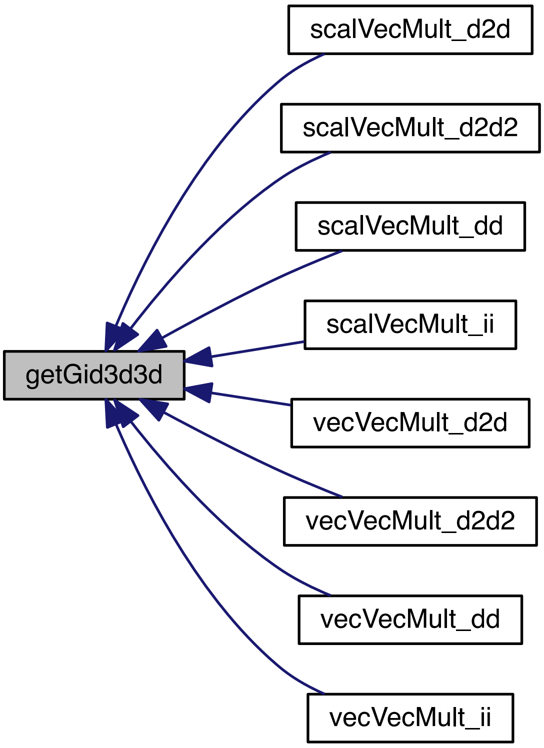
```
00078                                                               {
00079
00080     int x = blockIdx.x * TILE_DIM + threadIdx.x;
00081     int y = blockIdx.y * TILE_DIM + threadIdx.y;
00082     int width = gridDim.x * TILE_DIM;
00083
00084     for(int j=0; j<xDim; j+=xDim){
00085         vecOut[ x*width + (y+j)  ] = vecIn[(y+j)*width + x];
00086     }
00087
00088 /*  unsigned int i = getGid3d3d();
00089     int bid = blockIdx.x*blockDim.x;
00090     int width = gridDim.x*blockDim.x;
00091 */}
```


writeOutParam
evolve
main


writeOutVortex
evolve
main


compCompMult
scalVecMult_d2d2
vecVecMult_d2d2
vecVecMult_ii


conj
complexDiv
findVortex
phaseTest
evolve
main


scalVecMult_d2d
scalVecMult_d2d2
scalVecMult_dd
scalVecMult_ii
getGid3d3d
vecVecMult_d2d
vecVecMult_d2d2
vecVecMult_dd
vecVecMult_ii

4.59.2.7 __host__ __device__ double2 realCompMult (double *rl*, double2 *cmp*)

Definition at line 180 of file gpu_functions.cu.

References result.

Referenced by scalVecMult_d2d(), and vecVecMult_d2d().

```
00180                                                               {
00181     double2 result;
00182     result.x = rl*cmp1.x;
00183     result.y = rl*cmp1.y;
00184     return result;
00185 }
```

Here is the caller graph for this function:

4.59.2.8 __global__ void scalVecMult_d2d (double2 ∗ *vecIn*, double *scalIn*, double2 ∗ *vecOut*)

Definition at line 22 of file gpu_functions.cu.

References getGid3d3d(), vis::i, and realCompMult().

```
00022                                                                           {
00023     unsigned int i = getGid3d3d();
00024     vecOut[i] = realCompMult(scalIn,vecIn[i]);
00025 }
```

Here is the call graph for this function:

4.59.2.9 __global__ void scalVecMult_d2d2 (double2 ∗ *vecIn*, double2 *scalIn*, double2 ∗ *vecOut*)

Definition at line 37 of file gpu_functions.cu.

References compCompMult(), getGid3d3d(), and vis::i.

```
00037                                                                             {
00038     unsigned int i = getGid3d3d();
00039     vecOut[i] = compCompMult(scalIn, vecIn[i]);
00040 }
```

Here is the call graph for this function:

4.59.2.10 __global__ void scalVecMult_dd (double ∗ *vecIn*, double *scalIn*, double ∗ *vecOut*)

Definition at line 27 of file gpu_functions.cu.

References getGid3d3d(), and vis::i.

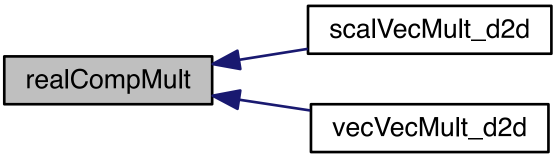
```
00027                                                                         {
00028     unsigned int i = getGid3d3d();
00029     vecOut[i] = scalIn*vecIn[i];
00030 }
```

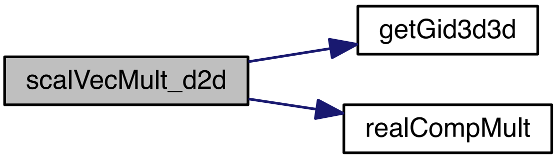
Here is the call graph for this function:

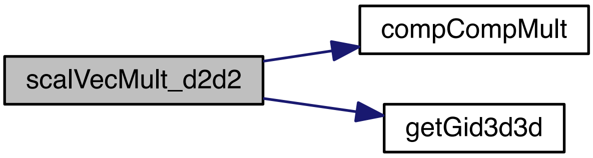
4.59.2.11 __global__ void scalVecMult_ii (int ∗ *vecIn*, int *scalIn*, int ∗ *vecOut*)

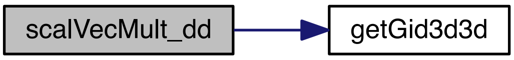
Definition at line 32 of file gpu_functions.cu.

References getGid3d3d(), and vis::i.


realCompMult
scalVecMult_d2d
vecVecMult_d2d


scalVecMult_d2d
getGid3d3d
realCompMult


scalVecMult_d2d2
compCompMult
getGid3d3d


scalVecMult_dd
getGid3d3d

```
00032                                                                                     {
00033     unsigned int i = getGid3d3d();
00034     vecOut[i] = scalIn*vecIn[i];
00035 }
```

Here is the call graph for this function:

4.59.2.12 template<unsigned int blockSize> __global__ void sumVector_d (double ∗ *vecIn,* double ∗ *vecOut,* unsigned int *n*)

Definition at line 107 of file gpu_functions.cu.

References vis::i.

```
00107                                                                                            {
00108     extern __shared__ double sdata[];
00109 
00110     unsigned int tid = threadIdx.x;
00111     unsigned int i = blockIdx.x*(blockSize*2) + tid;
00112     unsigned int gridSize = blockSize*2*gridDim.x;
00113     sdata[tid]=0.0;
00114 
00115     while ( i < n ){
00116         sdata[tid] += vecIn[i] + vecIn[i + blockSize];
00117         i += gridSize;
00118     }
00119     if(blockSize >= 1024) { if(tid < 512) { sdata[tid] += sdata[tid+512];} __syncthreads; }
00120     if(blockSize >= 512) { if(tid < 256) { sdata[tid] += sdata[tid+256];} __syncthreads; }
00121     if(blockSize >= 256) { if(tid < 128) { sdata[tid] += sdata[tid+128];} __syncthreads; }
00122     if(blockSize >= 128) { if(tid < 64) { sdata[tid] += sdata[tid+64];} __syncthreads; }
00123 
00124     if (tid < 32){
00125         if(blockSize >= 64) sdata[tid] += sdata[tid+32];
00126         if(blockSize >= 32) sdata[tid] += sdata[tid+16];
00127         if(blockSize >= 16) sdata[tid] += sdata[tid+8];
00128         if(blockSize >= 8) sdata[tid] += sdata[tid+4];
00129         if(blockSize >= 4) sdata[tid] += sdata[tid+2];
00130         if(blockSize >= 2) sdata[tid] += sdata[tid+1];
00131     }
00132     if(tid == 0) vecOut[blockIdx.x] = sdata[0];
00133 }
```

4.59.2.13 template<unsigned int blockSize> __global__ void sumVector_d2 (double2 ∗ *vecIn,* double2 ∗ *vecOut,* unsigned int *n*)

Definition at line 136 of file gpu_functions.cu.

References vis::i.

```
00136                                                                                               {
00137     extern __shared__ double2 sdata[];
00138 
00139     unsigned int tid = threadIdx.x;
00140     unsigned int i = blockIdx.x*(blockSize*2) + tid;
00141     unsigned int gridSize = blockSize*2*gridDim.x;
00142     sdata[tid].x=0.0;   sdata[tid].y=0.0;
00143 
00144     while ( i < n ){
00145         sdata[tid].x += vecIn[i].x + vecIn[i + blockSize].x;
00146         sdata[tid].y += vecIn[i].y + vecIn[i + blockSize].y;
00147         i += gridSize;
00148     }
00149     if(blockSize >= 1024) { if(tid < 512) { sdata[tid].x += sdata[tid+512].x; sdata[tid].y += sdata[tid+512
      ].y; } __syncthreads; }
00150     if(blockSize >= 512)  { if(tid < 256) { sdata[tid].x += sdata[tid+256].x; sdata[tid].y += sdata[tid+256
      ].y; } __syncthreads; }
00151     if(blockSize >= 256)  { if(tid < 128) { sdata[tid].x += sdata[tid+128].x; sdata[tid].y += sdata[tid+128
      ].y; } __syncthreads; }
00152     if(blockSize >= 128)  { if(tid < 64)  { sdata[tid].x += sdata[tid+64].x;  sdata[tid].y += sdata[tid+64]
      .y;  } __syncthreads; }
00153 
00154     if (tid < 32){
00155         if(blockSize >= 64){ sdata[tid].x += sdata[tid+32].x; sdata[tid].y += sdata[tid+32].y; }
00156         if(blockSize >= 32){ sdata[tid].x += sdata[tid+16].x; sdata[tid].y += sdata[tid+16].y; }
00157         if(blockSize >= 16){ sdata[tid].x += sdata[tid+8].x;  sdata[tid].y += sdata[tid+8].y; }
00158         if(blockSize >= 8){  sdata[tid].x += sdata[tid+4].x;  sdata[tid].y += sdata[tid+4].y; }
00159         if(blockSize >= 4){  sdata[tid].x += sdata[tid+2].x;  sdata[tid].y += sdata[tid+2].y; }
00160         if(blockSize >= 2){  sdata[tid].x += sdata[tid+1].x;  sdata[tid].y += sdata[tid+1].y; }
00161     }
00162     if(tid == 0) vecOut[blockIdx.x] = sdata[0];
00163 }
```

4.59.2.14 __global__ void vecVecMult_d2d (double2 ∗ *vec1In,* double ∗ *vec2In,* double2 ∗ *vecOut*)

Definition at line 55 of file gpu_functions.cu.

References getGid3d3d(), vis::i, and realCompMult().

```
00055                                                                                  {
00056     unsigned int i = getGid3d3d();
00057     vecOut[i] = realCompMult(vec2In[i],vec1In[i]);
00058 }
```

Here is the call graph for this function:

4.59.2.15 __global__ void vecVecMult_d2d2 (double2 ∗ *vec1In,* double2 ∗ *vec2In,* double2 ∗ *vecOut*)

Definition at line 50 of file gpu_functions.cu.

References compCompMult(), getGid3d3d(), and vis::i.

```
00050                                                                                    {
00051     unsigned int i = getGid3d3d();
00052     vecOut[i] = compCompMult(vec1In[i],vec2In[i]);
00053 }
```

Here is the call graph for this function:

4.59.2.16 __global__ void vecVecMult_dd (double ∗ *vec1In,* double ∗ *vec2In,* double ∗ *vecOut*)

Definition at line 60 of file gpu_functions.cu.

References getGid3d3d(), and vis::i.

```
00060                                                                              {
00061     unsigned int i = getGid3d3d();
00062     vecOut[i] = vec1In[i]*vec2In[i];
00063 }
```

Here is the call graph for this function:

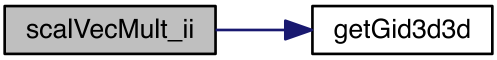
4.59.2.17 __global__ void vecVecMult_ii (int ∗ *vec1In,* int ∗ *vec2In,* int ∗ *vecOut*)

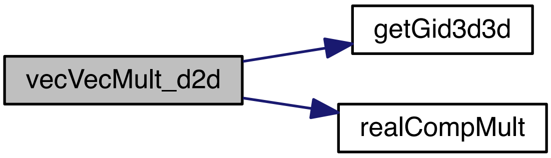
Definition at line 65 of file gpu_functions.cu.

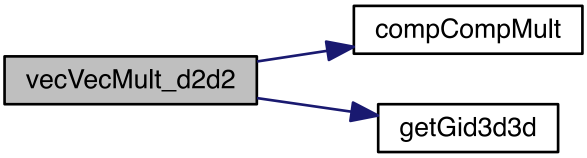
References compCompMult(), getGid3d3d(), and vis::i.

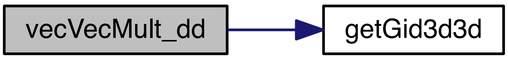
```
00065                                                                     {
00066     unsigned int i = getGid3d3d();
00067     vecOut[i] = compCompMult(vec1In[i],vec1In[i]);
00068 }
```

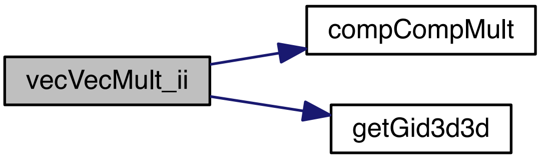
Here is the call graph for this function:


scalVecMult_ii
getGid3d3d


vecVecMult_d2d
getGid3d3d
realCompMult


vecVecMult_d2d2
compCompMult
getGid3d3d


vecVecMult_dd
getGid3d3d


vecVecMult_ii
compCompMult
getGid3d3d

4.60 gpu_functions.cu

```
#ifndef T32B4
    #define TILE_DIM 32 //small segment to be computed
    #define BLOCK_ROW 4 // sum of the two should match threads
#endif


/*
*  Returns the global (not grid) index for the relevant thread in a 3d grid 3d block fashion. I will use 1d 1d mostly here though.
*/
__device__ unsigned int getGid3d3d(){
    int gid = blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) + threadIdx.y ) + blockIdx.x ) + threadIdx.x;
    return gid;
}


//#############################################################################################################//
/*
*  Scalar x Vector functions. Double-Double, Double-Complex, Complex-Complex, Int-Int
*/
//#############################################################################################################//

__global__ void scalVecMult_d2d(double2 *vecIn, double scalIn, double2 *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = realCompMult(scalIn,vecIn[i]);
}

__global__ void scalVecMult_dd(double *vecIn, double scalIn, double *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = scalIn*vecIn[i];
}

__global__ void scalVecMult_ii(int *vecIn, int scalIn, int *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = scalIn*vecIn[i];
}

__global__ void scalVecMult_d2d2(double2 *vecIn, double2 scalIn, double2 *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = compCompMult(scalIn, vecIn[i]);
}

//#############################################################################################################//

//#############################################################################################################//
/*
*  Vector x Vector functions. Double-Double, Double-Complex, Complex-Complex, Int-Int
*/
//#############################################################################################################//

__global__ void vecVecMult_d2d2(double2 *vec1In, double2 *vec2In, double2 *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = compCompMult(vec1In[i],vec2In[i]);
}

__global__ void vecVecMult_d2d(double2 *vec1In, double *vec2In, double2 *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = realCompMult(vec2In[i],vec1In[i]);
}

__global__ void vecVecMult_dd(double *vec1In, double *vec2In, double *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = vec1In[i]*vec2In[i];
}

__global__ void vecVecMult_ii(int *vec1In, int *vec2In, int *vecOut){
    unsigned int i = getGid3d3d();
    vecOut[i] = compCompMult(vec1In[i],vec1In[i]);
}

//#############################################################################################################//

//#############################################################################################################//
/*
*  Matrix transpose function. Double-Double, Double-Complex, Complex-Complex, Int-Int
*/
```

```
// #############################################################################################################//

__global__ void matTrans(double2 *vecIn, double2 *vecOut){

    int x = blockIdx.x * TILE_DIM + threadIdx.x;
    int y = blockIdx.y * TILE_DIM + threadIdx.y;
    int width = gridDim.x * TILE_DIM;

    for(int j=0; j<xDim; j+=xDim){
        vecOut[ x*width + (y+j)  ] = vecIn[(y+j)*width + x];
    }

/*  unsigned int i = getGid3d3d();
    int bid = blockIdx.x*blockDim.x;
    int width = gridDim.x*blockDim.x;
*/}


// #############################################################################################################//

// #############################################################################################################//
/*
*  Parallel summation. Double, Complex
*/
// #############################################################################################################//

//Taken from cuda slide 1.1-beta
/*
* n is the number of elements to sum by a single thread. Values of 64-2048 are best, allegedly.
*/
template <unsigned int blockSize>
__global__ void sumVector_d(double* vecIn, double* vecOut, unsigned int n){
    extern __shared__ double sdata[];

    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(blockSize*2) + tid;
    unsigned int gridSize = blockSize*2*gridDim.x;
    sdata[tid]=0.0;

    while ( i < n ){
        sdata[tid] += vecIn[i] + vecIn[i + blockSize];
        i += gridSize;
    }
    if(blockSize >= 1024) { if(tid < 512) { sdata[tid] += sdata[tid+512];} __syncthreads; }
    if(blockSize >= 512) { if(tid < 256) { sdata[tid] += sdata[tid+256];} __syncthreads; }
    if(blockSize >= 256) { if(tid < 128) { sdata[tid] += sdata[tid+128];} __syncthreads; }
    if(blockSize >= 128) { if(tid < 64) { sdata[tid] += sdata[tid+64];} __syncthreads; }

    if (tid < 32){
        if(blockSize >= 64) sdata[tid] += sdata[tid+32];
        if(blockSize >= 32) sdata[tid] += sdata[tid+16];
        if(blockSize >= 16) sdata[tid] += sdata[tid+8];
        if(blockSize >= 8) sdata[tid] += sdata[tid+4];
        if(blockSize >= 4) sdata[tid] += sdata[tid+2];
        if(blockSize >= 2) sdata[tid] += sdata[tid+1];
    }
    if(tid == 0) vecOut[blockIdx.x] = sdata[0];
}

template <unsigned int blockSize>
__global__ void sumVector_d2(double2* vecIn, double2* vecOut, unsigned int n){
    extern __shared__ double2 sdata[];

    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(blockSize*2) + tid;
    unsigned int gridSize = blockSize*2*gridDim.x;
    sdata[tid].x=0.0;   sdata[tid].y=0.0;

    while ( i < n ){
        sdata[tid].x += vecIn[i].x + vecIn[i + blockSize].x;
        sdata[tid].y += vecIn[i].y + vecIn[i + blockSize].y;
        i += gridSize;
    }
    if(blockSize >= 1024) { if(tid < 512) { sdata[tid].x += sdata[tid+512].x; sdata[tid].y += sdata[tid+512].y; } __syncthreads; }
    if(blockSize >= 512)  { if(tid < 256) { sdata[tid].x += sdata[tid+256].x; sdata[tid].y += sdata[tid+256].y; } __syncthreads; }
    if(blockSize >= 256)  { if(tid < 128) { sdata[tid].x += sdata[tid+128].x; sdata[tid].y += sdata[tid+128].y; } __syncthreads; }
    if(blockSize >= 128)  { if(tid < 64)  { sdata[tid].x += sdata[tid+64].x;  sdata[tid].y += sdata[tid+64].y;  } __syncthreads; }

    if (tid < 32){
        if(blockSize >= 64){ sdata[tid].x += sdata[tid+32].x; sdata[tid].y += sdata[tid+32].y; }
        if(blockSize >= 32){ sdata[tid].x += sdata[tid+16].x; sdata[tid].y += sdata[tid+16].y; }
        if(blockSize >= 16){ sdata[tid].x += sdata[tid+8].x;  sdata[tid].y += sdata[tid+8].y; }
        if(blockSize >= 8){  sdata[tid].x += sdata[tid+4].x;  sdata[tid].y += sdata[tid+4].y; }
        if(blockSize >= 4){  sdata[tid].x += sdata[tid+2].x;  sdata[tid].y += sdata[tid+2].y; }
        if(blockSize >= 2){  sdata[tid].x += sdata[tid+1].x;  sdata[tid].y += sdata[tid+1].y; }
    }
    if(tid == 0) vecOut[blockIdx.x] = sdata[0];
}

//
###########################################################################################################//

/*
*  Device functions for dealing with complex numbers.
*/
__host__ __device__ double2 compMagnitude(double2 cmp1);
__host__ __device__ double2 realCompMult(double rl, double2 cmp);
__host__ __device__ double2 compCompMult(double2 cmp1, double2 cmp2);
__host__ __device__ double2 compSum(double2 cmp1, double2 cmp2);
__host__ __device__ double2 conj(double2 cmp);


__host__ __device__ double compMagnitude(double2 cmp1){
    return sqrt(cmp1.x*cmp1.x + cmp1.y*cmp1.y);
}
__host__ __device__ double2 realCompMult(double rl, double2 cmp){
    double2 result;
    result.x = rl*cmp1.x;
    result.y = rl*cmp1.y;
    return result;
}
__host__ __device__ double2 compCompMult(double2 cmp1, double2 cmp2){

}
```

4.61 src/kernels.cu File Reference

```
#include "../include/constants.h"
#include <stdio.h>
```

Include dependency graph for kernels.cu:

Functions

- __device__ unsigned int getGid3d3d ()
- __device__ unsigned int getBid3d3d ()
- __device__ unsigned int getTid3d3d ()
- __device__ double2 conjugate (double2 in)
- __device__ double2 realCompMult (double scalar, double2 comp)
- __device__ double complexMagnitude (double2 in)
- __host__ __device__ double complexMagnitudeSquared (double2 in)
- __host__ __device__ double2 complexMultiply (double2 in1, double2 in2)
- __device__ double2 braKetMult (double2 in1, double2 in2)
- __global__ void cMult (double2 ∗in1, double2 ∗in2, double2 ∗out)

  Performs complex multiplication of in1 and in2, giving result as out.
- __global__ void cMultDensity (double2 ∗in1, double2 ∗in2, double2 ∗out, double dt, double mass, double omegaZ, int gstate, int N)
- __global__ void scalarDiv (double2 ∗in, double factor, double2 ∗out)

  Divides both components of vector type "in", by the value "factor".
- __global__ void scalarDiv_wfcNorm (double2 ∗in, double dr, double2 ∗pSum, double2 ∗out)

  As above, but normalises for wfc.
- __global__ void angularOp (double omega, double dt, double2 ∗wfc, double ∗xpyypx, double2 ∗out)
- __global__ void multipass (double2 ∗input, double2 ∗output, int pass)

  Routine for parallel summation.

- __global__ void energyCalc (double2 ∗wfc, double2 ∗op, double dt, double2 ∗energy, int gnd_state, int op_space, double sqrt_omegaz_mass)
- template<typename T >
  __global__ void pSumT (T ∗in1, T ∗output, int pass)

  Routine for parallel summation.
- __global__ void pSum (double ∗in1, double ∗output, int pass)

  Routine for parallel summation.

Variables

- __constant__ double gDenConst = 2.535425438831619e-59

4.61.1 Function Documentation

4.61.1.1 __global__ void angularOp (double *omega,* double *dt,* double2 ∗ *wfc,* double ∗ *xpyypx,* double2 ∗ *out*)

Definition at line 148 of file kernels.cu.

References getGid3d3d(), and result.

```
00148                                                                                         {
00149     unsigned int gid = getGid3d3d();
00150     double2 result;
00151     double op;
00152     op = exp( -omega*xpyypx[gid]*dt);
00153     result.x=wfc[gid].x*op;
00154     result.y=wfc[gid].y*op;
00155     out[gid]=result;
00156 }
```

Here is the call graph for this function:

4.61.1.2 __device__ double2 braKetMult (double2 *in1,* double2 *in2*) [inline]

Definition at line 83 of file kernels.cu.

References complexMultiply(), and conjugate().

Referenced by energyCalc().

```
00084 {
00085     return complexMultiply(conjugate(in1),in2);
00086 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.61.1.3 __global__ void cMult (double2 ∗ *in1,* double2 ∗ *in2,* double2 ∗ *out*)

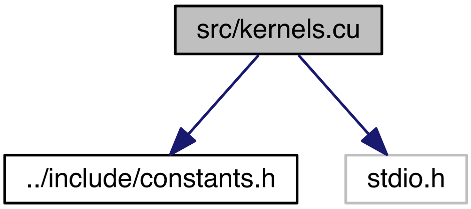
Performs complex multiplication of in1 and in2, giving result as out.

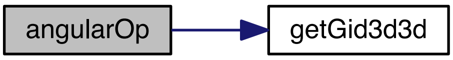
Definition at line 91 of file kernels.cu.

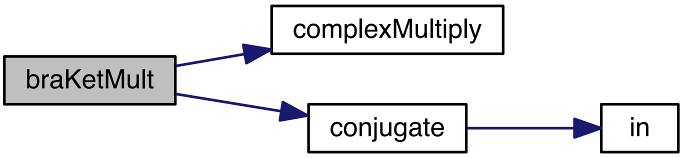
References getGid3d3d(), and result.

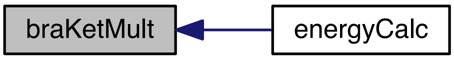
```
00091                                                                   {
00092     double2 result;
00093     unsigned int gid = getGid3d3d();
00094     result.x = (in1[gid].x*in2[gid].x - in1[gid].y*in2[gid].y);
00095     result.y = (in1[gid].x*in2[gid].y + in1[gid].y*in2[gid].x);
00096     out[gid] = result;
00097 }
```

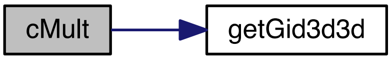
Here is the call graph for this function:


src/kernels.cu
../include/constants.h
stdio.h


angularOp
getGid3d3d


braKetMult
complexMultiply
conjugate
in


braKetMult
energyCalc


cMult
getGid3d3d

4.61.1.4 __global__ void cMultDensity (double2 ∗ *in1*, double2 ∗ *in2*, double2 ∗ *out*, double *dt*, double *mass*, double *omegaZ*, int *gstate*, int *N*)

Definition at line 99 of file kernels.cu.

References complexMagnitudeSquared(), HBAR, mass, observables::N, PI, result, x, and y.

```
00099
                            {
00100     double2 result;
00101     double gDensity;
00102     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00103     gDensity = (0.5*N)*complexMagnitudeSquared(in2[tid])*4*
      HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*
      HBAR));
00104
00105     if(gstate == 0){
00106         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR) );
00107         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00108         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00109     }
00110     else{
00111         double2 tmp;
00112         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(
      dt/HBAR));
00113         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(
      dt/HBAR));
00114
00115         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00116         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00117     }
00118     out[tid] = result;
00119 }
```

Here is the call graph for this function:

4.61.1.5 __device__ double complexMagnitude (double2 *in*)

Definition at line 65 of file kernels.cu.

```
00065                                            {
00066     return sqrt(in.x*in.x + in.y*in.y);
00067 }
```

4.61.1.6 __host__ __device__ double complexMagnitudeSquared (double2 *in*)

Definition at line 69 of file kernels.cu.

Referenced by cMultDensity(), and energyCalc().

```
00069                                                                {
00070     return in.x*in.x + in.y*in.y;
00071 }
```

Here is the caller graph for this function:

4.61.1.7 __host__ __device__ double2 complexMultiply (double2 *in1*, double2 *in2*)

Definition at line 73 of file kernels.cu.

References result.

Referenced by braKetMult().

```
00073                                                                            {
00074     double2 result;
00075     result.x = (in1.x*in2.x - in1.y*in2.y);
00076     result.y = (in1.x*in2.y + in1.y*in2.x);
00077     return result;
00078 }
```

Here is the caller graph for this function:

4.61.1.8 __device__ double2 conjugate (double2 *in*)

Definition at line 51 of file kernels.cu.

References in(), and result.

Referenced by braKetMult().

```
00051                                         {
00052     double2 result = in;
00053     result.y = -result.y;
00054     return result;
00055 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.61.1.9 __global__ void energyCalc (double2 ∗ *wfc,* double2 ∗ *op,* double *dt,* double2 ∗ *energy,* int *gnd_state,* int *op_space,* double *sqrt_omegaz_mass*)

Definition at line 188 of file kernels.cu.

References braKetMult(), complexMagnitudeSquared(), dt, gDenConst, getGid3d3d(), HBAR, realCompMult(), result, and x.

```
00188
                                   {
00189     unsigned int gid = getGid3d3d();
00190     double hbar_dt = HBAR/dt;
00191     double g_local = 0.0;
00192     double2 result;
00193     double opLocal;
00194     if(op_space)
00195         g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(
      wfc[gid]);
00196     if(!gnd_state){
00197         opLocal = -log(op[gid].x + g_local)*hbar_dt;
00198     }
00199     else{
00200         opLocal = cos(op[gid].x + g_local)*hbar_dt;
00201     }
00202     result = braKetMult(wfc[gid], realCompMult(opLocal,
      wfc[gid]));
00203     //printf("oplocal=%e    Resx=%e Resy=%e\n",opLocal,result.x,result.y);
00204     energy[gid].x += result.x;
00205     energy[gid].y += result.y;
00206 }
```

Here is the call graph for this function:

4.61.1.10 __device__ unsigned int getBid3d3d ( )

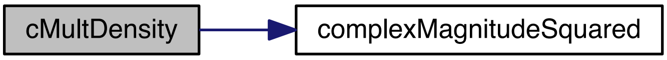
Definition at line 41 of file kernels.cu.

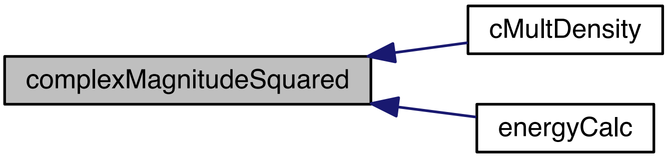
```
00041                                    {
00042     return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00043 }
```

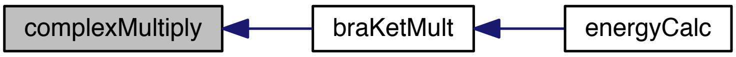
4.61.1.11 __device__ unsigned int getGid3d3d ( )

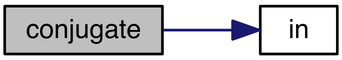
Definition at line 26 of file kernels.cu.

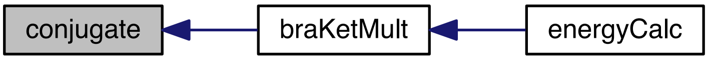
Referenced by angularOp(), cMult(), energyCalc(), multipass(), pSum(), pSumT(), scalarDiv(), and scalarDiv_wfc↩
Norm().


cMultDensity
complexMagnitudeSquared


complexMagnitudeSquared
cMultDensity
energyCalc


complexMultiply
braKetMult
energyCalc


conjugate
in


conjugate
braKetMult
energyCalc

```
00026                                    {
00027     //int idx_x = blockIdx.x * blockDim.x + threadIdx.x;
00028     //int idx_y = blockIdx.y * blockDim.y + threadIdx.y;
00029     //int idx_z = blockIdx.z * blockDim.z + threadIdx.z;
00030 
00031 
00032     //int bidx = blockIdx.x  +  gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00033 
00034     //int gid =  blockDim.x*(idx_z * blockDim.y + idx_y) + idx_x;
00035 //    int gid = blockDim.x * (   blockDim.y* (blockDim.z + ( threadIdx.z * blockDim.y ) )  + threadIdx.y )
       + threadIdx.x;
00036     int gid = blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
      threadIdx.y ) + blockIdx.x ) + threadIdx.x;
00037     return gid;
00038 }
```

Here is the caller graph for this function:

4.61.1.12 __device__ unsigned int getTid3d3d ( )

Definition at line 47 of file kernels.cu.

```
00047                                    {
00048     return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) )  + threadIdx.y )  +
      threadIdx.x;
00049 }
```

4.61.1.13 __global__ void multipass (double2 ∗ *input,* double2 ∗ *output,* int *pass*)

Routine for parallel summation.

Can be looped over from host.

Definition at line 161 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
00161                                                                           {
00162     unsigned int tid = threadIdx.x;
00163     unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
00164     unsigned int gid = getGid3d3d();
00165     extern __shared__ double2 sdata[];
00166     sdata[tid] = input[gid];
00167     if(pass == 0){
00168         sdata[tid].x *= sdata[tid].x;
00169         sdata[tid].y *= sdata[tid].y;
00170     }
00171     __syncthreads();
00172     for(int i = blockDim.x>>1; i > 0; i>>=1){
00173         if(tid < blockDim.x>>1){
00174             sdata[tid].x += sdata[tid + i].x;
00175             sdata[tid].y += sdata[tid + i].y;
00176         }
00177         __syncthreads();
00178     }
00179     if(tid==0){
00180         output[bid] = sdata[0];
00181     }
00182 }
```

Here is the call graph for this function:

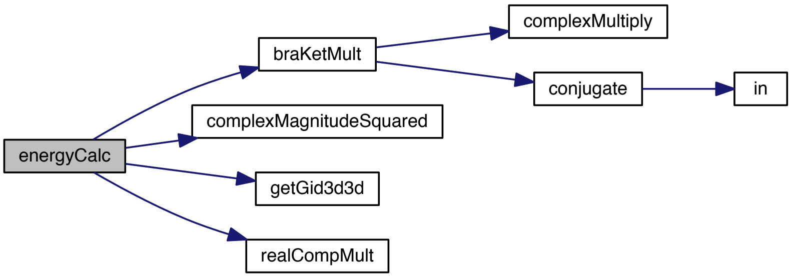
4.61.1.14 __global__ void pSum (double ∗ *in1,* double ∗ *output,* int *pass*)

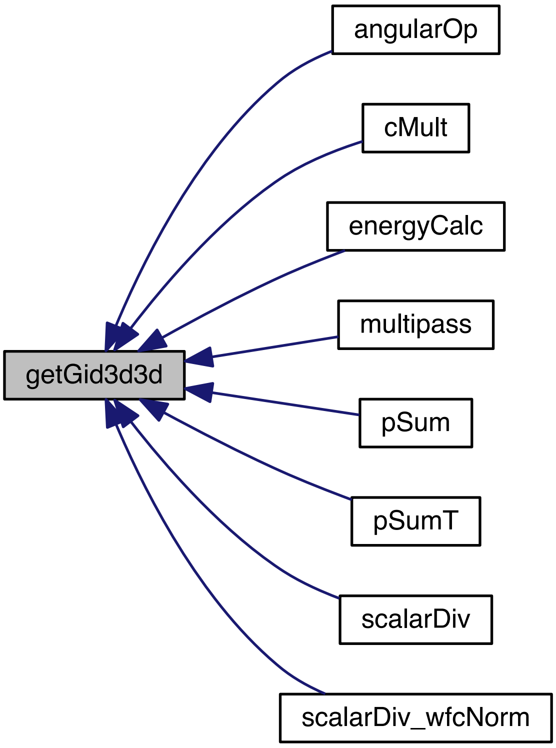
Routine for parallel summation.

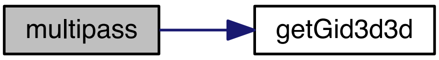
Can be looped over from host.

Definition at line 234 of file kernels.cu.

References getGid3d3d(), and vis::i.


energyCalc
braKetMult
complexMultiply
conjugate
in
complexMagnitudeSquared
getGid3d3d
realCompMult


getGid3d3d
angularOp
cMult
energyCalc
multipass
pSum
pSumT
scalarDiv
scalarDiv_wfcNorm


multipass
getGid3d3d

```
00234                                                           {
00235         unsigned int tid = threadIdx.x;
00236         unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
00237         unsigned int gid = getGid3d3d();
00238         extern __shared__ double sdata2[];
00239         for(int i = blockDim.x>>1; i > 0; i>>=1){
00240                 if(tid < blockDim.x>>1){
00241                         sdata2[tid] += sdata2[tid + i];
00242                 }
00243                 __syncthreads();
00244         }
00245         if(tid==0){
00246                 output[bid] = sdata2[0];
00247         }
00248 }
```

Here is the call graph for this function:

4.61.1.15 template<typename T > __global__ void pSumT (T ∗ *in1,* T ∗ *output,* int *pass*)

Routine for parallel summation.

Can be looped over from host.

Definition at line 215 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
00215                                                                       {
00216         unsigned int tid = threadIdx.x;
00217         unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
00218         unsigned int gid = getGid3d3d();
00219         extern __shared__ T sdata[];
00220         for(int i = blockDim.x>>1; i > 0; i>>=1){
00221                 if(tid < blockDim.x>>1){
00222                         sdata[tid] += sdata[tid + i];
00223                 }
00224                 __syncthreads();
00225         }
00226         if(tid==0){
00227                 output[bid] = sdata[0];
00228         }
00229 }
```

Here is the call graph for this function:

4.61.1.16 __device__ double2 realCompMult (double *scalar,* double2 *comp*)

Definition at line 57 of file kernels.cu.

References result.

Referenced by energyCalc().

```
00057                                                            {
00058     double2 result;
00059     result.x = scalar * comp.x;
00060     result.y = scalar * comp.y;
00061     return result;
00062 }
```

Here is the caller graph for this function:

4.61.1.17 __global__ void scalarDiv (double2 ∗ *in,* double *factor,* double2 ∗ *out*)

Divides both components of vector type "in", by the value "factor".

Results given with "out"

Definition at line 125 of file kernels.cu.

References getGid3d3d(), and result.

```
00125                                                                         {
00126     double2 result;
00127     //extern __shared__ double2 tmp_in[];
00128     unsigned int gid = getGid3d3d();
00129     result.x = (in[gid].x*factor);
00130     result.y = (in[gid].y*factor);
00131     out[gid] = result;
00132 }
```

Here is the call graph for this function:

4.61.1.18 __global__ void scalarDiv_wfcNorm (double2 ∗ *in*, double *dr*, double2 ∗ *pSum*, double2 ∗ *out*)

As above, but normalises for wfc.

Definition at line 137 of file kernels.cu.

References getGid3d3d(), result, x, and y.

```
00137                                                                                         {
00138     unsigned int gid = getGid3d3d();
00139     double2 result;
00140     double norm = sqrt((pSum[0].x + pSum[0].y)*dr);
00141     result.x = (in[gid].x/norm);
00142     result.y = (in[gid].y/norm);
00143     out[gid] = result;
00144 }
```

Here is the call graph for this function:

4.61.2 Variable Documentation

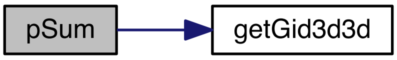
4.61.2.1 __constant__ double gDenConst = 2.535425438831619e-59

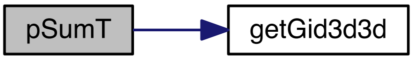
Definition at line 23 of file kernels.cu.

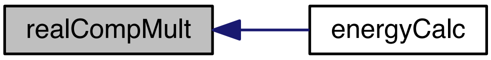
Referenced by energyCalc().

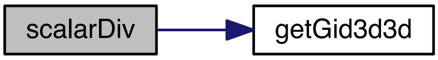
4.62 kernels.cu

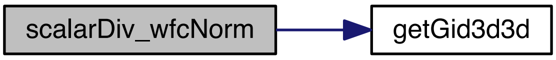
```
00001 /*
00002 * kernels.cu - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
00011 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
00012 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00013 * License for more details. You should have received a copy of the GNU
00014 * Lesser General Public License along with this library; if not, write
00015 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
00016 * Boston, MA 02111-1307 USA
00017 */
00018
00019 #include "../include/constants.h"
00020 #include <stdio.h>
00021
00022
00023 __constant__ double gDenConst = 2.535425438831619e-59;//Evaluted in MATLAB:
       HBAR*(4.67e-9)*sqrt(8*HBAR*PI)*;
00024 //inline __device__ unsigned int getGid3d3d(){
00025
00026 __device__ unsigned int getGid3d3d(){
00027     //int idx_x = blockIdx.x * blockDim.x + threadIdx.x;
00028     //int idx_y = blockIdx.y * blockDim.y + threadIdx.y;
00029     //int idx_z = blockIdx.z * blockDim.z + threadIdx.z;
00030
```


pSum
getGid3d3d


pSumT
getGid3d3d


realCompMult
energyCalc


scalarDiv
getGid3d3d


scalarDiv_wfcNorm
getGid3d3d

```

    //int bidx = blockIdx.x  +  gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);

    //int gid =  blockDim.x*(idx_z * blockDim.y + idx_y) + idx_x;
//    int gid = blockDim.x * (   blockDim.y* (blockDim.z + ( threadIdx.z * blockDim.y ) )  + threadIdx.y ) + threadIdx.x;
    int gid = blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) + threadIdx.y ) + blockIdx.x ) + threadIdx.x;
    return gid;
}

//inline __device__ unsigned int getBid3d3d(){
__device__ unsigned int getBid3d3d(){
    return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
}


//inline __device__ unsigned int getTid3d3d(){
__device__ unsigned int getTid3d3d(){
    return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) )  + threadIdx.y )  + threadIdx.x;
}

__device__ double2 conjugate(double2 in){
    double2 result = in;
    result.y = -result.y;
    return result;
}

__device__ double2 realCompMult(double scalar, double2 comp){
    double2 result;
    result.x = scalar * comp.x;
    result.y = scalar * comp.y;
    return result;
}

//inline __device__ double complexMagnitude(double2 in){
__device__ double complexMagnitude(double2 in){
    return sqrt(in.x*in.x + in.y*in.y);
}

__host__ __device__ double complexMagnitudeSquared(double2 in){
    return in.x*in.x + in.y*in.y;
}

__host__ __device__ double2 complexMultiply(double2 in1, double2 in2){
    double2 result;
    result.x = (in1.x*in2.x - in1.y*in2.y);
    result.y = (in1.x*in2.y + in1.y*in2.x);
    return result;
}

/*
* Used to perform conj(in1)*in2; == < in1 | in2 >
*/
inline __device__ double2 braKetMult(double2 in1, double2 in2)
{
    return complexMultiply(conjugate(in1),in2);
}

__global__ void cMult(double2* in1, double2* in2, double2* out){
    double2 result;
    unsigned int gid = getGid3d3d();
    result.x = (in1[gid].x*in2[gid].x - in1[gid].y*in2[gid].y);
    result.y = (in1[gid].x*in2[gid].y + in1[gid].y*in2[gid].x);
    out[gid] = result;
}

__global__ void cMultDensity(double2* in1, double2* in2, double2* out, double dt, double mass,double omegaZ, int gstate, int N){
    double2 result;
    double gDensity;
    int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
    gDensity = (0.5*N)*complexMagnitudeSquared(in2[tid])*4*HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*HBAR));

    if(gstate == 0){
        double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR) );
        result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
        result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
    }
    else{
        double2 tmp;
        tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(dt/HBAR));
        tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(dt/
HBAR));

        result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
        result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
    }
    out[tid] = result;
}

__global__ void scalarDiv(double2* in, double factor, double2* out){
    double2 result;
    //extern __shared__ double2 tmp_in[];
    unsigned int gid = getGid3d3d();
    result.x = (in[gid].x*factor);
    result.y = (in[gid].y*factor);
    out[gid] = result;
}

__global__ void scalarDiv_wfcNorm(double2* in, double dr, double2* pSum, double2* out){
    unsigned int gid = getGid3d3d();
    double2 result;
    double norm = sqrt((pSum[0].x + pSum[0].y)*dr);
    result.x = (in[gid].x/norm);
    result.y = (in[gid].y/norm);
    out[gid] = result;
}

__global__ void angularOp(double omega, double dt, double2* wfc, double* xpyypx, double2* out){
    unsigned int gid = getGid3d3d();
    double2 result;
    double op;
    op = exp( -omega*xpyypx[gid]*dt);
    result.x=wfc[gid].x*op;
    result.y=wfc[gid].y*op;
    out[gid]=result;
}

__global__ void multipass(double2* input, double2* output, int pass){
    unsigned int tid = threadIdx.x;
    unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
    unsigned int gid = getGid3d3d();
    extern __shared__ double2 sdata[];
    sdata[tid] = input[gid];
    if(pass == 0){
        sdata[tid].x *= sdata[tid].x;
        sdata[tid].y *= sdata[tid].y;
    }
    __syncthreads();
    for(int i = blockDim.x>>1; i > 0; i>>=1){
        if(tid < blockDim.x>>1){
            sdata[tid].x += sdata[tid + i].x;
            sdata[tid].y += sdata[tid + i].y;
        }
        __syncthreads();
    }
    if(tid==0){
        output[bid] = sdata[0];
    }
}


/*
* Calculates all of the energy of the current state. sqrt_omegaz_mass = sqrt(omegaZ/mass), part of the nonlin interaction term
*/
__global__ void energyCalc(double2 *wfc, double2 *op, double dt, double2 *energy, int gnd_state, int op_space, double sqrt_omegaz_mass){
    unsigned int gid = getGid3d3d();
    double hbar_dt = HBAR/dt;
    double g_local = 0.0;
    double2 result;
    double opLocal;
    if(op_space)
        g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(wfc[gid]);
    if(!gnd_state){
        opLocal = -log(op[gid].x + g_local)*hbar_dt;
    }
    else{
        opLocal = cos(op[gid].x + g_local)*hbar_dt;
    }
    result = braKetMult(wfc[gid], realCompMult(opLocal,wfc[gid]));
    //printf("oplocal=%e    Resx=%e Resy=%e\n",opLocal,result.x,result.y);
    energy[gid].x += result.x;
    energy[gid].y += result.y;
}
```

```


//#############################################################################################
//#############################################################################################

template<typename T> __global__ void pSumT(T* in1, T* output, int pass){
        unsigned int tid = threadIdx.x;
        unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
        unsigned int gid = getGid3d3d();
        extern __shared__ T sdata[];
        for(int i = blockDim.x>>1; i > 0; i>>=1){
                if(tid < blockDim.x>>1){
                        sdata[tid] += sdata[tid + i];
                }
                __syncthreads();
        }
        if(tid==0){
                output[bid] = sdata[0];
        }
}

__global__ void pSum(double* in1, double* output, int pass){
        unsigned int tid = threadIdx.x;
        unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
        unsigned int gid = getGid3d3d();
        extern __shared__ double sdata2[];
        for(int i = blockDim.x>>1; i > 0; i>>=1){
                if(tid < blockDim.x>>1){
                        sdata2[tid] += sdata2[tid + i];
                }
                __syncthreads();
        }
        if(tid==0){
                output[bid] = sdata2[0];
        }
}



//#############################################################################################
//#############################################################################################
```

4.63 src/minions.cc File Reference

```
#include "../include/minions.h"
```

Include dependency graph for minions.cc:

Functions

- double psi2 (double2 in)
- double maxValue (double *grid, int len)
- double minValue (double *grid, int len)
- double sumAvg (double *in, int len)
- double fInvSqRt (double in)

  id magic hackery
- void coordSwap (struct Vortex *vCoords, int src, int dest)
- double complexMag (double2 in)
- double complexMag2 (double2 in)
- double2 complexMult (double2 in1, double2 in2)
- double2 complexScale (double2 comp, double scale)
- double2 conj (double2 c)
- double2 complexDiv (double2 num, double2 den)

4.63.1 Function Documentation

4.63.1.1 double2 complexDiv (double2 *num,* double2 *den*)

Definition at line 101 of file minions.cc.

References hist3d::c, complexMag2(), complexMult(), complexScale(), and conj().

Referenced by findVortex(), and phaseTest().

```
00101                                             {
00102     double2 c = conj(den);
00103     return complexScale(complexMult(num,c),(1.0/
      complexMag2(den)));
00104 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.63.1.2 double complexMag (double2 *in*)

Definition at line 73 of file minions.cc.

Referenced by findVortex(), and phaseTest().

```
00073                              {
00074     return sqrt(in.x*in.x + in.y*in.y);
00075 }
```

Here is the caller graph for this function:

4.63.1.3 double complexMag2 (double2 *in*)

Definition at line 77 of file minions.cc.

Referenced by complexDiv().

```
00077                               {
00078     return in.x*in.x + in.y*in.y;
00079 }
```

Here is the caller graph for this function:

4.63.1.4 double2 complexMult (double2 *in1,* double2 *in2*)

Definition at line 81 of file minions.cc.

References result.

Referenced by complexDiv(), and main().

```
00081                                              {
00082     double2 result;
00083     result.x = (in1.x*in2.x - in1.y*in2.y);
00084     result.y = (in1.x*in2.y + in1.y*in2.x);
00085     return result;
00086 }
```

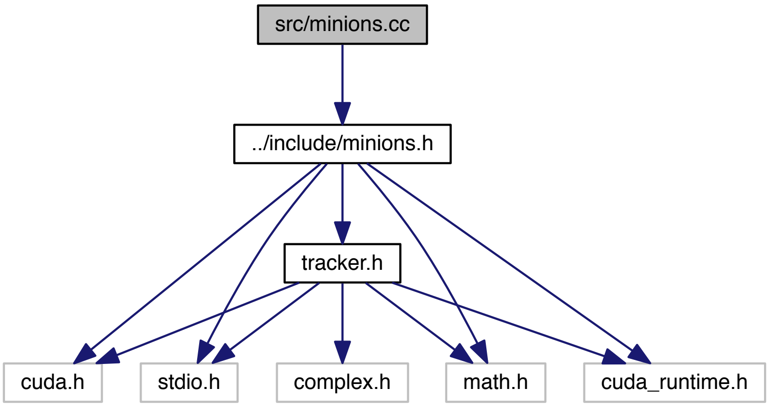
Here is the caller graph for this function:

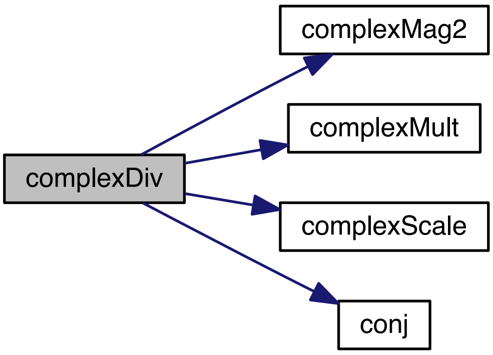
4.63.1.5 double2 complexScale (double2 *comp,* double *scale*)

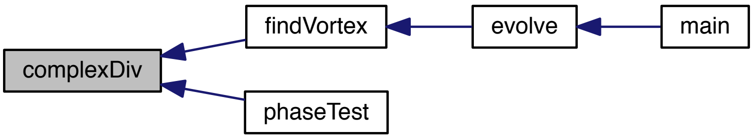
Definition at line 88 of file minions.cc.

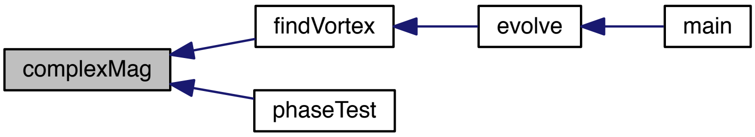
References result.

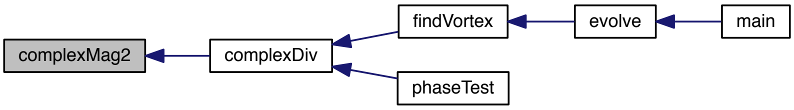
Referenced by complexDiv(), findVortex(), and phaseTest().

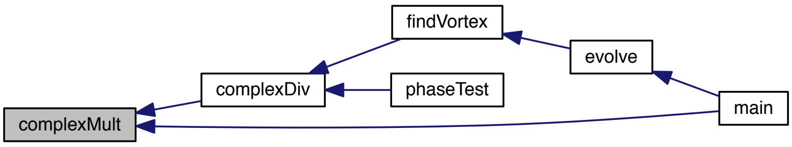

src/minions.cc
../include/minions.h
tracker.h
cuda.h
stdio.h
complex.h
math.h
cuda_runtime.h


complexMag2
complexMult
complexDiv
complexScale
conj


findVortex
evolve
main
complexDiv
phaseTest


complexMag
findVortex
evolve
main
phaseTest


complexMag2
complexDiv
findVortex
evolve
main
phaseTest


findVortex
evolve
complexDiv
phaseTest
main
complexMult

```
00088                                                  {
00089     double2 result;
00090     result.x = comp.x*scale;
00091     result.y = comp.y*scale;
00092     return result;
00093 }
```

Here is the caller graph for this function:

4.63.1.6 double2 conj (double2 *c*)

Definition at line 95 of file minions.cc.

References hist3d::c, and result.

Referenced by complexDiv().

```
00095                        {
00096     double2 result = c;
00097     result.y = -result.y;
00098     return result;
00099 }
```

Here is the caller graph for this function:

4.63.1.7 void coordSwap (struct Vortex ∗ *vCoords,* int *src,* int *dest*)

Definition at line 67 of file minions.cc.

Referenced by main(), and vortArrange().

```
00067                                                           {
00068     struct Vortex d = vCoords[dest];
00069     vCoords[dest] = vCoords[src];
00070     vCoords[src] = d;
00071 }
```

Here is the caller graph for this function:

4.63.1.8 double fInvSqRt (double *in*)

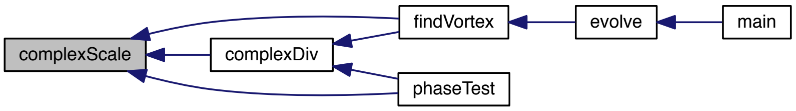
id magic hackery

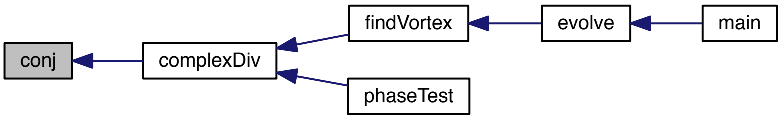
Definition at line 52 of file minions.cc.

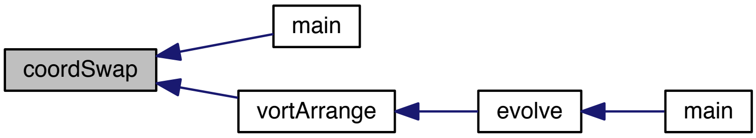
References in(), and l.

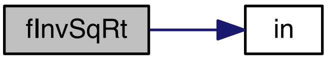
```
00052                           {
00053     long long l;
00054     double in05, calc;
00055     const double threehalfs = 1.5;
00056
00057     in05 = in*0.5;
00058     calc=in;
00059     l = * (long long*) &calc;
00060     l = 0x5fe6eb50c7b537a9LL - (l >> 1);
00061     calc = *(double *) &l;
00062     calc = calc*( 1.5 - (in05*calc*calc) );
00063
00064     return calc;
00065 }
```

Here is the call graph for this function:


complexScale
complexDiv
findVortex
phaseTest
evolve
main


conj
complexDiv
findVortex
phaseTest
evolve
main


coordSwap
main
vortArrange
evolve
main


fInvSqRt
in

4.63.1.9 double maxValue (double ∗ *grid,* int *len*)

Definition at line 25 of file minions.cc.

References vis::i.

Referenced by findOLMaxima().

```
00025                                    {
00026     double max = grid[0];
00027     for (unsigned int i=1;i<len-1;++i){
00028         if(max<grid[i])
00029             max=grid[i];
00030     }
00031     return max;
00032 }
```

Here is the caller graph for this function:

4.63.1.10 double minValue (double ∗ *grid,* int *len*)

Definition at line 34 of file minions.cc.

References vis::i.

Referenced by vortAngle().

```
00034                                    {
00035     double min = grid[0];
00036     for (unsigned int i=1;i<len-1;++i){
00037         if(min>grid[i])
00038             min=grid[i];
00039     }
00040     return min;
00041 }
```

Here is the caller graph for this function:

4.63.1.11 double psi2 (double2 *in*)

Definition at line 21 of file minions.cc.

Referenced by evolve().

```
00021                       {
00022     return in.x*in.x + in.y*in.y;
00023 }
```

Here is the caller graph for this function:

4.63.1.12 double sumAvg (double ∗ *in,* int *len*)

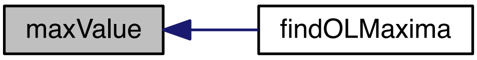
Definition at line 43 of file minions.cc.

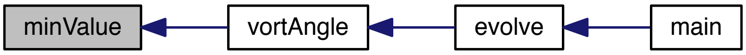
References vis::i.

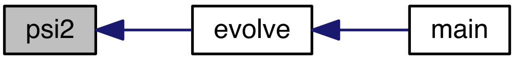
Referenced by evolve().

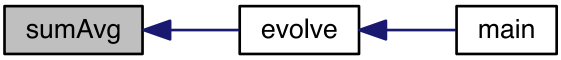
```
00043                                 {
00044     double avg = 0.0;
00045 
00046     for (unsigned int i=0; i<len; ++i){
00047         avg += in[i];
00048     }
00049     return avg/len;
00050 }
```

Here is the caller graph for this function:


maxValue
findOLMaxima


minValue
vortAngle
evolve
main


psi2
evolve
main


sumAvg
evolve
main

4.64 minions.cc

```
/*
* minions.cc - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#include "../include/minions.h"

double psi2(double2 in){
    return in.x*in.x + in.y*in.y;
}

double maxValue(double* grid,int len){
    double max = grid[0];
    for (unsigned int i=1;i<len-1;++i){
        if(max<grid[i])
            max=grid[i];
    }
    return max;
}

double minValue(double* grid,int len){
    double min = grid[0];
    for (unsigned int i=1;i<len-1;++i){
        if(min>grid[i])
            min=grid[i];
    }
    return min;
}

double sumAvg(double* in, int len){
    double avg = 0.0;

    for (unsigned int i=0; i<len; ++i){
        avg += in[i];
    }
    return avg/len;
}

double fInvSqRt(double in){
    long long l;
    double in05, calc;
    const double threehalfs = 1.5;

    in05 = in*0.5;
    calc=in;
    l = * (long long*) &calc;
    l = 0x5fe6eb50c7b537a9LL - (l >> 1);
    calc = *(double *) &l;
    calc = calc*( 1.5 - (in05*calc*calc) );

    return calc;
}

void coordSwap(struct Vortex *vCoords, int src, int dest){
    struct Vortex d = vCoords[dest];
    vCoords[dest] = vCoords[src];
    vCoords[src] = d;
}

double complexMag(double2 in){
    return sqrt(in.x*in.x + in.y*in.y);
}

double complexMag2(double2 in){
    return in.x*in.x + in.y*in.y;
}

double2 complexMult(double2 in1, double2 in2){
    double2 result;
    result.x = (in1.x*in2.x - in1.y*in2.y);
    result.y = (in1.x*in2.y + in1.y*in2.x);
    return result;
}

double2 complexScale(double2 comp, double scale){
    double2 result;
    result.x = comp.x*scale;
    result.y = comp.y*scale;
    return result;
}

double2 conj(double2 c){
    double2 result = c;
    result.y = -result.y;
    return result;
}

double2 complexDiv(double2 num, double2 den){
    double2 c = conj(den);
    return complexScale(complexMult(num,c),(1.0/complexMag2(den)));
}

/*
int qSort(int2 *vCoords, int *vCoordsP int index, int length){
    if(index < 2){
        return 0;
    }
    int2 pivot;
    int l = 0;
    int r = length - 1;
    while (l <= r){
        0;
    }
}
*/
```

4.65 src/multigpu.cu File Reference

4.66 multigpu.cu

4.67 src/split_op.cu File Reference

```
#include "../include/split_op.h"
#include "../include/kernels.h"
#include "../include/constants.h"
#include "../include/fileIO.h"
#include "../include/tracker.h"
#include "../include/minions.h"
#include "../include/ds.h"
```

Include dependency graph for split_op.cu:

Functions

- int isError (int result, char *c)
- int initialise (double omegaX, double omegaY, int N)
- int evolve (cufftDoubleComplex *gpuWfc, cufftDoubleComplex *gpuMomentumOp, cufftDoubleComplex *gpuPositionOp, void *gpu1dyPx, void *gpu1dxPy, cufftDoubleComplex *gpuParSum, int gridSize, int numSteps, int threads, unsigned int gstate, int lz, int nonlin, int printSteps, int N, unsigned int ramp)
- void parSum (double2 *gpuWfc, double2 *gpuParSum, int xDim, int yDim, int threads)
- void optLatSetup (struct Vortex centre, double *V, struct Vortex *vArray, int num_vortices, double theta_opt, double intensity, double *v_opt, double *x, double *y)

  Matches the optical lattice to the vortex lattice.

- double energy_angmom (double ∗Energy, double ∗Energy_gpu, double2 ∗V_op, double2 ∗K_op, double dx, double dy, double2 ∗gpuWfc, int gState)

  Calculates energy and angular momentum of current state.
- template<typename T >
  void parSum (T ∗gpuToSumArr, T ∗gpuParSum, int xDim, int yDim, int threads)
- int parseArgs (int argc, char ∗∗argv)
- void delta_define (double ∗x, double ∗y, double x0, double y0, double ∗delta)
- int main (int argc, char ∗∗argv)

Variables

- char buffer [100]
- int verbose
- int device
- int kick_it
- double gammaY
- double omega
- double timeTotal
- double angle_sweep
- Params ∗ paramS
- Array params
- double x0_shift
- double y0_shift
- double Rxy
- double a0x
- double a0y

4.67.1 Function Documentation

4.67.1.1 void delta_define (double ∗ *x,* double ∗ *y,* double *x0,* double *y0,* double ∗ *delta*)

Definition at line 833 of file split_op.cu.

References dt, dx, EV_opt, HBAR, vis::i, V, xDim, and yDim.

Referenced by main().

```
00833                                                                              {
00834     for (unsigned int i=0; i<xDim; ++i){
00835         for (unsigned int j=0; j<yDim; ++j){
00836             delta[j*xDim + i] = 1e6*HBAR*exp( -( pow( x[i] - x0, 2)  +  pow(
      y[j] - y0, 2) )/(5*dx*dx) );
00837             EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + delta[j*xDim +
      i])*(dt/(2*HBAR)));
00838             EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + delta[j*xDim +
      i])*(dt/(2*HBAR)));
00839         }
00840     }
00841 
00842 }
```

Here is the caller graph for this function:

4.67.1.2 double energy_angmom (double ∗ *Energy,* double ∗ *Energy_gpu,* double2 ∗ *V_op,* double2 ∗ *K_op,* double *dx,* double *dy,* double2 ∗ *gpuWfc,* int *gState*)

Calculates energy and angular momentum of current state.

Definition at line 628 of file split_op.cu.

References vis::i, result, xDim, and yDim.

```

                                          {
    double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
    double result=0;

    for (int i=0; i < xDim*yDim; ++i){
        Energy[i] = 0.0;
    }


/*  cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);

    energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
    result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
    scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise

    energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
    result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
    scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise

    err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);

    for(int i=0; i<xDim*yDim; i++){
        result += energy[i].x;
        //printf("En=%E\n",result*dx*dy);
    }
    return result*dx*dy;
*/

}
```

4.67.1.3 int evolve (cufftDoubleComplex ∗ *gpuWfc,* cufftDoubleComplex ∗ *gpuMomentumOp,* cufftDoubleComplex ∗ *gpuPositionOp,* void ∗ *gpu1dyPx,* void ∗ *gpu1dxPy,* cufftDoubleComplex ∗ *gpuParSum,* int *gridSize,* int *numSteps,* int *threads,* unsigned int *gstate,* int *lz,* int *nonlin,* int *printSteps,* int *N,* unsigned int *ramp*)

∗∗ ##

∗∗

∗∗ HERE BE DRAGONS OF THE MOST DANGEROUS KIND! ∗∗

∗∗ ##

∗∗

∗∗ ##

∗∗

∗∗ ##

∗∗

∗∗ More F'n' Dragons! ∗∗

∗∗ ##

∗∗

∗∗ ##

∗∗

Definition at line 295 of file split_op.cu.

References a_s, angle_sweep, appendData(), buffer, Vortex::coords, dt, dx, dy, EV, EV_opt, findVortex(), gdt, vis::i, interaction, kick_it, mass, omega, omegaX, omegaZ, parSum(), PI, plan_1d, plan_2d, psi2(), result, sepAvg(), Vortex::sign, sumAvg(), V_gpu, V_opt, vortAngle(), vortArrange(), vortCentre(), vortPos(), vortSepAvg(), wfc, Vortex::wind, write_it, writeOut(), writeOutDouble(), writeOutParam(), writeOutVortex(), x, xDim, xi, and yDim.

Referenced by main().

```
                                                                                  {

    //Because no two operations are created equally. Multiplimultiplication is faster than divisions.
    double renorm_factor_2d=1.0/pow(gridSize,0.5);
    double renorm_factor_1d=1.0/pow(xDim,0.5);

    clock_t begin, end;
    double time_spent;
    double Dt;
    if(gstate==0){
        Dt = gdt;
        printf("Timestep for grounstate solver set as: %E\n",Dt);
    }
    else{
        Dt = dt;
        printf("Timestep for evolution set as: %E\n",Dt);
    }
    begin = clock();
    double omega_0=omega*omegaX;

    #if 0

    int gridSum = 1<<6;
    double *densitySubset = (double*) malloc(sizeof(double)*gridSum);
    #pragma omp parallel for private(k)
    for (int j=0; j<gridSum; ++j){
        for (int k=0; k<gridSum; ++k){
            densitySubset[j*gridSum + k] = psi2(wfc[ ( (yDim/2) - (gridSum/2) + j )*yDim  + ( (xDim/2)  - (gridSum/2) + k )]);
        }
    }
    xi = 1/sqrt(8*PI*a_s*sumAvg(densitySubset,gridSum)/(dx*dy));//defined central condensate density
    printf("Avg healing length at centre=%E\n",xi);
    #endif

    //Double buffering and will attempt to thread free and calloc operations to hide time penalty. Or may not bother.
    int num_vortices[2] = {0,0};
    int num_latt_max = 0;
    int* vortexLocation; //binary matrix of size xDim*yDim, 1 for vortex at specified index, 0 otherwise
    int* olMaxLocation = (int*) calloc(xDim*yDim,sizeof(int));

    struct Vortex central_vortex; //vortex closest to the central position
    double vort_angle; //Angle of vortex lattice. Add to optical lattice for alignment.
    struct Vortex *vortCoords = NULL; //array of vortex coordinates from vortexLocation 1's
    struct Vortex *vortCoordsP = NULL; //Previous array of vortex coordinates from vortexLocation 1's
    int2 *olCoords = NULL; //array of vortex coordinates from vortexLocation 1's
    int2 *vortDelta = NULL;

    double vortOLSigma=0.0;
    double sepAvg = 0.0;

    int num_kick = 0;
    double t_kick = (2*PI/omega_0)/(6*Dt);

    for(int i=0; i < numSteps; ++i){
        if ( ramp == 1 ){
            omega_0=omegaX*((omega-0.39)*((double)i/(double)(numSteps)) + 0.39); //Adjusts omega for the appropriate trap frequency.
        }
        if(i % printSteps == 0){
            printf("Step: %d    Omega: %lf\n",i,omega_0/omegaX);
            cudaMemcpy(wfc, gpuWfc, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
            end = clock();
            time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
            printf("Time spent: %lf\n",time_spent);
            char* fileName = "";
            printf("ramp=%d     gstate=%d   rg=%d       \n",ramp,gstate,ramp | (gstate<<1));
            switch ( ramp | (gstate<<1) ){
                case 0:
                    fileName = "wfc_0_const";
                    break;
                case 1:
                    fileName = "wfc_0_ramp";
                    break;
                case 2:
                    fileName = "wfc_ev";
                    vortexLocation = (int*) calloc(xDim*yDim,sizeof(int));
                    num_vortices[0] = findVortex(vortexLocation, wfc, 1e-4, xDim, x, i);
                    if(i==0){
                        vortCoords = (struct Vortex*) malloc(sizeof(struct Vortex)*(2*num_vortices[0]));
                        vortCoordsP = (struct Vortex*) malloc(sizeof(struct Vortex)*(2*num_vortices[0]));
                        vortPos(vortexLocation, vortCoords, xDim, wfc);
                        central_vortex = vortCentre(vortCoords, num_vortices[0], xDim);
                        //if(angle_sweep==0.0)
                            vort_angle = vortAngle(vortCoords,central_vortex, num_vortices[0] + PI*angle_sweep/180.0);
                        //else
                        //  vort_angle = angle_sweep;
                        appendData(&params,"Vort_angle",vort_angle);
                        //optLatSetup(central_vortex, V, vortCoords, num_vortices[0], vort_angle,
 laser_power*HBAR*sqrt(omegaX*omegaY), V_opt, x, y);
                        sepAvg = vortSepAvg(vortCoords,central_vortex,num_vortices[0]);
                        if(kick_it == 2){
                            printf("Kicked it 1\n");
                            cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
                        }
                        writeOutDouble(buffer,"V_opt_1",V_opt,xDim*yDim,0);
                        writeOut(buffer,"EV_opt_1",EV_opt,xDim*yDim,0);
                        appendData(&params,"Central_vort_x",(double)central_vortex.coords.x);
                        appendData(&params,"Central_vort_y",(double)central_vortex.coords.y);
                        appendData(&params,"Central_vort_winding",(double)central_vortex.wind);
                        appendData(&params,"Central_vort_sign",(double)central_vortex.sign);
                        appendData(&params,"Num_vort",(double)num_vortices[0]);
                        writeOutParam(buffer, params, "Params.dat");
                    }
                    else if(num_vortices[0] > num_vortices[1]){
                        printf("Number of vortices changed from %d to %d\n",num_vortices[1],num_vortices[0]);
                        vortPos(vortexLocation, vortCoords, xDim,wfc);
                    }
                    else{
                        vortPos(vortexLocation, vortCoords, xDim,wfc);
                        vortArrange(vortCoords, vortCoordsP, num_vortices[0]);
                    }
            /*      num_latt_max = findOLMaxima(olMaxLocation, V_opt, 1e-4, xDim, x);
                    if(num_latt_max == num_vortices[0]){
                        olCoords = (int2*) malloc(sizeof(int2)*num_latt_max);
                        olPos(olMaxLocation, olCoords, xDim);
                        vortOLSigma = sigVOL(vortCoords, olCoords, x, num_latt_max);
                        writeOutInt2(buffer, "opt_max_arr", olCoords, num_latt_max, i);
                        free(olCoords);
                    }*/
                    writeOutVortex(buffer, "vort_arr", vortCoords, num_vortices[0],i);
                    printf("Located %d vortices\n",num_vortices[0]);
                    printf("Sigma=%e\n",vortOLSigma);
                    free(vortexLocation);
                    num_vortices[1] = num_vortices[0];
                    memcpy(vortCoordsP,vortCoords,sizeof(int2)*num_vortices[0]);
                    break;
                case 3:
                    fileName = "wfc_ev_ramp";
                    break;
                default:
                    break;
            }
            if(write_it)
                writeOut(buffer, fileName, wfc, xDim*yDim,i);
            //printf("Energy[t@%d]=%E\n",i,energy_angmom(gpuPositionOp, gpuMomentumOp, dx, dy, gpuWfc,gstate));
/*          cudaMemcpy(V_gpu, V, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
            cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
            cudaMemcpy(V_gpu, , sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
            cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
*/      }

        if(i % (int) t_kick+1 == 0 && num_kick<=6 && gstate==1 && kick_it == 1 ){
            cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
            ++num_kick;
        }
        /*
         * U_r(dt/2)*wfc
         */
        if(nonlin == 1){
            cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,0.5*Dt,mass,omegaZ,gstate,N*interaction);
        }
        else {
            cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
        }

        /*
         * U_p(dt)*fft2(wfc)
         */
        result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD);
        scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc); //Normalise
        cMult<<<grid,threads>>>(gpuMomentumOp,gpuWfc,gpuWfc);
        result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE);
        scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc); //Normalise

        /*
         * U_r(dt/2)*wfc
         */
        if(nonlin == 1){
            cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,Dt*0.5,mass,omegaZ,gstate,N*interaction);
        }
        else {
            cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
        }

        if( (i % (int)t_kick+1 ==0 && num_kick<=6 && gstate==1) || (kick_it >= 1 && i==0) ){
            cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
            printf("Got here\n");
        }
        /**************************************************************/
        /* Angular momentum xPy-yPx   */
        if(lz == 1){
            switch(i%2 | (gstate<<1)){
                case 0: //Groundstate solver, even step
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dxPy, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);

                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dyPx, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                    break;

                case 1: //Groundstate solver, odd step
                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dyPx, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);

                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dxPy, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    break;

                case 2: //Real time evolution, even step
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dxPy, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);

                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dyPx, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                    break;

                case 3: //Real time evolution, odd step
                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dyPx, gpuWfc);
                    result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                    result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                    scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);

```

```
00557                 result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
00558                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
00559                 cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dxPy, gpuWfc);
00560                 result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
00561                 scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
00562                 break;
00563 
00564             }
00565         }
00566         /**************************************************************/
00567 
00568         if(gstate==0){
00569             parSum(gpuWfc, gpuParSum, xDim, yDim, threads);
00570         }
00571     }
00572     return 0;
00573 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.67.1.4 int initialise (double *omegaX,* double *omegaY,* int *N*)

Definition at line 48 of file split_op.cu.

References a0x, a0y, a_s, appendData(), vis_ev::b, buffer, dt, dx, dy, EappliedField, EK, Energy, Energy_gpu, EV, EV_opt, ExPy, EyPx, gammaY, gdt, GK, grid, GV, HBAR, vis::i, K, K_gpu, l, mass, omega, omegaZ, par_sum, Phi, PI, plan_1d, plan_2d, r, result, Rxy, threads, V, V_gpu, V_opt, wfc, wfc_backup, wfc_gpu, writeOut(), writeOutDouble(), x, xDim, xMax, xp, xPy, xPy_gpu, y, yDim, yMax, yp, yPx, and yPx_gpu.

Referenced by main().

```
00048                                                    {
00049     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00050     unsigned int xD=1,yD=1,zD=1;
00051     threads = 128;
00052     unsigned int b = xDim*yDim/threads;  //number of blocks in simulation
00053     unsigned long long maxElements = 65536*65536ULL; //largest number of elements
00054 
00055     if( b < (1<<16) ){
00056         xD = b;
00057     }
00058     else if( (b >= (1<<16) ) && (b <= (maxElements)) ){
00059         int t1 = log(b)/log(2);
00060         float t2 = (float) t1/2;
00061         t1 = (int) t2;
00062         if(t2 > (float) t1){
00063             xD <<= t1;
00064             yD <<= (t1 + 1);
00065         }
00066         else if(t2 == (float) t1){
00067             xD <<= t1;
00068             yD <<= t1;
00069         }
00070     }
00071     else{
00072         printf("Outside range of supported indexing");
00073         exit(-1);
00074     }
00075     printf("Compute grid dimensions chosen as X=%d  Y=%d\n",xD,yD);
00076 
00077     grid.x=xD;
00078     grid.y=yD;
00079     grid.z=zD;
00080     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00081 
00082     unsigned int i,j; //Used in for-loops for indexing
00083 
00084     unsigned int gSize = xDim*yDim;
00085     double xOffset, yOffset;
00086     xOffset=0.0;//5.0e-6;
00087     yOffset=0.0;//5.0e-6;
00088 
00089     mass = 1.4431607e-25; //Rb 87 mass, kg
00090     appendData(&params,"Mass",mass);
00091     a_s = 4.67e-9;
00092     appendData(&params,"a_s",a_s);
00093 
00094     double sum = 0.0;
```

```
    a0x = sqrt(HBAR/(2*mass*omegaX));
    a0y = sqrt(HBAR/(2*mass*omegaY));
    appendData(&params,"a0x",a0x);
    appendData(&params,"a0y",a0y);

    Rxy = pow(15,0.2)*pow(N*a_s*sqrt(mass*omegaZ/HBAR),0.2);
    appendData(&params,"Rxy",Rxy);
    //Rxy = pow(15,0.2)*pow(N*4.67e-9*sqrt(mass*pow(omegaX*omegaY,0.5)/HBAR),0.2);
    double bec_length = sqrt( HBAR/mass*sqrt( omegaX*omegaX * ( 1 - omega*omega) ) );
    xMax = 6*Rxy*a0x;//10*bec_length;//6*Rxy*a0x;
    yMax = 6*Rxy*a0y;//10*bec_length;//
    appendData(&params,"xMax",xMax);
    appendData(&params,"yMax",yMax);

    double pxMax, pyMax;
    pxMax = (PI/xMax)*(xDim>>1);
    pyMax = (PI/yMax)*(yDim>>1);
    appendData(&params,"pyMax",pyMax);
    appendData(&params,"pxMax",pxMax);

    dx = xMax/(xDim>>1);
    dy = yMax/(yDim>>1);
    appendData(&params,"dx",dx);
    appendData(&params,"dy",dy);

    double dpx, dpy;
    dpx = PI/(xMax);
    dpy = PI/(yMax);
    appendData(&params,"dpx",dpx);
    appendData(&params,"dpy",dpy);

    //printf("a0x=%e  a0y=%e \n dx=%e   dx=%e\n R_xy=%e\n",a0x,a0y,dx,dy,Rxy);
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    //double *x,*y,*xp,*yp;
    x = (double *) malloc(sizeof(double) * xDim);
    y = (double *) malloc(sizeof(double) * yDim);
    xp = (double *) malloc(sizeof(double) * xDim);
    yp = (double *) malloc(sizeof(double) * yDim);

    /*
     * Pos and Mom grids
     */
    for(i=0; i<xDim/2; ++i){
        x[i] = -xMax + (i+1)*dx;
        x[i + (xDim/2)] = (i+1)*dx;

        y[i] = -yMax + (i+1)*dy;
        y[i + (yDim/2)] = (i+1)*dy;

        xp[i] = (i+1)*dpx;
        xp[i + (xDim/2)] = -pxMax + (i+1)*dpx;

        yp[i] = (i+1)*dpy;
        yp[i + (yDim/2)] = -pyMax + (i+1)*dpy;
    }

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    /* Initialise wavefunction, momentum and position operators on host */
    Energy = (double*) malloc(sizeof(double) * gSize);
    r = (double *) malloc(sizeof(double) * gSize);
    Phi = (double *) malloc(sizeof(double) * gSize);
    wfc = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    wfc_backup = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * (gSize/threads));
    K = (double *) malloc(sizeof(double) * gSize);
    V = (double *) malloc(sizeof(double) * gSize);
    V_opt = (double *) malloc(sizeof(double) * gSize);
    GK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    GV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EV_opt = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    xPy = (double *) malloc(sizeof(double) * gSize);
    yPx = (double *) malloc(sizeof(double) * gSize);
//  GxPy = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
//  GyPx = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    ExPy = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EyPx = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EappliedField = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);

    /* Initialise wfc, EKp, and EVr buffers on GPU */
    cudaMalloc((void**) &Energy_gpu, sizeof(double) * gSize);
    cudaMalloc((void**) &wfc_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &K_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &V_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &xPy_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &yPx_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &par_sum, sizeof(cufftDoubleComplex) * (gSize/threads));
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    #ifdef __linux
    int cores = omp_get_num_procs();
    appendData(&params,"Cores_Total",cores);
    appendData(&params,"Cores_Max",cores/2);
    omp_set_num_threads(cores/2);
    #pragma omp parallel for private(j)
    #endif
    for( i=0; i < xDim; i++ ){
        for( j=0; j < yDim; j++ ){
            //Remember, you are going from -PI to +PI, not 0 to 2PI
            if(x[i]>=0){
                Phi[(i*yDim + j)] = atan((y[j] + dx/10)/(x[i]) ) - PI/2.0;
            }
            else
                Phi[(i*yDim + j)] = atan((y[j] + dx/10)/(x[i]) ) + PI/2.0;

            Phi[(i*yDim + j)] = fmod(l*Phi[(i*xDim + j)],2*PI);

            wfc[(i*yDim + j)].x = exp(-( pow((x[i])/(Rxy*a0x),2) + pow((y[j])/(Rxy*a0y),2) ) )*cos(Phi[(i*xDim + j)]);
            wfc[(i*yDim + j)].y = -exp(-( pow((x[i])/(Rxy*a0x),2) + pow((y[j])/(Rxy*a0y),2) ) )*sin(Phi[(i*xDim + j)]);

            V[(i*yDim + j)] = 0.5*mass*( pow(omegaX*(x[i]+xOffset),2) + pow(gammaY*omegaY*(y[j]+yOffset),2) );
            K[(i*yDim + j)] = (HBAR*HBAR/(2*mass))*(xp[i]*xp[i] + yp[j]*yp[j]);

            //V_opt[i*yDim + j] = cos(sin(2*PI/3)*(x[i]/(xMax*0.01)) + cos(2*PI/3)*(y[j]/(yMax*0.01)))
            //        + cos(sin(4*PI/3)*(x[i]/(xMax*0.01)) + cos(4*PI/3)*(y[j]/(yMax*0.01)))
            //        + cos(sin(2*PI)*(x[i]/(xMax*0.01)) + cos(2*PI)*(y[j]/(yMax*0.01)));

            GV[(i*yDim + j)].x = exp( -V[(i*xDim + j)]*(gdt/(2*HBAR)));
            GK[(i*yDim + j)].x = exp( -K[(i*xDim + j)]*(gdt/HBAR));
            GV[(i*yDim + j)].y = 0.0;
            GK[(i*yDim + j)].y = 0.0;

            xPy[(i*yDim + j)] = x[i]*yp[j];
            yPx[(i*yDim + j)] = -y[j]*xp[i];

//          GxPy[(i*yDim + j)].x = exp( -omega*xPy[(i*yDim + j)]*gdt);
//          GxPy[(i*yDim + j)].y = 0.0;
//          GyPx[(i*yDim + j)].x = exp( -omega*yPx[(i*yDim + j)]*gdt);
//          GyPx[(i*yDim + j)].y = 0.0;

            EV[(i*yDim + j)].x=cos( -V[(i*xDim + j)]*(dt/(2*HBAR)));
            EV[(i*yDim + j)].y=sin( -V[(i*xDim + j)]*(dt/(2*HBAR)));
            EK[(i*yDim + j)].x=cos( -K[(i*xDim + j)]*(dt/HBAR));
            EK[(i*yDim + j)].y=sin( -K[(i*xDim + j)]*(dt/HBAR));

            ExPy[(i*yDim + j)].x=cos(-omega*omegaX*xPy[(i*xDim + j)]*dt);
            ExPy[(i*yDim + j)].y=sin(-omega*omegaX*xPy[(i*xDim + j)]*dt);
            EyPx[(i*yDim + j)].x=cos(-omega*omegaX*yPx[(i*xDim + j)]*dt);
            EyPx[(i*yDim + j)].y=sin(-omega*omegaX*yPx[(i*xDim + j)]*dt);

            sum+=sqrt(wfc[(i*xDim + j)].x*wfc[(i*xDim + j)].x + wfc[(i*xDim + j)].y*wfc[(i*xDim + j)].y);
        }
    }
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
    //hdfWriteDouble(xDim, V, 0, "V_0");
    //hdfWriteComplex(xDim, wfc, 0, "wfc_0");
    writeOutDouble(buffer,"V",V,xDim*yDim,0);
    //writeOutDouble(buffer,"V_opt",V_opt,xDim*yDim,0);
    writeOutDouble(buffer,"K",K,xDim*yDim,0);
    writeOutDouble(buffer,"xPy",xPy,xDim*yDim,0);
    writeOutDouble(buffer,"yPx",yPx,xDim*yDim,0);
    writeOut(buffer,"WFC",wfc,xDim*yDim,0);
    writeOut(buffer,"ExPy",ExPy,xDim*yDim,0);
    writeOut(buffer,"EyPx",EyPx,xDim*yDim,0);
    writeOutDouble(buffer,"Phi",Phi,xDim*yDim,0);
    writeOutDouble(buffer,"r",r,xDim*yDim,0);
    writeOutDouble(buffer,"x",x,xDim,0);
    writeOutDouble(buffer,"y",y,yDim,0);
    writeOutDouble(buffer,"px",xp,xDim,0);
    writeOutDouble(buffer,"py",yp,yDim,0);
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    //free(V);
    free(K); free(r); //free(Phi);

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    sum=sqrt(sum*dx*dy);
    //#pragma omp parallel for reduction(+:sum) private(j)
    for (i = 0; i < xDim; i++){
        for (j = 0; j < yDim; j++){
            wfc[(i*yDim + j)].x = (wfc[(i*yDim + j)].x)/(sum);
            wfc[(i*yDim + j)].y = (wfc[(i*yDim + j)].y)/(sum);
        }
    }

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    result = cufftPlan2d(&plan_2d, xDim, yDim, CUFFT_Z2Z);
    if(result != CUFFT_SUCCESS){
        printf("Result:=%d\n",result);
        printf("Error: Could not execute cufftPlan2d(%s ,%d, %d).\n", "plan_2d", (unsigned int)xDim, (
unsigned int)yDim);
        return -1;
    }

    result = cufftPlan1d(&plan_1d, xDim, CUFFT_Z2Z, yDim);
    if(result != CUFFT_SUCCESS){
        printf("Result:=%d\n",result);
        printf("Error: Could not execute cufftPlan3d(%s ,%d ,%d ).\n", "plan_1d", (unsigned int)xDim, (
unsigned int)yDim);
        return -1;
    }

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    return 0;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.67.1.5 int isError (int *result,* char ∗ *c*)

Definition at line 42 of file split_op.cu.

References result.

```
                                    {
    if(result!=0){printf("Error has occurred for method %s with return type %d\n",
c,result);
        exit(result);
    }
    return result;
}
```

4.67.1.6 int main (int *argc,* char ∗∗ *argv*)

Definition at line 845 of file split_op.cu.

References ang_mom, atoms, buffer, complexMult(), delta_define(), device, dx, dy, EK, err, esteps, EV, evolve(), ExPy, EyPx, GK, gpe, gsteps, GV, vis::i, initArr(), initialise(), K_gpu, l, omegaX, omegaY, par_sum, parseArgs(), PI, print, read_wfc, readIn(), timeTotal, V_gpu, V_opt, wfc, wfc_gpu, writeOutDouble(), writeOutParam(), x, x0_shift, xDim, xPy, xPy_gpu, y, y0_shift, yDim, yPx, and yPx_gpu.

```
                                   {

    time_t start,fin;
    time(&start);
    printf("Start: %s\n", ctime(&start));
    initArr(&params,32);
    //appendData(&params,ctime(&start),0.0);
    parseArgs(argc,argv);
    cudaSetDevice(device);
    //************************************************************//
    /*
    * Initialise the Params data structure to track params and variables
    */
    //************************************************************//
    //paramS = (Params *) malloc(sizeof(Params));
    //strcpy(paramS->data,"INIT");
    //paramS->next=NULL;

    initialise(omegaX,omegaY,atoms);
    timeTotal = 0.0;
    //************************************************************//
    /*
    * Groundstate finder section
    */
    //************************************************************//
    writeOutParam(buffer, params, "Params.dat");
    if(read_wfc == 1){
        printf("Loading wavefunction...");
        wfc=readIn("wfc_load","wfci_load",xDim, yDim);
        printf("Wavefunction loaded.\n");
    }

    double2 ph;
    double x_0,y_0;
    x_0 = 0;//(0.5*xDim)*dx;
    y_0 = 0;//(0.5*yDim)*dy;
    for(int i=0; i < xDim; i++ ){
        for(int j=0; j < yDim; j++ ){
            ph.x = cos( fmod( l*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
            ph.y = sin( fmod( l*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
            wfc[(i*yDim + j)] = complexMult( wfc[(i*yDim + j)], ph );
        }
    }
    printf("l=%e\n",l);
    if(gsteps > 0){
        err=cudaMemcpy(K_gpu, GK, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(V_gpu, GV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(xPy_gpu, xPy, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(yPx_gpu, yPx, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);

        evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
xPy_gpu, par_sum, xDim*yDim, gsteps, 128, 0, ang_mom,
gpe, print, atoms, 0);
        cudaMemcpy(wfc, wfc_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
    }

    free(GV); free(GK); free(xPy); free(yPx);

    //************************************************************//
    /*
    * Evolution
    */
    //************************************************************//
    if(esteps > 0){
        err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(K_gpu, EK, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
00932            exit(1);
00933        err=cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
00934        if(err!=cudaSuccess)
00935            exit(1);
00936        err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*xDim*yDim,
      cudaMemcpyHostToDevice);
00937        if(err!=cudaSuccess)
00938            exit(1);
00939
00940        delta_define(x, y, (523.6667 - 512 + x0_shift)*dx, (512.6667 - 512  +
      y0_shift)*dy, V_opt);
00941        writeOutDouble(buffer,"V_opt",V_opt,xDim*yDim,0);
00942 //  exit(1);
00943        evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
      xPy_gpu, par_sum, xDim*yDim, esteps, 128, 1, ang_mom,
      gpe, print, atoms, 0);
00944
00945    }
00946    free(EV); free(EK); free(ExPy); free(EyPx);
00947    free(x);free(y);
00948    cudaFree(wfc_gpu); cudaFree(K_gpu); cudaFree(V_gpu); cudaFree(
      yPx_gpu); cudaFree(xPy_gpu); cudaFree(par_sum);
00949
00950    time(&fin);
00951    //appendData(&params,ctime(&fin),0.0);
00952    printf("Finish: %s\n", ctime(&fin));
00953    printf("Total time: %ld seconds\n ",(long)fin-start);
00954    //appendData(&params,"t_duration",fin-start);
00955    return 0;
00956 }
```

Here is the call graph for this function:

4.67.1.7 void optLatSetup (struct Vortex *centre,* double ∗ *V,* struct Vortex ∗ *vArray,* int *num_vortices,* double *theta_opt,* double *intensity,* double ∗ *v_opt,* double ∗ *x,* double ∗ *y*)

Matches the optical lattice to the vortex lattice.

Definition at line 602 of file split_op.cu.

References appendData(), dt, dx, EV_opt, HBAR, vis::i, PI, vortSepAvg(), xDim, and yDim.

```
00602
                                                        {
00603    int i,j;
00604    double sepMin = vortSepAvg(vArray,centre,num_vortices)*dx;
00605    appendData(&params,"Vort_sep",(double)sepMin);
00606    double k= (2*PI/sqrt(3))/sepMin;
00607    double x_shift = 0.0;//((xDim/2) - centre.x)*dx;//These values may need to be negated. As of yet
       unsure.
00608    double y_shift = 0.0;//((yDim/2) - centre.y)*dy;
00609    appendData(&params,"2PI/sqrt(3)/a",(double)k);
00610    //#pragma omp parallel for private(j)
00611    for ( j=0; j<yDim; ++j ){
00612        for ( i=0; i<xDim; ++i ){
00613            v_opt[j*xDim + i] = intensity*(
00614                              pow(abs(cos( sin( 2*PI/3 + theta_opt)*(x[i]*k + x_shift) + cos( 2*
      PI/3 + theta_opt )*(y[j]*k + y_shift))),2)
00615                        + 0*pow(abs(cos( sin( 4*PI/3 + theta_opt)*(x[i]*k + x_shift) + cos( 4*
      PI/3 + theta_opt )*(y[j]*k + y_shift))),2)
00616                        + 0*pow(abs(cos( sin(    0    + theta_opt)*(x[i]*k + x_shift) + cos(    0    +
      theta_opt )*(y[j]*k + y_shift))),2)
00617                        );
00618            EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim +
      i])*(dt/(2*HBAR)));
00619            EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(
      dt/(2*HBAR)));
00620        }
00621    }
00622
00623 }
```

Here is the call graph for this function:

4.67.1.8 int parseArgs (int *argc,* char ∗∗ *argv*)

Definition at line 687 of file split_op.cu.

References ang_mom, angle_sweep, appendData(), atoms, device, dt, esteps, gammaY, gdt, gpe, gsteps, interaction, kick_it, l, laser_power, omega, omegaX, omegaY, omegaZ, print, read_wfc, write_it, x0_shift, xDim, y0_shift, and yDim.

Referenced by main().

```
                                    {
    int opt;
    while ((opt = getopt (argc, argv, "d:x:y:w:G:g:e:T:t:n:p:r:o:L:l:s:i:P:X:Y:O:k:W:U:V:")) != -1) {
        switch (opt)
        {
            case 'x':
                xDim = atoi(optarg);
                printf("Argument for x is given as %d\n",xDim);
                appendData(&params,"xDim",(double)xDim);
                break;
            case 'y':
                yDim = atoi(optarg);
                printf("Argument for y is given as %d\n",yDim);
                appendData(&params,"yDim",(double)yDim);
                break;
            case 'w':
                omega = atof(optarg);
                printf("Argument for OmegaRotate is given as %E\n",omega);
                appendData(&params,"omega",omega);
                break;
            case 'G':
                gammaY = atof(optarg);
                printf("Argument for gamma is given as %E\n",gammaY);
                appendData(&params,"gammaY",gammaY);
                break;
            case 'g':
                gsteps = atof(optarg);
                printf("Argument for Groundsteps is given as %ld\n",gsteps);
                appendData(&params,"gsteps",gsteps);
                break;
            case 'e':
                esteps = atof(optarg);
                printf("Argument for EvSteps is given as %ld\n",esteps);
                appendData(&params,"esteps",esteps);
                break;
            case 'T':
                gdt = atof(optarg);
                printf("Argument for groundstate Timestep is given as %E\n",gdt);
                appendData(&params,"gdt",gdt);
                break;
            case 't':
                dt = atof(optarg);
                printf("Argument for Timestep is given as %E\n",dt);
                appendData(&params,"dt",dt);
                break;
            case 'd':
                device = atoi(optarg);
                printf("Argument for device is given as %d\n",device);
                appendData(&params,"device",device);
                break;
            case 'n':
                atoms = atof(optarg);
                printf("Argument for atoms is given as %ld\n",atoms);
                appendData(&params,"atoms",atoms);
                break;
            case 'r':
                read_wfc  = atoi(optarg);
                printf("Argument for ReadIn is given as %d\n",read_wfc);
                appendData(&params,"read_wfc",(double)read_wfc);
                break;
            case 'p':
                print = atoi(optarg);
                printf("Argument for Printout is given as %d\n",print);
                appendData(&params,"print_out",(double)print);
                break;
            case 'L':
                l = atof(optarg);
                printf("Vortex winding is given as : %E\n",l);
                appendData(&params,"winding",l);
                break;
            case 'l':
                ang_mom = atoi(optarg);
                printf("Angular Momentum mode engaged: %d\n",ang_mom);
                appendData(&params,"corotating",(double)ang_mom);
                break;
            case 's':
                gpe = atoi(optarg);
                printf("Non-linear mode engaged: %d\n",gpe);
                appendData(&params,"gpe",gpe);
                break;
            case 'o':
                omegaZ = atof(optarg);
                printf("Argument for OmegaZ is given as %E\n",omegaZ);
                appendData(&params,"omegaZ",omegaZ);
                break;
            case 'i':
                interaction = atof(optarg);
                printf("Argument for interaction scaling is %E\n",interaction);
                appendData(&params,"int_scaling",interaction);
                break;
            case 'P':
                laser_power = atof(optarg);
                printf("Argument for laser power is %E\n",laser_power);
                appendData(&params,"laser_power",laser_power);
                break;
            case 'X':
                omegaX = atof(optarg);
                printf("Argument for omegaX is %E\n",omegaX);
                appendData(&params,"omegaX",omegaX);
                break;
            case 'Y':
                omegaY = atof(optarg);
                printf("Argument for omegaY is %E\n",omegaY);
                appendData(&params,"omegaY",omegaY);
                break;
            case 'O':
                angle_sweep = atof(optarg);
                printf("Argument for angle_sweep is %E\n",angle_sweep);
                appendData(&params,"angle_sweep",angle_sweep);
                break;
            case 'k':
                kick_it = atoi(optarg);
                printf("Argument for kick_it is %i\n",kick_it);
                appendData(&params,"kick_it",kick_it);
                break;
            case 'W':
                write_it = atoi(optarg);
                printf("Argument for write_it is %i\n",write_it);
                appendData(&params,"write_it",write_it);
                break;
            case 'U':
                x0_shift = atof(optarg);
                printf("Argument for x0_shift is %lf\n",x0_shift);
                appendData(&params,"x0_shift",x0_shift);
                break;
            case 'V':
                y0_shift = atof(optarg);
                printf("Argument for y0_shift is %lf\n",y0_shift);
                appendData(&params,"y0_shift",y0_shift);
                break;
            case '?':
                if (optopt == 'c') {
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
                } else if (isprint (optopt)) {
                    fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                } else {
                    fprintf (stderr,"Unknown option character `\\x%x'.\n",optopt);
                }
                return -1;
            default:
                abort ();
        }
    }
    return 0;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.67.1.9 void parSum (double2 ∗ *gpuWfc,* double2 ∗ *gpuParSum,* int *xDim,* int *yDim,* int *threads*)

Definition at line 578 of file split_op.cu.

References dx, dy, threads, and yDim.

Referenced by evolve().

```
00578                                                                                      {
00579         int grid_tmp = xDim*yDim;
00580         int block = grid_tmp/threads;
00581         int thread_tmp = threads;
00582         int pass = 0;
00583         while((double)grid_tmp/threads > 1.0){
00584             if(grid_tmp == xDim*yDim){
00585                 multipass<<<block,threads,threads*sizeof(double2)>>>(&gpuWfc[0],&gpuParSum[0],pass);
00586             }
00587             else{
00588                 multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass
      );
00589             }
00590             grid_tmp /= threads;
00591             block = (int) ceil((double)grid_tmp/threads);
00592             pass++;
00593         }
00594         thread_tmp = grid_tmp;
00595         multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00596         scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
00597 }
```

Here is the caller graph for this function:

4.67.1.10 template<typename T > void parSum (T ∗ *gpuToSumArr*, T ∗ *gpuParSum*, int *xDim*, int *yDim*, int *threads*)

Definition at line 665 of file split_op.cu.

References dx, dy, threads, and yDim.

```
00665                                                                                                {
00666                 int grid_tmp = xDim*yDim;
00667                 int block = grid_tmp/threads;
00668                 int thread_tmp = threads;
00669                 int pass = 0;
00670                 while((double)grid_tmp/threads > 1.0){
00671                         if(grid_tmp == xDim*yDim){
00672                                 multipass<<<block,threads,threads*sizeof(T)>>>(&gpuToSumArr[0],&gpuParSum[0
      ],pass);
00673                         }
00674                         else{
00675                                 multipass<<<block,thread_tmp,thread_tmp*sizeof(T)>>>(&gpuParSum[0],&
      gpuParSum[0],pass);
00676                         }
00677                         grid_tmp /= threads;
00678                         block = (int) ceil((double)grid_tmp/threads);
00679                         pass++;
00680                 }
00681                 thread_tmp = grid_tmp;
00682                 multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00683                 scalarDiv_wfcNorm<<<grid,threads>>>(gpuToSumArr, dx*dy, gpuParSum, gpuToSumArr);
00684 }
```

4.67.2 Variable Documentation

4.67.2.1 double a0x

Definition at line 39 of file split_op.cu.

Referenced by initialise().

4.67.2.2 double a0y

Definition at line 39 of file split_op.cu.

Referenced by initialise().

4.67.2.3 double angle_sweep

Definition at line 34 of file split_op.cu.

Referenced by evolve(), and parseArgs().

4.67.2.4 char buffer[100]

Definition at line 27 of file split_op.cu.

Referenced by evolve(), initialise(), and main().

4.67.2.5 int device

Definition at line 29 of file split_op.cu.

Referenced by main(), and parseArgs().

4.67.2.6 double gammaY

Definition at line 31 of file split_op.cu.

Referenced by initialise(), and parseArgs().

4.67.2.7 int kick_it

Definition at line 30 of file split_op.cu.

Referenced by evolve(), and parseArgs().

4.67.2.8 double omega

Definition at line 32 of file split_op.cu.

Referenced by evolve(), initialise(), and parseArgs().

4.67.2.9 Params$*$ paramS

Definition at line 35 of file split_op.cu.

4.67.2.10 Array params

Definition at line 36 of file split_op.cu.

4.67.2.11 double Rxy

Definition at line 38 of file split_op.cu.

Referenced by initialise().

4.67.2.12 double timeTotal

Definition at line 33 of file split_op.cu.

Referenced by main().

4.67.2.13 int verbose

Definition at line 28 of file split_op.cu.

4.67.2.14 double x0_shift

Definition at line 37 of file split_op.cu.

Referenced by main(), and parseArgs().

4.67.2.15 double y0_shift

Definition at line 37 of file split_op.cu.

Referenced by main(), and parseArgs().

4.68 split_op.cu

```
/*
* split_op.cu - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#include "../include/split_op.h"
#include "../include/kernels.h"
#include "../include/constants.h"
#include "../include/fileIO.h"
#include "../include/tracker.h"
#include "../include/minions.h"
#include "../include/ds.h"

char buffer[100];
int verbose;
int device;
int kick_it;
double gammaY;
double omega;
double timeTotal;
double angle_sweep;
Params *paramS;
Array params;
double x0_shift, y0_shift;
double Rxy;
double a0x, a0y;
/* Buffer and FILE for IO */

int isError(int result, char* c){
    if(result!=0){printf("Error has occurred for method %s with return type %d\n",c,result);
        exit(result);
    }
    return result;
}
int initialise(double omegaX, double omegaY, int N){
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
    unsigned int xD=1,yD=1,zD=1;
    threads = 128;
    unsigned int b = xDim*yDim/threads;  //number of blocks in simulation
    unsigned long long maxElements = 65536*65536ULL; //largest number of elements

    if( b < (1<<16) ){
        xD = b;
    }
    else if( (b >= (1<<16) ) && (b <= (maxElements)) ){
        int t1 = log(b)/log(2);
        float t2 = (float) t1/2;
        t1 = (int) t2;
        if(t2 > (float) t1){
            xD <<= t1;
            yD <<= (t1 + 1);
        }
        else if(t2 == (float) t1){
            xD <<= t1;
            yD <<= t1;
        }
    }
    else{
        printf("Outside range of supported indexing");
        exit(-1);
    }
    printf("Compute grid dimensions chosen as X=%d  Y=%d\n",xD,yD);

    grid.x=xD;
    grid.y=yD;
    grid.z=zD;
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    unsigned int i,j; //Used in for-loops for indexing

    unsigned int gSize = xDim*yDim;
    double xOffset, yOffset;
    xOffset=0.0;//5.0e-6;
    yOffset=0.0;//5.0e-6;

    mass = 1.4431607e-25; //Rb 87 mass, kg
    appendData(&params,"Mass",mass);
    a_s = 4.67e-9;
    appendData(&params,"a_s",a_s);

    double sum = 0.0;

    a0x = sqrt(HBAR/(2*mass*omegaX));
    a0y = sqrt(HBAR/(2*mass*omegaY));
    appendData(&params,"a0x",a0x);
    appendData(&params,"a0y",a0y);

    Rxy = pow(15,0.2)*pow(N*a_s*sqrt(mass*omegaZ/HBAR),0.2);
    appendData(&params,"Rxy",Rxy);
    //Rxy = pow(15,0.2)*pow(N*4.67e-9*sqrt(mass*pow(omegaX*omegaY,0.5)/HBAR),0.2);
    double bec_length = sqrt( HBAR/mass*sqrt( omegaX*omegaX * ( 1 - omega*omega) ) );
    xMax = 6*Rxy*a0x;//10*bec_length;//6*Rxy*a0x;
    yMax = 6*Rxy*a0y;//10*bec_length;//
    appendData(&params,"xMax",xMax);
    appendData(&params,"yMax",yMax);

    double pxMax, pyMax;
    pxMax = (PI/xMax)*(xDim>>1);
    pyMax = (PI/yMax)*(yDim>>1);
    appendData(&params,"pyMax",pyMax);
    appendData(&params,"pxMax",pxMax);

    dx = xMax/(xDim>>1);
    dy = yMax/(yDim>>1);
    appendData(&params,"dx",dx);
    appendData(&params,"dy",dy);

    double dpx, dpy;
    dpx = PI/(xMax);
    dpy = PI/(yMax);
    appendData(&params,"dpx",dpx);
    appendData(&params,"dpy",dpy);

    //printf("a0x=%e  a0y=%e \n dx=%e   dx=%e\n R_xy=%e\n",a0x,a0y,dx,dy,Rxy);
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    //double *x,*y,*xp,*yp;
    x = (double *) malloc(sizeof(double) * xDim);
    y = (double *) malloc(sizeof(double) * yDim);
    xp = (double *) malloc(sizeof(double) * xDim);
    yp = (double *) malloc(sizeof(double) * yDim);

    /*
     * Pos and Mom grids
     */
    for(i=0; i<xDim/2; ++i){
        x[i] = -xMax + (i+1)*dx;
        x[i + (xDim/2)] = (i+1)*dx;

        y[i] = -yMax + (i+1)*dy;
        y[i + (yDim/2)] = (i+1)*dy;

        xp[i] = (i+1)*dpx;
        xp[i + (xDim/2)] = -pxMax + (i+1)*dpx;

        yp[i] = (i+1)*dpy;
        yp[i + (yDim/2)] = -pyMax + (i+1)*dpy;
    }

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    /* Initialise wavefunction, momentum and position operators on host */
    Energy = (double*) malloc(sizeof(double) * gSize);
    r = (double *) malloc(sizeof(double) * gSize);
    Phi = (double *) malloc(sizeof(double) * gSize);
    wfc = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    wfc_backup = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * (gSize/threads));
    K = (double *) malloc(sizeof(double) * gSize);
    V = (double *) malloc(sizeof(double) * gSize);
    V_opt = (double *) malloc(sizeof(double) * gSize);
    GK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    GV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EK = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EV = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EV_opt = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    xPy = (double *) malloc(sizeof(double) * gSize);
    yPx = (double *) malloc(sizeof(double) * gSize);
//  GxPy = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
//  GyPx = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    ExPy = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EyPx = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);
    EappliedField = (cufftDoubleComplex *) malloc(sizeof(cufftDoubleComplex) * gSize);

    /* Initialise wfc, EKp, and EVr buffers on GPU */
    cudaMalloc((void**) &Energy_gpu, sizeof(double) * gSize);
    cudaMalloc((void**) &wfc_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &K_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &V_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &xPy_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &yPx_gpu, sizeof(cufftDoubleComplex) * gSize);
    cudaMalloc((void**) &par_sum, sizeof(cufftDoubleComplex) * (gSize/threads));
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    #ifdef __linux
    int cores = omp_get_num_procs();
    appendData(&params,"Cores_Total",cores);
    appendData(&params,"Cores_Max",cores/2);
    omp_set_num_threads(cores/2);
    #pragma omp parallel for private(j)
    #endif
    for( i=0; i < xDim; i++ ){
        for( j=0; j < yDim; j++ ){
            //Remember, you are going from -PI to +PI, not 0 to 2PI
            if(x[i]>=0){
                Phi[(i*yDim + j)] = atan((y[j] + dx/10)/(x[i]) ) - PI/2.0;
            }
            else
                Phi[(i*yDim + j)] = atan((y[j] + dx/10)/(x[i]) ) + PI/2.0;

            Phi[(i*yDim + j)] = fmod(l*Phi[(i*xDim + j)],2*PI);

            wfc[(i*yDim + j)].x = exp(-( pow((x[i])/(Rxy*a0x),2) + pow((y[j])/(Rxy*a0y),2) ) )*cos(Phi[(i*xDim + j)]);
            wfc[(i*yDim + j)].y = -exp(-( pow((x[i])/(Rxy*a0x),2) + pow((y[j])/(Rxy*a0y),2) ) )*sin(Phi[(i*xDim + j)]);

            V[(i*yDim + j)] = 0.5*mass*( pow(omegaX*(x[i]+xOffset),2) + pow(gammaY*omegaY*(y[j]+yOffset),2) );
            K[(i*yDim + j)] = (HBAR*HBAR/(2*mass))*(xp[i]*xp[i] + yp[j]*yp[j]);

            //V_opt[i*yDim + j] = cos(sin(2*PI/3)*(x[i]/(xMax*0.01)) + cos(2*PI/3)*(y[j]/(yMax*0.01)))
            //        + cos(sin(4*PI/3)*(x[i]/(xMax*0.01)) + cos(4*PI/3)*(y[j]/(yMax*0.01)))
            //        + cos(sin(2*PI)*(x[i]/(xMax*0.01)) + cos(2*PI)*(y[j]/(yMax*0.01)));

            GV[(i*yDim + j)].x = exp( -V[(i*xDim + j)]*(gdt/(2*HBAR)));
            GK[(i*yDim + j)].x = exp( -K[(i*xDim + j)]*(gdt/HBAR));
            GV[(i*yDim + j)].y = 0.0;
            GK[(i*yDim + j)].y = 0.0;

            xPy[(i*yDim + j)] = x[i]*yp[j];
            yPx[(i*yDim + j)] = -y[j]*xp[i];

//          GxPy[(i*yDim + j)].x = exp( -omega*xPy[(i*yDim + j)]*gdt);
//          GxPy[(i*yDim + j)].y = 0.0;
//          GyPx[(i*yDim + j)].x = exp( -omega*yPx[(i*yDim + j)]*gdt);
//          GyPx[(i*yDim + j)].y = 0.0;

            EV[(i*yDim + j)].x=cos( -V[(i*xDim + j)]*(dt/(2*HBAR)));
            EV[(i*yDim + j)].y=sin( -V[(i*xDim + j)]*(dt/(2*HBAR)));
            EK[(i*yDim + j)].x=cos( -K[(i*xDim + j)]*(dt/HBAR));
            EK[(i*yDim + j)].y=sin( -K[(i*xDim + j)]*(dt/HBAR));

            ExPy[(i*yDim + j)].x=cos(-omega*omegaX*xPy[(i*xDim + j)]*dt);
            ExPy[(i*yDim + j)].y=sin(-omega*omegaX*xPy[(i*xDim + j)]*dt);
            EyPx[(i*yDim + j)].x=cos(-omega*omegaX*yPx[(i*xDim + j)]*dt);
            EyPx[(i*yDim + j)].y=sin(-omega*omegaX*yPx[(i*xDim + j)]*dt);

            sum+=sqrt(wfc[(i*xDim + j)].x*wfc[(i*xDim + j)].x + wfc[(i*xDim + j)].y*wfc[(i*xDim + j)].y);
        }
    }
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
    //hdfWriteDouble(xDim, V, 0, "V_0");
    //hdfWriteComplex(xDim, wfc, 0, "wfc_0");
    writeOutDouble(buffer,"V",V,xDim*yDim,0);
    //writeOutDouble(buffer,"V_opt",V_opt,xDim*yDim,0);
    writeOutDouble(buffer,"K",K,xDim*yDim,0);
    writeOutDouble(buffer,"xPy",xPy,xDim*yDim,0);
    writeOutDouble(buffer,"yPx",yPx,xDim*yDim,0);
    writeOut(buffer,"WFC",wfc,xDim*yDim,0);
    writeOut(buffer,"ExPy",ExPy,xDim*yDim,0);
    writeOut(buffer,"EyPx",EyPx,xDim*yDim,0);
    writeOutDouble(buffer,"Phi",Phi,xDim*yDim,0);
    writeOutDouble(buffer,"r",r,xDim*yDim,0);
    writeOutDouble(buffer,"x",x,xDim,0);
    writeOutDouble(buffer,"y",y,yDim,0);
    writeOutDouble(buffer,"px",xp,xDim,0);
    writeOutDouble(buffer,"py",yp,yDim,0);
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    //free(V);
    free(K); free(r); //free(Phi);

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    sum=sqrt(sum*dx*dy);
    //#pragma omp parallel for reduction(+:sum) private(j)
    for (i = 0; i < xDim; i++){
        for (j = 0; j < yDim; j++){
            wfc[(i*yDim + j)].x = (wfc[(i*yDim + j)].x)/(sum);
            wfc[(i*yDim + j)].y = (wfc[(i*yDim + j)].y)/(sum);
        }
    }

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    result = cufftPlan2d(&plan_2d, xDim, yDim, CUFFT_Z2Z);
    if(result != CUFFT_SUCCESS){
        printf("Result:=%d\n",result);
        printf("Error: Could not execute cufftPlan2d(%s ,%d, %d).\n", "plan_2d", (unsigned int)xDim, (unsigned int)yDim);
        return -1;
    }

    result = cufftPlan1d(&plan_1d, xDim, CUFFT_Z2Z, yDim);
    if(result != CUFFT_SUCCESS){
        printf("Result:=%d\n",result);
        printf("Error: Could not execute cufftPlan3d(%s ,%d ,%d ).\n", "plan_1d", (unsigned int)xDim, (unsigned int)yDim);
        return -1;
    }

    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//

    return 0;
}

int evolve( cufftDoubleComplex *gpuWfc,
            cufftDoubleComplex *gpuMomentumOp,
            cufftDoubleComplex *gpuPositionOp,
            void *gpu1dyPx,
            void *gpu1dxPy,
            cufftDoubleComplex *gpuParSum,
            int gridSize, int numSteps, int threads,
            unsigned int gstate, int lz, int nonlin, int printSteps, int N, unsigned int ramp){

    //Because no two operations are created equally. Multiplimultiplication is faster than divisions.
    double renorm_factor_2d=1.0/pow(gridSize,0.5);
    double renorm_factor_1d=1.0/pow(xDim,0.5);

    clock_t begin, end;
    double time_spent;
    double Dt;
    if(gstate==0){
        Dt = gdt;
        printf("Timestep for grounstate solver set as: %E\n",Dt);
    }
    else{
        Dt = dt;
        printf("Timestep for evolution set as: %E\n",Dt);
    }
    begin = clock();
    double omega_0=omega*omegaX;

    #if 0

    int gridSum = 1<<6;
    double *densitySubset = (double*) malloc(sizeof(double)*gridSum);
    #pragma omp parallel for private(k)
    for (int j=0; j<gridSum; ++j){
        for (int k=0; k<gridSum; ++k){
            densitySubset[j*gridSum + k] = psi2(wfc[ ( (yDim/2) - (gridSum/2) + j )*yDim  + ( (xDim/2)  - (gridSum/2) + k )]);
        }
    }
    xi = 1/sqrt(8*PI*a_s*sumAvg(densitySubset,gridSum)/(dx*dy));//defined central condensate density
    printf("Avg healing length at centre=%E\n",xi);
    #endif

    //Double buffering and will attempt to thread free and calloc operations to hide time penalty. Or may not bother.
    int num_vortices[2] = {0,0};
    int num_latt_max = 0;
    int* vortexLocation; //binary matrix of size xDim*yDim, 1 for vortex at specified index, 0 otherwise
    int* olMaxLocation = (int*) calloc(xDim*yDim,sizeof(int));

    struct Vortex central_vortex; //vortex closest to the central position
    double vort_angle; //Angle of vortex lattice. Add to optical lattice for alignment.
    struct Vortex *vortCoords = NULL; //array of vortex coordinates from vortexLocation 1's
    struct Vortex *vortCoordsP = NULL; //Previous array of vortex coordinates from vortexLocation 1's
    int2 *olCoords = NULL; //array of vortex coordinates from vortexLocation 1's
    int2 *vortDelta = NULL;

    double vortOLSigma=0.0;
    double sepAvg = 0.0;

    int num_kick = 0;
    double t_kick = (2*PI/omega_0)/(6*Dt);

    for(int i=0; i < numSteps; ++i){
        if ( ramp == 1 ){
            omega_0=omegaX*((omega-0.39)*((double)i/(double)(numSteps)) + 0.39); //Adjusts omega for the appropriate trap frequency.
        }
        if(i % printSteps == 0){
            printf("Step: %d    Omega: %lf\n",i,omega_0/omegaX);
            cudaMemcpy(wfc, gpuWfc, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
            end = clock();
            time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
            printf("Time spent: %lf\n",time_spent);
            char* fileName = "";
            printf("ramp=%d     gstate=%d   rg=%d       \n",ramp,gstate,ramp | (gstate<<1));
            switch ( ramp | (gstate<<1) ){
                case 0:
                    fileName = "wfc_0_const";
                    break;
                case 1:
                    fileName = "wfc_0_ramp";
                    break;
                case 2:
                    fileName = "wfc_ev";
                    vortexLocation = (int*) calloc(xDim*yDim,sizeof(int));
                    num_vortices[0] = findVortex(vortexLocation, wfc, 1e-4, xDim, x, i);
                    if(i==0){
                        vortCoords = (struct Vortex*) malloc(sizeof(struct Vortex)*(2*num_vortices[0]));
                        vortCoordsP = (struct Vortex*) malloc(sizeof(struct Vortex)*(2*num_vortices[0]));
                        vortPos(vortexLocation, vortCoords, xDim, wfc);
                        central_vortex = vortCentre(vortCoords, num_vortices[0], xDim);
                        //if(angle_sweep==0.0)
                            vort_angle = vortAngle(vortCoords,central_vortex, num_vortices[0] + PI*angle_sweep/180.0);
                        //else
                        //  vort_angle = angle_sweep;
                        appendData(&params,"Vort_angle",vort_angle);
                        //optLatSetup(central_vortex, V, vortCoords, num_vortices[0], vort_angle, laser_power*HBAR*sqrt(omegaX*omegaY), V_opt, x, y);
                        sepAvg = vortSepAvg(vortCoords,central_vortex,num_vortices[0]);
                        if(kick_it == 2){
                            printf("Kicked it 1\n");
                            cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*
xDim*yDim, cudaMemcpyHostToDevice);
                        }
                        writeOutDouble(buffer,"V_opt_1",V_opt,xDim*yDim,0);
                        writeOut(buffer,"EV_opt_1",EV_opt,xDim*yDim,0);
                        appendData(&params,"Central_vort_x",(double)central_vortex.coords.x);
                        appendData(&params,"Central_vort_y",(double)central_vortex.coords.y);
                        appendData(&params,"Central_vort_winding",(double)central_vortex.wind);
                        appendData(&params,"Central_vort_sign",(double)central_vortex.sign);
                        appendData(&params,"Num_vort",(double)num_vortices[0]);
                        writeOutParam(buffer, params, "Params.dat");
                    }
                    else if(num_vortices[0] > num_vortices[1]){
                        printf("Number of vortices changed from %d to %d\n",num_vortices[1],num_vortices[0]);
                        vortPos(vortexLocation, vortCoords, xDim,wfc);
                    }
                    else{
                        vortPos(vortexLocation, vortCoords, xDim,wfc);
                        vortArrange(vortCoords, vortCoordsP, num_vortices[0]);
                    }
            /*      num_latt_max = findOLMaxima(olMaxLocation, V_opt, 1e-4, xDim, x);
                    if(num_latt_max == num_vortices[0]){
                        olCoords = (int2*) malloc(sizeof(int2)*num_latt_max);
                        olPos(olMaxLocation, olCoords, xDim);
                        vortOLSigma = sigVOL(vortCoords, olCoords, x, num_latt_max);
                        writeOutInt2(buffer, "opt_max_arr", olCoords, num_latt_max, i);
                        free(olCoords);
                    }*/
                    writeOutVortex(buffer, "vort_arr", vortCoords, num_vortices[0], i);
                    printf("Located %d vortices\n",num_vortices[0]);
                    printf("Sigma=%e\n",vortOLSigma);
                    free(vortexLocation);
                    num_vortices[1] = num_vortices[0];
                    memcpy(vortCoordsP,vortCoords,sizeof(int2)*num_vortices[0]);
                    break;
                case 3:
                    fileName = "wfc_ev_ramp";
                    break;
                default:
                    break;
            }
            if(write_it)
                writeOut(buffer, fileName, wfc, xDim*yDim, i);
            //printf("Energy[t@%d]=%E\n",i,energy_angmom(gpuPositionOp, gpuMomentumOp, dx, dy, gpuWfc,gstate));
/*          cudaMemcpy(V_gpu, V, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
            cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
            cudaMemcpy(V_gpu, , sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
            cudaMemcpy(K_gpu, K, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
*/      }

        if(i % (int) t_kick+1 == 0 && num_kick<=6 && gstate==1 && kick_it == 1 ){
            cudaMemcpy(V_gpu, EV_opt, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
            ++num_kick;
        }
        /*
         * U_r(dt/2)*wfc
         */
        if(nonlin == 1){
            cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,0.5*Dt, mass,omegaZ,gstate,N*interaction);
        }
        else {
            cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
        }

        /*
         * U_p(dt)*fft2(wfc)
         */
        result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD);
        scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc); //Normalise
        cMult<<<grid,threads>>>(gpuMomentumOp,gpuWfc,gpuWfc);
        result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE);
        scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc); //Normalise

        /*
         * U_r(dt/2)*wfc
         */
        if(nonlin == 1){
            cMultDensity<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc,Dt*0.5,mass,omegaZ,gstate,N*interaction);
        }
        else {
            cMult<<<grid,threads>>>(gpuPositionOp,gpuWfc,gpuWfc);
        }

        if( (i % (int)t_kick+1 ==0 && num_kick<=6 && gstate==1) || (kick_it >= 1 && i==0) ){
            cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
            printf("Got here\n");
        }
        /**************************************************************/
        /* Angular momentum xPy-yPx   */
        if(lz == 1){
            switch(i%2 | (gstate<<1)){
                case 0: //Groundstate solver, even step
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dxPy, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);

                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dyPx, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                break;

                case 1: //Groundstate solver, odd step
                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dyPx, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);

                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                angularOp<<<grid,threads>>>(omega_0, Dt, gpuWfc, (double*) gpu1dxPy, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                break;

                case 2: //Real time evolution, even step
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dxPy, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);

                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dyPx, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                break;

                case 3: //Real time evolution, odd step
                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_FORWARD); //2D forward
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE); //1D inverse to wfc_yPx
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dyPx, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_PxPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                result = cufftExecZ2Z(plan_2d,gpuWfc,gpuWfc,CUFFT_INVERSE); //2D Inverse
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_2d,gpuWfc);

                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_FORWARD); // wfc_xPy
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                cMult<<<grid,threads>>>(gpuWfc, (cufftDoubleComplex*) gpu1dxPy, gpuWfc);
                result = cufftExecZ2Z(plan_1d,gpuWfc,gpuWfc,CUFFT_INVERSE);
                scalarDiv<<<grid,threads>>>(gpuWfc,renorm_factor_1d,gpuWfc);
                break;

            }
        }
        /*************************************************************/

        if(gstate==0){
            parSum(gpuWfc, gpuParSum, xDim, yDim, threads);
        }
    }
    return 0;
}

/*
 * Used to perform parallel summation on WFC and normalise
 */
void parSum(double2* gpuWfc, double2* gpuParSum, int xDim, int yDim, int threads){
        int grid_tmp = xDim*yDim;
        int block = grid_tmp/threads;
        int thread_tmp = threads;
        int pass = 0;
        while((double)grid_tmp/threads > 1.0){
            if(grid_tmp == xDim*yDim){
                multipass<<<block,threads,threads*sizeof(double2)>>>(&gpuWfc[0],&gpuParSum[0],pass);
            }
            else{
                multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass);
            }
            grid_tmp /= threads;
            block = (int) ceil((double)grid_tmp/threads);
            pass++;
        }
        thread_tmp = grid_tmp;
        multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
        scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
}

void optLatSetup(struct Vortex centre, double* V, struct Vortex *vArray, int num_vortices, double theta_opt, double intensity, double* v_opt, double *x, double *y){
    int i,j;
    double sepMin = vortSepAvg(vArray,centre,num_vortices)*dx;
    appendData(&params,"Vort_sep",(double)sepMin);
    double k= (2*PI/sqrt(3))/sepMin;
    double x_shift = 0.0;//((xDim/2) - centre.x)*dx;//These values may need to be negated. As of yet unsure.
    double y_shift = 0.0;//((yDim/2) - centre.y)*dy;
    appendData(&params,"2PI/sqrt(3)/a",(double)k);
    //#pragma omp parallel for private(j)
    for ( j=0; j<yDim; ++j ){
        for ( i=0; i<xDim; ++i ){
            v_opt[j*xDim + i] = intensity*(
                                  pow(abs(cos( sin( 2*PI/3 + theta_opt)*(x[i]*k + x_shift) + cos( 2*PI/3 + theta_opt )*(y[j]*k + y_shift))),2)
                            + 0*pow(abs(cos( sin( 4*PI/3 + theta_opt)*(x[i]*k + x_shift) + cos( 4*PI/3 + theta_opt )*(y[j]*k + y_shift))),2)
                            + 0*pow(abs(cos( sin(    0    + theta_opt)*(x[i]*k + x_shift) + cos(    0    + theta_opt )*(y[j]*k + y_shift))),2)
                            );
            EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(dt/(2*HBAR)));
            EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(dt/(2*HBAR)));
        }
    }

}

double energy_angmom(double *Energy, double* Energy_gpu, double2 *V_op, double2 *K_op, double dx, double dy, double2 *gpuWfc, int gState){
    double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
    double result=0;

    for (int i=0; i < xDim*yDim; ++i){
        Energy[i] = 0.0;
    }


/*  cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);

    energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
    result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
    scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise

    energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
    result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
    scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise

    err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);

    for(int i=0; i<xDim*yDim; i++){
        result += energy[i].x;
        //printf("En=%E\n",result*dx*dy);
    }
    return result*dx*dy;
*/

}


//####################################################################################################################
//####################################################################################################################

/*
 * Used to perform parallel summation using templates from c++
 */
template<typename T> void parSum(T *gpuToSumArr, T *gpuParSum, int xDim, int yDim, int threads){
                int grid_tmp = xDim*yDim;
                int block = grid_tmp/threads;
                int thread_tmp = threads;
                int pass = 0;
                while((double)grid_tmp/threads > 1.0){
                        if(grid_tmp == xDim*yDim){
                                multipass<<<block,threads,threads*sizeof(T)>>>(&gpuToSumArr[0],&gpuParSum[0],pass);
                        }
                        else{
                                multipass<<<block,thread_tmp,thread_tmp*sizeof(T)>>>(&gpuParSum[0],&gpuParSum[0],pass);
                        }
                        grid_tmp /= threads;
                        block = (int) ceil((double)grid_tmp/threads);
                        pass++;
                }
                thread_tmp = grid_tmp;
                multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
                scalarDiv_wfcNorm<<<grid,threads>>>(gpuToSumArr, dx*dy, gpuParSum, gpuToSumArr);
}
//####################################################################################################################
//####################################################################################################################
int parseArgs(int argc, char** argv){
    int opt;
    while ((opt = getopt (argc, argv, "d:x:y:w:G:g:e:T:t:n:p:r:o:L:l:s:i:P:X:Y:O:k:W:U:V:")) != -1) {
        switch (opt)
        {
            case 'x':
                xDim = atoi(optarg);
                printf("Argument for x is given as %d\n",xDim);
                appendData(&params,"xDim",(double)xDim);
                break;
            case 'y':
                yDim = atoi(optarg);
                printf("Argument for y is given as %d\n",yDim);
                appendData(&params,"yDim",(double)yDim);
                break;
            case 'w':
                omega = atof(optarg);
                printf("Argument for OmegaRotate is given as %E\n",omega);
                appendData(&params,"omega",omega);
                break;
            case 'G':
                gammaY = atof(optarg);
                printf("Argument for gamma is given as %E\n",gammaY);
                appendData(&params,"gammaY",gammaY);
                break;
            case 'g':
                gsteps = atof(optarg);
                printf("Argument for Groundsteps is given as %ld\n",gsteps);
                appendData(&params,"gsteps",gsteps);
                break;
            case 'e':
                esteps = atof(optarg);
                printf("Argument for EvSteps is given as %ld\n",esteps);
                appendData(&params,"esteps",esteps);
                break;
            case 'T':
                gdt = atof(optarg);
                printf("Argument for groundstate Timestep is given as %E\n",gdt);
                appendData(&params,"gdt",gdt);
                break;
            case ’t’:
                dt = atof(optarg);
                printf("Argument for Timestep is given as %E\n",dt);
                appendData(&params,"dt",dt);
                break;
            case ’d’:
                device = atoi(optarg);
                printf("Argument for device is given as %d\n",device);
                appendData(&params,"device",device);
                break;
            case ’n’:
                atoms = atof(optarg);
                printf("Argument for atoms is given as %ld\n",atoms);
                appendData(&params,"atoms",atoms);
                break;
            case ’r’:
                read_wfc  = atoi(optarg);
                printf("Argument for ReadIn is given as %d\n",read_wfc);
                appendData(&params,"read_wfc",(double)read_wfc);
                break;
            case ’p’:
                print = atoi(optarg);
                printf("Argument for Printout is given as %d\n",print);
                appendData(&params,"print_out",(double)print);
                break;
            case ’L’:
                l = atof(optarg);
                printf("Vortex winding is given as : %E\n",l);
                appendData(&params,"winding",l);
                break;
            case ’l’:
                ang_mom = atoi(optarg);
                printf("Angular Momentum mode engaged: %d\n",ang_mom);
                appendData(&params,"corotating",(double)ang_mom);
                break;
            case ’s’:
                gpe = atoi(optarg);
                printf("Non-linear mode engaged: %d\n",gpe);
                appendData(&params,"gpe",gpe);
                break;
            case ’o’:
                omegaZ = atof(optarg);
                printf("Argument for OmegaZ is given as %E\n",omegaZ);
                appendData(&params,"omegaZ",omegaZ);
                break;
            case ’i’:
                interaction = atof(optarg);
                printf("Argument for interaction scaling is %E\n",interaction);
                appendData(&params,"int_scaling",interaction);
                break;
            case ’P’:
                laser_power = atof(optarg);
                printf("Argument for laser power is %E\n",laser_power);
                appendData(&params,"laser_power",laser_power);
                break;
            case ’X’:
                omegaX = atof(optarg);
                printf("Argument for omegaX is %E\n",omegaX);
                appendData(&params,"omegaX",omegaX);
                break;
            case ’Y’:
                omegaY = atof(optarg);
                printf("Argument for omegaY is %E\n",omegaY);
                appendData(&params,"omegaY",omegaY);
                break;
            case ’O’:
                angle_sweep = atof(optarg);
                printf("Argument for angle_sweep is %E\n",angle_sweep);
                appendData(&params,"angle_sweep",angle_sweep);
                break;
            case ’k’:
                kick_it = atoi(optarg);
                printf("Argument for kick_it is %i\n",kick_it);
                appendData(&params,"kick_it",kick_it);
                break;
            case ’W’:
                write_it = atoi(optarg);
                printf("Argument for write_it is %i\n",write_it);
                appendData(&params,"write_it",write_it);
                break;
            case ’U’:
                x0_shift = atof(optarg);
                printf("Argument for x0_shift is %lf\n",x0_shift);
                appendData(&params,"x0_shift",x0_shift);
                break;
            case 'V':
                y0_shift = atof(optarg);
                printf("Argument for y0_shift is %lf\n",y0_shift);
                appendData(&params,"y0_shift",y0_shift);
                break;
            case '?':
                if (optopt == 'c') {
                    fprintf (stderr, "Option -%c requires an argument.\n", optopt);
                } else if (isprint (optopt)) {
                    fprintf (stderr, "Unknown option `-%c'.\n", optopt);
                } else {
                    fprintf (stderr,"Unknown option character `\\x%x'.\n",optopt);
                }
                return -1;
            default:
                abort ();
        }
    }
    return 0;
}

void delta_define(double *x, double *y, double x0, double y0, double *delta){
    for (unsigned int i=0; i<xDim; ++i){
        for (unsigned int j=0; j<yDim; ++j){
            delta[j*xDim + i] = 1e6*HBAR*exp( -( pow( x[i] - x0, 2)  +  pow( y[j] - y0, 2) )/(5*dx*dx) );
            EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + delta[j*xDim + i])*(dt/(2*HBAR)));
            EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + delta[j*xDim + i])*(dt/(2*HBAR)));
        }
    }

}


int main(int argc, char **argv){

    time_t start,fin;
    time(&start);
    printf("Start: %s\n", ctime(&start));
    initArr(&params,32);
    //appendData(&params,ctime(&start),0.0);
    parseArgs(argc,argv);
    cudaSetDevice(device);
    //************************************************************//
    /*
    * Initialise the Params data structure to track params and variables
    */
    //************************************************************//
    //paramS = (Params *) malloc(sizeof(Params));
    //strcpy(paramS->data,"INIT");
    //paramS->next=NULL;

    initialise(omegaX,omegaY,atoms);
    timeTotal = 0.0;
    //************************************************************//
    /*
    * Groundstate finder section
    */
    //************************************************************//
    writeOutParam(buffer, params, "Params.dat");
    if(read_wfc == 1){
        printf("Loading wavefunction...");
        wfc=readIn("wfc_load","wfci_load",xDim, yDim);
        printf("Wavefunction loaded.\n");
    }

    double2 ph;
    double x_0,y_0;
    x_0 = 0;//(0.5*xDim)*dx;
    y_0 = 0;//(0.5*yDim)*dy;
    for(int i=0; i < xDim; i++ ){
        for(int j=0; j < yDim; j++ ){
            ph.x = cos( fmod( l*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
            ph.y = sin( fmod( l*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
            wfc[(i*yDim + j)] = complexMult( wfc[(i*yDim + j)], ph );
        }
    }
    printf("l=%e\n",l);
    if(gsteps > 0){
        err=cudaMemcpy(K_gpu, GK, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(V_gpu, GV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(xPy_gpu, xPy, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(yPx_gpu, yPx, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);

        evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
xPy_gpu, par_sum, xDim*yDim, gsteps, 128, 0, ang_mom,
gpe, print, atoms, 0);
        cudaMemcpy(wfc, wfc_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
    }

    free(GV); free(GK); free(xPy); free(yPx);

    //************************************************************//
    /*
    * Evolution
    */
    //************************************************************//
    if(esteps > 0){
        err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*xDim*
yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(K_gpu, EK, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);
        err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*xDim*yDim,
cudaMemcpyHostToDevice);
        if(err!=cudaSuccess)
            exit(1);

        delta_define(x, y, (523.6667 - 512 + x0_shift)*dx, (512.6667 - 512  +
y0_shift)*dy, V_opt);
        writeOutDouble(buffer,"V_opt",V_opt,xDim*yDim,0);
//  exit(1);
        evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
xPy_gpu, par_sum, xDim*yDim, esteps, 128, 1, ang_mom,
gpe, print, atoms, 0);

    }
    free(EV); free(EK); free(ExPy); free(EyPx);
    free(x);free(y);
    cudaFree(wfc_gpu); cudaFree(K_gpu); cudaFree(V_gpu); cudaFree(
yPx_gpu); cudaFree(xPy_gpu); cudaFree(par_sum);

    time(&fin);
    //appendData(&params,ctime(&fin),0.0);
    printf("Finish: %s\n", ctime(&fin));
    printf("Total time: %ld seconds\n ",(long)fin-start);
    //appendData(&params,"t_duration",fin-start);
    return 0;
}
```

4.69 src/srt.cc File Reference

Functions

- double sepAvg (int2 *vArray, int2 centre, int length)

4.69.1 Function Documentation

4.69.1.1 double sepAvg (int2 ∗ *vArray,* int2 *centre,* int *length*)

Definition at line 19 of file srt.cc.

References vis::i, and result.

Referenced by evolve().

```
00019                                                     {
00020     double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00021     for (int i=0; i<length; ++i){
00022         result += sqrt( pow(centre.x - v_array[i].x,2) + pow(centre.y - v_array[i].y,2));
00023     }
00024     return result/length;
00025 }
```

Here is the caller graph for this function:

4.70 srt.cc

```
00001 /*
00002 * tracker.cc - GPUE: Split Operator based GPU solver for Nonlinear
00003 * Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
00004 * Morgan, Neil Crowley.
00005
00006 * This library is free software; you can redistribute it and/or modify
00007 * it under the terms of the GNU Lesser General Public License as
00008 * published by the Free Software Foundation; either version 2.1 of the
00009 * License, or (at your option) any later version. This library is
00010 * distributed in the hope that it will be useful, but WITHOUT ANY
00011 * WARRANTY; without even the implied warranty of MERCHANTABILITY or
00012 * FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
00013 * License for more details. You should have received a copy of the GNU
00014 * Lesser General Public License along with this library; if not, write
00015 * to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
00016 * Boston, MA 02111-1307 USA
00017 */
00018
00019 double sepAvg(int2 *vArray, int2 centre, int length){
00020     double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00021     for (int i=0; i<length; ++i){
00022         result += sqrt( pow(centre.x - v_array[i].x,2) + pow(centre.y - v_array[i].y,2));
00023     }
00024     return result/length;
00025 }
```

4.71 src/test/test.cc File Reference

```
#include "../include/minions.h"
```

Include dependency graph for test.cc:

Functions

- int main ()

4.71.1 Function Documentation

4.71.1.1 int main ( )

Definition at line 2 of file test.cc.

References Vortex::coords, coordSwap(), vis::i, x, and y.

```
00002          {
00003     int2 *coords = (int2*) calloc(sizeof(int2),8);
00004     for (int i=0; i<8; ++i){
00005         coords[i].x = i;
00006         coords[i].y = i;
00007         printf("Coords[%d].x = %d\n",i,coords[i].x);
00008         printf("Coords[%d].y = %d\n",i,coords[i].y);
00009     }
00010     int src=1,dest=3;
00011     coordSwap(coords, src, dest);
00012     for (int i=0; i<8; ++i){
00013         printf("Coords[%d].x = %d\n",i,coords[i].x);
00014         printf("Coords[%d].y = %d\n",i,coords[i].y);
00015     }
00016     return 0;
00017 }
```

Here is the call graph for this function:

4.72 test.cc

```
00001 #include "../include/minions.h"
00002 int main(){
00003     int2 *coords = (int2*) calloc(sizeof(int2),8);
00004     for (int i=0; i<8; ++i){
00005         coords[i].x = i;
00006         coords[i].y = i;
00007         printf("Coords[%d].x = %d\n",i,coords[i].x);
00008         printf("Coords[%d].y = %d\n",i,coords[i].y);
00009     }
00010     int src=1,dest=3;
00011     coordSwap(coords, src, dest);
00012     for (int i=0; i<8; ++i){
00013         printf("Coords[%d].x = %d\n",i,coords[i].x);
00014         printf("Coords[%d].y = %d\n",i,coords[i].y);
00015     }
00016     return 0;
00017 }
```

4.73 src/test.cc File Reference

`#include "../include/minions.h"`

Include dependency graph for test.cc:

Functions

- int main ()

4.73.1 Function Documentation

4.73.1.1 int main ( )

Definition at line 2 of file test.cc.

References Vortex::coords, coordSwap(), vis::i, x, and y.

```
00002          {
00003     int2 *coords = (int2*) calloc(sizeof(int2),8);
00004     for (int i=0; i<8; ++i){
00005         coords[i].x = i;
00006         coords[i].y = i;
00007         printf("Coords[%d].x = %d\n",i,coords[i].x);
00008         printf("Coords[%d].y = %d\n",i,coords[i].y);
00009     }
00010     int src=1,dest=3;
00011     coordSwap(coords, src, dest);
00012     for (int i=0; i<8; ++i){
00013         printf("Coords[%d].x = %d\n",i,coords[i].x);
```

```
        printf("Coords[%d].y = %d\n",i,coords[i].y);
    }
    return 0;
}
```

Here is the call graph for this function:

4.74 test.cc

```
#include "../include/minions.h"
int main(){
    int2 *coords = (int2*) calloc(sizeof(int2),8);
    for (int i=0; i<8; ++i){
        coords[i].x = i;
        coords[i].y = i;
        printf("Coords[%d].x = %d\n",i,coords[i].x);
        printf("Coords[%d].y = %d\n",i,coords[i].y);
    }
    int src=1,dest=3;
    coordSwap(coords, src, dest);
    for (int i=0; i<8; ++i){
        printf("Coords[%d].x = %d\n",i,coords[i].x);
        printf("Coords[%d].y = %d\n",i,coords[i].y);
    }
    return 0;
}
```

4.75 src/test/test_fileIO.cc File Reference

```
#include "../../src/fileIO.cc"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <cuda_runtime.h>
```

Include dependency graph for test_fileIO.cc:

Functions

- int main (int argc, char **argv)

Variables

- char buffer [100]

4.75.1 Function Documentation

4.75.1.1 int main (int *argc,* char ** *argv*)

Definition at line 9 of file test_fileIO.cc.

References buffer, vis::i, r, readIn(), and writeOut().

```
00009                                   {
00010     double2 *r;
00011     char *filename1, *filename2;
00012     filename1 = "test_allocation_0.txt";
00013     filename2 = "test_allocation_1.txt";
00014     r = (double2 *) malloc(sizeof(double2)*100);
00015     for (int i=0; i<99; ++i){
00016         r[i].x = i*1.0;
00017         r[i].y = 0.0;
00018     }
00019     writeOut(buffer,filename1,r,100,0);
00020     r = readIn("test_allocation_0.txti_0","test_allocation_0.txt_0",100,1);
00021     writeOut(buffer,filename2,r,100,0);
00022
00023     return 0;
00024 }
```

Here is the call graph for this function:

4.75.2 Variable Documentation

4.75.2.1 char buffer[100]

Definition at line 7 of file test_fileIO.cc.

Referenced by main().

4.76 test_fileIO.cc

```
00001 #include "../../src/fileIO.cc"
00002 #include <stdlib.h>
00003 #include <stdio.h>
00004 #include <string.h>
00005 #include <cuda_runtime.h>
00006
00007 char buffer[100];
00008
00009 int main(int argc, char **argv){
00010     double2 *r;
00011     char *filename1, *filename2;
00012     filename1 = "test_allocation_0.txt";
00013     filename2 = "test_allocation_1.txt";
00014     r = (double2 *) malloc(sizeof(double2)*100);
00015     for (int i=0; i<99; ++i){
00016         r[i].x = i*1.0;
00017         r[i].y = 0.0;
00018     }
00019     writeOut(buffer,filename1,r,100,0);
00020     r = readIn("test_allocation_0.txti_0","test_allocation_0.txt_0",100,1);
00021     writeOut(buffer,filename2,r,100,0);
00022
00023     return 0;
00024 }
```

4.77 src/tracker.cc File Reference

```
#include "../include/tracker.h"
#include "../include/fileIO.h"
#include "../include/minions.h"
#include "../include/constants.h"
```

Include dependency graph for tracker.cc:

Functions

- double vortSepAvg (struct Vortex ∗vArray, struct Vortex centre, int length)
- int findOLMaxima (int ∗marker, double ∗Vopt, double radius, int xDim, double ∗x)
- int findVortex (int ∗marker, double2 ∗wfc, double radius, int xDim, double ∗x, int timestep)
- void olPos (int ∗marker, int2 ∗olLocation, int xDim)

  Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.
- int phaseTest (int2 vLoc, double2 ∗wfc, int xDim)
- void vortPos (int ∗marker, struct Vortex ∗vLocation, int xDim, double2 ∗wfc)

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

- void vortArrange (struct Vortex ∗vCoordsC, struct Vortex ∗vCoordsP, int length)
- struct Vortex vortCentre (struct Vortex ∗cArray, int length, int xDim)
- double vortAngle (struct Vortex ∗vortCoords, struct Vortex central, int numVort)
- double sigVOL (struct Vortex ∗vArr, int2 ∗opLatt, double ∗x, int numVort)

  Sigma of vortex lattice and optical lattice.

Variables

- char bufferT [1024]

4.77.1 Function Documentation

4.77.1.1 int findOLMaxima (int ∗ *marker,* double ∗ *Vopt,* double *radius,* int *xDim,* double ∗ *x*)

Definition at line 40 of file tracker.cc.

References vis::i, and maxValue().

```
00040                                                                                   {
00041     double gridValues[9];
00042     int2 mIndex[1024];
00043     int2 index;
00044     int i,j,found;
00045     found=0;
00046     for (i=1; i<xDim-1; ++i ){
00047         for(j=1; j<xDim-1;++j){
00048             if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00049                 gridValues[0] = Vopt[(i-1)*xDim + (j-1)];
00050                 gridValues[1] = Vopt[(i-1)*xDim + j];
00051                 gridValues[2] = Vopt[(i-1)*xDim + (j+1)];
00052                 gridValues[3] = Vopt[i*xDim + (j-1)];
00053                 gridValues[4] = Vopt[i*xDim + j];
00054                 gridValues[5] = Vopt[i*xDim + (j+1)];
00055                 gridValues[6] = Vopt[(i+1)*xDim + (j-1)];
00056                 gridValues[7] = Vopt[(i+1)*xDim + j];
00057                 gridValues[8] = Vopt[(i+1)*xDim + (j+1)];
00058                 if(fabs((gridValues[4]-maxValue(gridValues,9))/gridValues[4]) <= 1e-7){
00059                     //printf ("%d,%d\n",i,j);
00060                     (marker)[i*xDim + j] = 1;
00061                     index.x=i;
00062                     index.y=j;
00063                     mIndex[found] = index;
00064                     ++found;
00065                 }
00066             }
00067         }
00068     }
00069     return found;
00070 }
```

Here is the call graph for this function:

4.77.1.2 int findVortex (int ∗ *marker,* double2 ∗ *wfc,* double *radius,* int *xDim,* double ∗ *x,* int *timestep*)

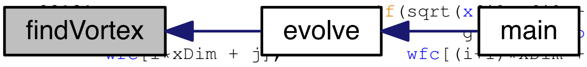
Definition at line 110 of file tracker.cc.

References complexDiv(), complexMag(), complexScale(), vis::i, PI, and y.

Referenced by evolve().

```
00110                                                                                               {
00111         double2 *g = (double2*) malloc(sizeof(double2)*4);
00112         double *phiDelta = (double*) malloc(sizeof(double)*4);
00113     int i,j,found;
00114     int cond_x, cond_y;
00115     cond_x = 0; cond_y = 0;
00116     found = 0;
```

```
00117     long rnd_value = 0;
00118     double sum = 0.0;
00119         for ( i=0; i < xDim-1; ++i ){
00120                 for( j=0; j < xDim-1; ++j ){
```



```
                         f(sqrt(x            x[j]*x[j]) < radius){
                                g         lexScale( complexDiv(
      xDim + j],             wfc[(i          j] ) ,     (complexMag(
      wfc[(i+1)*xDim + j]) / complexMag( wfc[i*xDim + j] )));
00123                               g[1] = complexScale( complexDiv(
      wfc[(i+1)*xDim + j],       wfc[(i+1)*xDim + (j+1)] ) ,     (complexMag(
      wfc[(i+1)*xDim + (j+1)]) / complexMag( wfc[(i+1)*xDim + j] )));
00124                               g[2] = complexScale( complexDiv(
      wfc[(i+1)*xDim + (j+1)], wfc[i*xDim + (j+1)] ) ,   (complexMag(
      wfc[i*xDim + (j+1)]) / complexMag( wfc[(i+1)*xDim + (j+1)] )));
00125                               g[3] = complexScale( complexDiv(
      wfc[i*xDim + (j+1)],       wfc[i*xDim + j] ) ,         (complexMag(
      wfc[i*xDim + j])      / complexMag( wfc[i*xDim + (j+1)] )));
00126
00127                 for (int k=0; k<4; ++k){
00128                     phiDelta[k] = atan2( g[k].y, g[k].x );
00129                     if(phiDelta[k] <= -PI){
00130                         phiDelta[k] += 2*PI;
00131                     }
00132                 }
00133                 sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00134                 rnd_value = lround(sum/(2*PI));
00135                                 if( sum >= 1.9*PI && cond_x <= 0 && cond_y <= 0){
00136                     marker[i*xDim + j] = rnd_value;
00137                     ++found;
00138                     sum = 0.0;
00139                     cond_x = 2; cond_y = 2;
00140                                 }
00141                 else if( sum <= -1.9*PI && cond_x <= 0 && cond_y <= 0 )  {
00142                     marker[i*xDim + j] = -rnd_value;
00143                     ++found;
00144                     sum = 0.0;
00145                     cond_x = 2; cond_y = 2;
00146
00147                 }
00148                 --cond_x;
00149                 --cond_y;
00150                     }
00151                 }
00152         }
00153     return found;
00154 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.77.1.3 void olPos (int ∗ *marker,* int2 ∗ *olLocation,* int *xDim*)

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

Definition at line 158 of file tracker.cc.

References vort::counter, vis::i, xDim, and y.

```
00158                                                          {
00159     int i,j;
00160     unsigned int counter=0;
00161     for(i=0; i<xDim; ++i){
00162         for(j=0; j<xDim; ++j){
00163             if((marker)[i*xDim + j] == 1){
00164                 (olLocation)[ counter ].x=i;
00165                 (olLocation)[ counter ].y=j;
00166                 ++counter;
00167             }
00168         }
00169     }
00170 }
```

4.77.1.4 int phaseTest (int2 *vLoc,* double2 ∗ *wfc,* int *xDim*)

Definition at line 172 of file tracker.cc.

References complexDiv(), complexMag(), complexScale(), PI, result, and y.

```
00172                                                   {
00173     int result = 0;
00174     double2 gridValues[4];
00175     double phiDelta[4];
00176     double sum=0.0;
00177     int i=vLoc.x, j=vLoc.y;
00178     gridValues[0] = complexScale( complexDiv(wfc[i*xDim + j],
      wfc[(i+1)*xDim + j]), (complexMag(wfc[(i+1)*xDim + j])/
      complexMag(wfc[i*xDim + j])));
00179         gridValues[1] = complexScale( complexDiv(wfc[(i+1)*
      xDim + j],wfc[(i+1)*xDim + (j+1)]), (complexMag(wfc[(i+1)*
      xDim + (j+1)])/complexMag(wfc[(i+1)*xDim + j])));
00180         gridValues[2] = complexScale( complexDiv(wfc[(i+1)*
      xDim + (j+1)],wfc[i*xDim + (j+1)]), (complexMag(wfc[i*
      xDim + (j+1)])/complexMag(wfc[(i+1)*xDim + (j+1)])));
00181         gridValues[3] = complexScale( complexDiv(wfc[i*
      xDim + (j+1)],wfc[i*xDim + j]), (complexMag(wfc[i*xDim + j])/
      complexMag(wfc[i*xDim + (j+1)])));
00182
00183     for (int k=0; k<4; ++k){
00184         phiDelta[k] = atan2(gridValues[k].y,gridValues[k].x);
00185                 if(phiDelta[k] <= -PI){
00186                     phiDelta[k] += 2*PI;
00187         }
00188     }
00189     sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
00190     if(sum >=1.8*PI){
00191         result = 1;
00192     }
00193     free(gridValues); free(phiDelta);
00194     return result;
00195 }
```

Here is the call graph for this function:

4.77.1.5 double sigVOL (struct Vortex ∗ *vArr,* int2 ∗ *opLatt,* double ∗ *x,* int *numVort*)

Sigma of vortex lattice and optical lattice.

Definition at line 274 of file tracker.cc.

References Vortex::coords, dx, and vis::i.

```
00274                                                                     {
00275     double sigma = 0.0;
00276     double dx = abs(x[1]-x[0]);
00277     for (int i=0; i<numVort; ++i){
00278         sigma += pow( abs( sqrt( (vArr[i].coords.x - opLatt[i].x)*(vArr[i].
      coords.x - opLatt[i].x) + (vArr[i].coords.y - opLatt[i].y)*(vArr[i].
      coords.y - opLatt[i].y) )*dx),2);
00279     }
00280     sigma /= numVort;
00281     return sigma;
00282 }
```

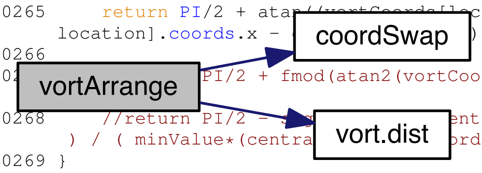
4.77.1.6 double vortAngle (struct Vortex ∗ *vortCoords,* struct Vortex *central,* int *numVort*)

Definition at line 255 of file tracker.cc.

References Vortex::coords, vis::i, minValue(), PI, and Vortex::sign.

Referenced by evolve().

```
00255                                                                           {
00256     int location;
00257     double sign=1.0;
00258     double minValue=2*512*512;//(pow(central.x - vortCoords[0].x,2) + pow(central.y -
       vortCoords[0].y,2));
00259     for (int i=0; i < numVort; ++i){
00260         if (minValue > (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.
      coords.y - vortCoords[i].coords.y,2)) && abs(central.coords.x - vortCoords[i].
      coords.x) > 1e-4 && abs(central.coords.y - vortCoords[i].coords.y) > 1e-4){
00261             minValue = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.
coords.y - vortCoords[i].coords.y,2));
            location = i;
        }
    }
    return PI/2 + atan((vortCoords[location].coords.y - central.coords.y) / (vortCoords[
location].coords.x - (                  ));

                PI/2 + fmod(atan2(vortCoords[location].y-central.y, vortCoords[location].x - central.x),

    //return PI/2 - s                   entral.x - vortCoords[location].x)*(central.x - vortCoords[location].x)
 ) / ( minValue*(centra                 ords[location].x) ) );
}
```



Here is the call graph for this function:

Here is the caller graph for this function:

4.77.1.7 void vortArrange (struct Vortex ∗ *vCoordsC,* struct Vortex ∗ *vCoordsP,* int *length*)

Definition at line 217 of file tracker.cc.

References Vortex::coords, coordSwap(), vort::dist(), and vis::i.

Referenced by evolve().

```
                                                                                          {
    int dist, dist_t;
    int i, j, index;
    for ( i = 0; i < length; ++i ){
        dist = 0x7FFFFFFF; //arbitrary big value
        index = i;
        for ( j = i; j < length ; ++j){
            dist_t = ( (vCoordsP[i].coords.x - vCoordsC[j].coords.x)*(vCoordsP[i].
coords.x - vCoordsC[j].coords.x) + (vCoordsP[i].coords.y - vCoordsC[j].
coords.y)*(vCoordsP[i].coords.y - vCoordsC[j].coords.y) );
            if(dist > dist_t ){
                dist = dist_t;
                index = j;
            }
        }
        coordSwap(vCoordsC,index,i);
    }
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.77.1.8 struct Vortex vortCentre (struct Vortex ∗ *cArray,* int *length,* int *xDim*)

Definition at line 236 of file tracker.cc.

References Vortex::coords, vort::counter, and vis::i.

Referenced by evolve().

```
                                                                                {
    int i, j, counter=0;
    int valX, valY;
    double valueTest, value = 0.0;
    valX = (cArray)[0].coords.x - ((xDim/2)-1);
    valY = (cArray)[0].coords.y - ((xDim/2)-1);
    value = sqrt( valX*valX + valY*valY );//Calcs the sqrt(x^2+y^2) from central position. try to minimise
 this value
    for ( i=1; i<length; ++i ){
        valX = (cArray)[i].coords.x - ((xDim/2)-1);
        valY = (cArray)[i].coords.y - ((xDim/2)-1);
        valueTest = sqrt(valX*valX + valY*valY);
        if(value > valueTest){
            value = valueTest;
            counter = i;
        }
    }
    return (cArray)[counter];
}
```

Here is the caller graph for this function:

4.77.1.9 void vortPos (int ∗ *marker,* struct Vortex ∗ *vLocation,* int *xDim,* double2 ∗ *wfc*)

Accepts matrix of vortex locations as argument, returns array of x,y coordinates of locations and first encountered vortex angle.

Definition at line 198 of file tracker.cc.

References Vortex::coords, vort::counter, vis::i, Vortex::sign, Vortex::wind, and xDim.

Referenced by evolve().

```
00198                                                                          {
00199     int i,j;
00200     unsigned int counter=0;
00201     for(i=0; i<xDim; ++i){
00202         for(j=0; j<xDim; ++j){
00203             if( abs((marker)[i*xDim + j]) >= 1){
00204                 (vLocation)[ counter ].coords.x=i;
00205                 (vLocation)[ counter ].coords.y=j;
00206                 (vLocation)[ counter ].sign = ( signbit(abs(marker[i*xDim + j])) == 0 ) ? 1 : -1;
00207                 (vLocation)[ counter ].wind = abs(marker[i*xDim + j]);
00208                 ++counter;
00209             }
00210         }
00211     }
00212 }
```

Here is the caller graph for this function:

4.77.1.10 double vortSepAvg (struct Vortex ∗ *vArray,* struct Vortex *centre,* int *length*)

Definition at line 26 of file tracker.cc.

References Vortex::coords, and result.

Referenced by evolve(), and optLatSetup().

```
00026                                                                         {
00027     double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00028     double min = 0.0;
00029     int index=0;
00030     min = sqrt( pow(centre.coords.x - vArray[0].coords.x,2) + pow(centre.
      coords.y - vArray[0].coords.y,2));
00031     for (int j=1; j<length; ++j){
00032         if(min > sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.
      coords.y - vArray[j].coords.y,2)) && sqrt( pow(centre.coords.x - vArray[j].
      coords.x,2) + pow(centre.coords.y - vArray[j].coords.y,2)) > 1e-7){
00033             min = sqrt(pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.
      coords.y - vArray[j].coords.y,2));
00034             index = j;
00035         }
00036     }
00037     return min;
00038 }
```

Here is the caller graph for this function:

4.77.2 Variable Documentation

4.77.2.1 char bufferT[1024]

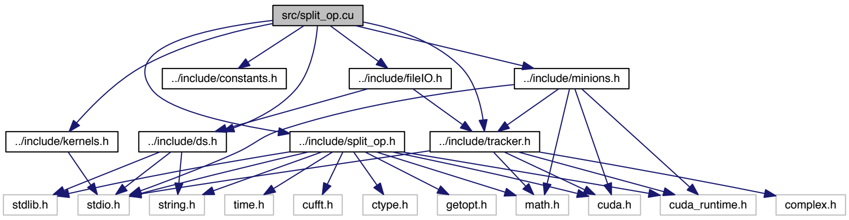
Definition at line 24 of file tracker.cc.

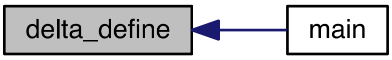
4.78 tracker.cc

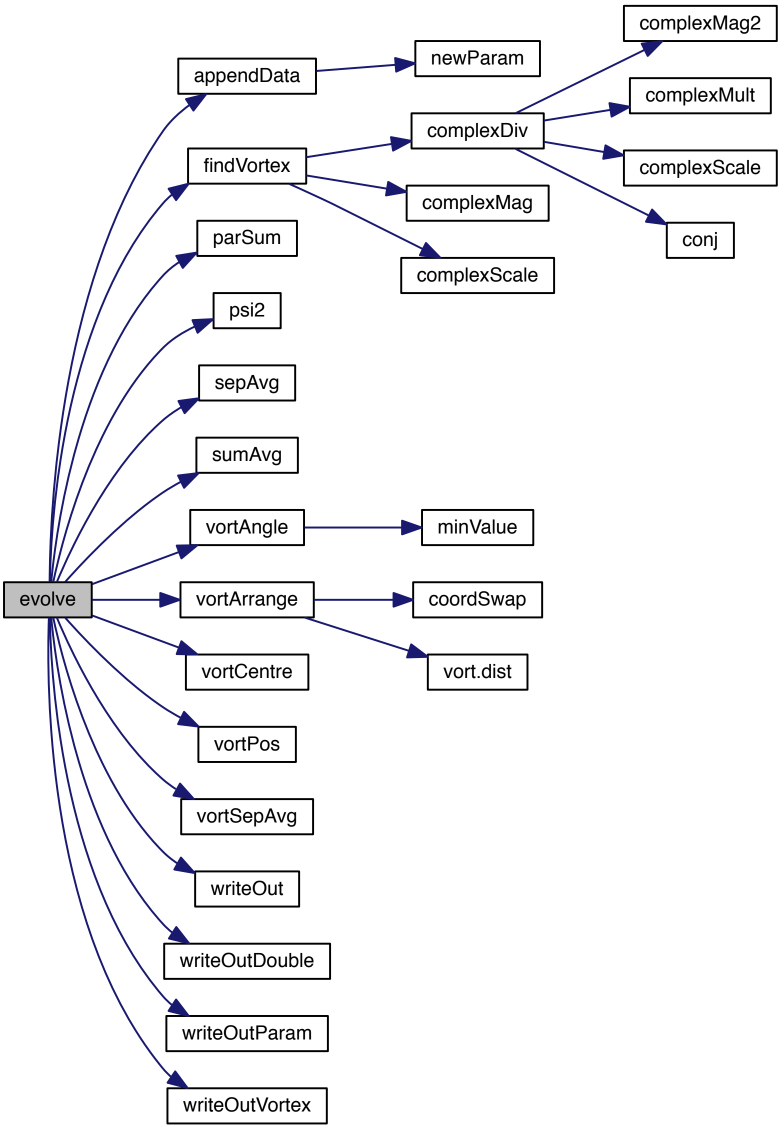
```
00001 /*
```

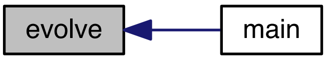
```
* tracker.cc - GPUE: Split Operator based GPU solver for Nonlinear
* Schrodinger Equation, Copyright (C) 2012, Lee J. O'Riordan, Tadhg
* Morgan, Neil Crowley.

* This library is free software; you can redistribute it and/or modify
* it under the terms of the GNU Lesser General Public License as
* published by the Free Software Foundation; either version 2.1 of the
* License, or (at your option) any later version. This library is
* distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or
* FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public
* License for more details. You should have received a copy of the GNU
* Lesser General Public License along with this library; if not, write
* to the Free Software Foundation, Inc., 59 Temple Place, Suite 330,
* Boston, MA 02111-1307 USA
*/

#include "../include/tracker.h"
#include "../include/fileIO.h"
#include "../include/minions.h"
#include "../include/constants.h"

char bufferT[1024];

double vortSepAvg(struct Vortex *vArray, struct Vortex centre, int length){
    double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
    double min = 0.0;
    int index=0;
    min = sqrt( pow(centre.coords.x - vArray[0].coords.x,2) + pow(centre.coords.y - vArray[0].coords.y,2));
    for (int j=1; j<length; ++j){
        if(min > sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.coords.y - vArray[j].coords.y,2)) && sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.coords.y - vArray[j].coords.y,2)) > 1e-7){
            min = sqrt(pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.coords.y - vArray[j].coords.y,2));
            index = j;
        }
    }
    return min;
}

int findOLMaxima(int *marker, double *Vopt, double radius, int xDim, double* x){
    double gridValues[9];
    int2 mIndex[1024];
    int2 index;
    int i,j,found;
    found=0;
    for (i=1; i<xDim-1; ++i ){
        for(j=1; j<xDim-1;++j){
            if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
                gridValues[0] = Vopt[(i-1)*xDim + (j-1)];
                gridValues[1] = Vopt[(i-1)*xDim + j];
                gridValues[2] = Vopt[(i-1)*xDim + (j+1)];
                gridValues[3] = Vopt[i*xDim + (j-1)];
                gridValues[4] = Vopt[i*xDim + j];
                gridValues[5] = Vopt[i*xDim + (j+1)];
                gridValues[6] = Vopt[(i+1)*xDim + (j-1)];
                gridValues[7] = Vopt[(i+1)*xDim + j];
                gridValues[8] = Vopt[(i+1)*xDim + (j+1)];
                if(fabs((gridValues[4]-maxValue(gridValues,9))/gridValues[4]) <= 1e-7){
                    //printf ("%d,%d\n",i,j);
                    (marker)[i*xDim + j] = 1;
                    index.x=i;
                    index.y=j;
                    mIndex[found] = index;
                    ++found;
                }
            }
        }
    }
    return found;
}

#ifdef VORT_MIN
int findVortex(int *marker, double2* wfc, double radius, int xDim, double* x, int timestep){
    double gridValues[9];
    int2 vIndex[1024];
    int2 index;
    int i,j,found;
    found=0;
//  #pragma omp parallel for private(j)
    for (i=1; i<xDim-1; ++i ){
        for(j=1; j<xDim-1;++j){
            if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
```

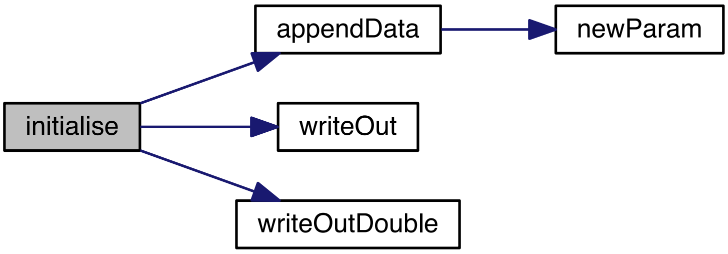
```
        gridValues[0] = psi2(wfc[(i-1)*xDim + (j-1)]);
        gridValues[1] = psi2(wfc[(i-1)*xDim + j]);
        gridValues[2] = psi2(wfc[(i-1)*xDim + (j+1)]);
        gridValues[3] = psi2(wfc[(i)*xDim + (j-1)]);
        gridValues[4] = psi2(wfc[(i)*xDim + j]);
        gridValues[5] = psi2(wfc[(i)*xDim + (j+1)]);
        gridValues[6] = psi2(wfc[(i+1)*xDim + (j-1)]);
        gridValues[7] = psi2(wfc[(i+1)*xDim + j]);
        gridValues[8] = psi2(wfc[(i+1)*xDim + (j+1)]);
        if(fabs((gridValues[4]-minValue(gridValues,9))/gridValues[4]) < 1e-7){
          //printf ("%d,%d\n",i,j);
          (marker)[i*xDim + j] = 1;
          index.x=i;
          index.y=j;
          vIndex[found] = index;
          found++;
        }
      }
    }
  }
  return found;
}
#else
/*
 * Phase winding method to determine vortex positions.
 *
 **/
int findVortex(int *marker, double2* wfc, double radius, int xDim, double *x, int timestep){
    double2 *g = (double2*) malloc(sizeof(double2)*4);
    double *phiDelta = (double*) malloc(sizeof(double)*4);
  int i,j,found;
  int cond_x, cond_y;
  cond_x = 0; cond_y = 0;
  found = 0;
  long rnd_value = 0;
  double sum = 0.0;
    for ( i=0; i < xDim-1; ++i ){
        for( j=0; j < xDim-1; ++j ){
            if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
                g[0] = complexScale( complexDiv( wfc[i*xDim + j], wfc[(i+1)*xDim + j] ) ,    (complexMag( wfc[(i+1)*xDim + j]) / complexMag( wfc[i*xDim + j] )));
                g[1] = complexScale( complexDiv( wfc[(i+1)*xDim + j], wfc[(i+1)*xDim + (j+1)] ) ,    (complexMag( wfc[(i+1)*xDim + (j+1)]) / complexMag( wfc[(i+1)*xDim + j] )));
                g[2] = complexScale( complexDiv( wfc[(i+1)*xDim + (j+1)], wfc[i*xDim + (j+1)] ) ,      (complexMag( wfc[i*xDim + (j+1)]) / complexMag( wfc[(i+1)*xDim + (j+1)] )));
                g[3] = complexScale( complexDiv( wfc[i*xDim + (j+1)], wfc[i*xDim + j] ) ,        (complexMag( wfc[i*xDim + j])      / complexMag( wfc[i*xDim + (j+1)] )));

        for (int k=0; k<4; ++k){
          phiDelta[k] = atan2( g[k].y, g[k].x );
          if(phiDelta[k] <= -PI){
            phiDelta[k] += 2*PI;
          }
        }
        sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
        rnd_value = lround(sum/(2*PI));
                if( sum >= 1.9*PI && cond_x <= 0 && cond_y <= 0){
          marker[i*xDim + j] = rnd_value;
          ++found;
          sum = 0.0;
          cond_x = 2; cond_y = 2;
                }
        else if( sum <= -1.9*PI && cond_x <= 0 && cond_y <= 0 )  {
          marker[i*xDim + j] = -rnd_value;
          ++found;
          sum = 0.0;
          cond_x = 2; cond_y = 2;

        }
        --cond_x;
        --cond_y;
            }
        }
    }
  return found;
}
#endif

void olPos(int *marker, int2 *olLocation, int xDim){
  int i,j;
  unsigned int counter=0;
  for(i=0; i<xDim; ++i){
    for(j=0; j<xDim; ++j){
            if((marker)[i*xDim + j] == 1){
                (olLocation)[ counter ].x=i;
                (olLocation)[ counter ].y=j;
                ++counter;
            }
        }
    }
}

int phaseTest(int2 vLoc, double2* wfc, int xDim){
    int result = 0;
    double2 gridValues[4];
    double phiDelta[4];
    double sum=0.0;
    int i=vLoc.x, j=vLoc.y;
    gridValues[0] = complexScale( complexDiv(wfc[i*xDim + j],wfc[(i+1)*xDim + j]), ( complexMag(wfc[(i+1)*xDim + j])/complexMag(wfc[i*xDim + j])));
        gridValues[1] = complexScale( complexDiv(wfc[(i+1)*xDim + j],wfc[(i+1)*xDim + (j+1)]), (complexMag(wfc[(i+1)*xDim + (j+1)])/complexMag(wfc[(i+1)*xDim + j])));
        gridValues[2] = complexScale( complexDiv(wfc[(i+1)*xDim + (j+1)],wfc[i*xDim + (j+1)]), (complexMag(wfc[i*xDim + (j+1)])/complexMag(wfc[(i+1)*xDim + (j+1)])));
        gridValues[3] = complexScale( complexDiv(wfc[i*xDim + (j+1)],wfc[i*xDim + j]), (complexMag(wfc[i*xDim + j])/complexMag(wfc[i*xDim + (j+1)])));

    for (int k=0; k<4; ++k){
        phiDelta[k] = atan2(gridValues[k].y,gridValues[k].x);
                if(phiDelta[k] <= -PI){
                    phiDelta[k] += 2*PI;
        }
    }
    sum = phiDelta[0] + phiDelta[1] + phiDelta[2] + phiDelta[3];
    if(sum >=1.8*PI){
        result = 1;
    }
    free(gridValues); free(phiDelta);
    return result;
}

void vortPos(int *marker, struct Vortex *vLocation, int xDim, double2 *wfc){
    int i,j;
    unsigned int counter=0;
    for(i=0; i<xDim; ++i){
        for(j=0; j<xDim; ++j){
            if( abs((marker)[i*xDim + j]) >= 1){
                (vLocation)[ counter ].coords.x=i;
                (vLocation)[ counter ].coords.y=j;
                (vLocation)[ counter ].sign = ( signbit(abs(marker[i*xDim + j])) == 0 ) ? 1 : -1;
                (vLocation)[ counter ].wind = abs(marker[i*xDim + j]);
                ++counter;
            }
        }
    }
}

/*
 * Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions
 */
void vortArrange(struct Vortex *vCoordsC, struct Vortex *vCoordsP, int length){
    int dist, dist_t;
    int i, j, index;
    for ( i = 0; i < length; ++i ){
        dist = 0x7FFFFFFF; //arbitrary big value
        index = i;
        for ( j = i; j < length ; ++j){
            dist_t = ( (vCoordsP[i].coords.x - vCoordsC[j].coords.x)*(vCoordsP[i].coords.x - vCoordsC[j].coords.x) + (vCoordsP[i].coords.y - vCoordsC[j].coords.y)*(vCoordsP[i].coords.y - vCoordsC[j].coords.y) );
            if(dist > dist_t ){
                dist = dist_t;
                index = j;
            }
        }
        coordSwap(vCoordsC,index,i);
    }
}

/* Determines the coords of the vortex closest to the central position. Useful for centering the optical lattice over v. lattice*
*/
struct Vortex vortCentre(struct Vortex *cArray, int length, int xDim){
    int i, j, counter=0;
    int valX, valY;
    double valueTest, value = 0.0;
    valX = (cArray)[0].coords.x - ((xDim/2)-1);
    valY = (cArray)[0].coords.y - ((xDim/2)-1);
    value = sqrt( valX*valX + valY*valY );//Calcs the sqrt(x^2+y^2) from central position. try to minimise this value
    for ( i=1; i<length; ++i ){
        valX = (cArray)[i].coords.x - ((xDim/2)-1);
        valY = (cArray)[i].coords.y - ((xDim/2)-1);
        valueTest = sqrt(valX*valX + valY*valY);
        if(value > valueTest){
            value = valueTest;
            counter = i;
        }
    }
    return (cArray)[counter];
}

double vortAngle(struct Vortex *vortCoords, struct Vortex central, int numVort){
    int location;
    double sign=1.0;
    double minValue=2*512*512;//(pow(central.x - vortCoords[0].x,2) + pow(central.y - vortCoords[0].y,2));
    for (int i=0; i < numVort; ++i){
        if (minValue > (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.coords.y - vortCoords[i].coords.y,2)) && abs(central.coords.x - vortCoords[i].coords.x) > 1e-4 && abs(central.coords.y - vortCoords[i].coords.y) > 1e-4){
            minValue = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.coords.y - vortCoords[i].coords.y,2));
            location = i;
        }
    }
    return PI/2 + atan((vortCoords[location].coords.y - central.coords.y) / (vortCoords[location].coords.x - central.coords.x));

    //return PI/2 + fmod(atan2(vortCoords[location].y-central.y, vortCoords[location].x - central.x), PI/3);
    //return PI/2 - sign*acos( ( (central.x - vortCoords[location].x)*(central.x - vortCoords[location].x) ) / ( minValue*(central.x - vortCoords[location].x) ) );
}

double sigVOL(struct Vortex *vArr, int2 *opLatt, double *x, int numVort){
    double sigma = 0.0;
    double dx = abs(x[1]-x[0]);
    for (int i=0; i<numVort; ++i){
        sigma += pow( abs( sqrt( (vArr[i].coords.x - opLatt[i].x)*(vArr[i].coords.x - opLatt[i].x) + (vArr[i].coords.y - opLatt[i].y)*(vArr[i].coords.y - opLatt[i].y) )*dx),2);
    }
    sigma /= numVort;
    return sigma;
}
```

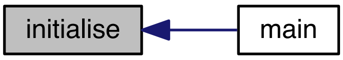

src/split_op.cu
../include/constants.h
../include/fileIO.h
../include/minions.h
../include/kernels.h
../include/ds.h
../include/split_op.h
../include/tracker.h
stdlib.h
stdio.h
string.h
time.h
cufft.h
ctype.h
getopt.h
math.h
cuda.h
cuda_runtime.h
complex.h

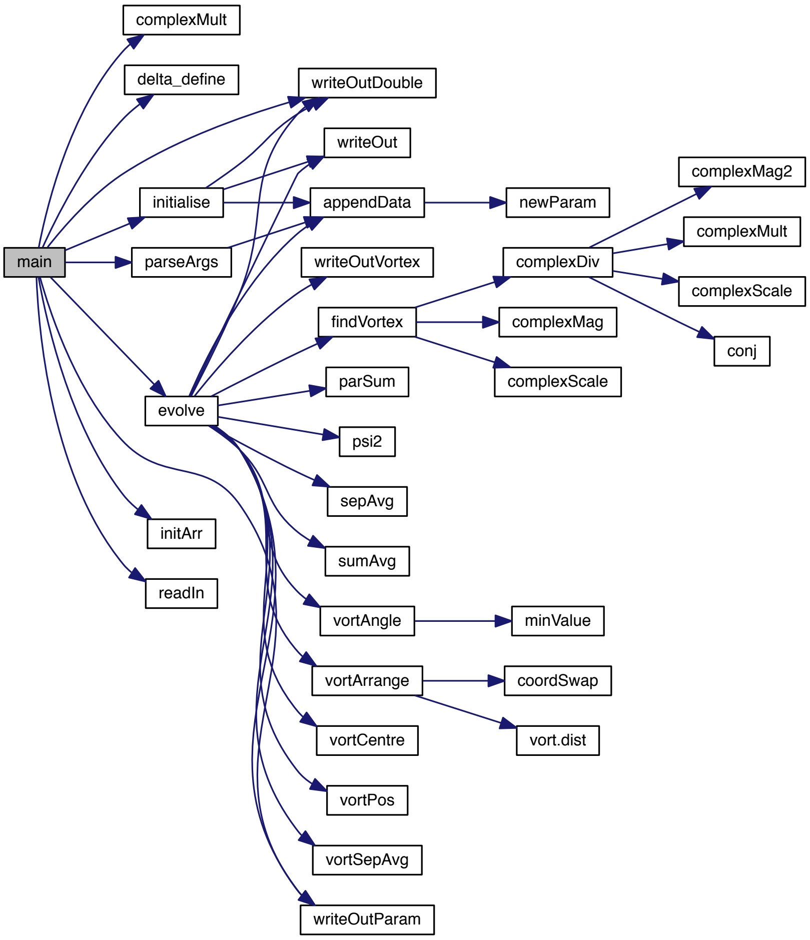

delta_define
main

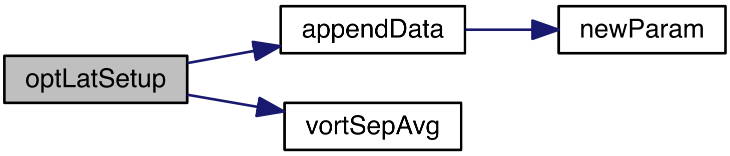

evolve
appendData
findVortex
parSum
psi2
sepAvg
sumAvg
vortAngle
vortArrange
vortCentre
vortPos
vortSepAvg
writeOut
writeOutDouble
writeOutParam
writeOutVortex
newParam
complexDiv
complexMag
complexScale
minValue
coordSwap
vort.dist
complexMag2
complexMult
complexScale
conj

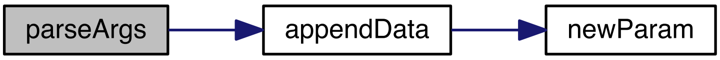

evolve
main

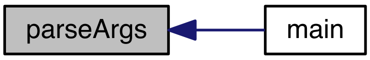

appendData
newParam
initialise
writeOut
writeOutDouble

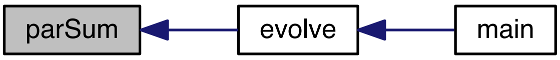

initialise
main

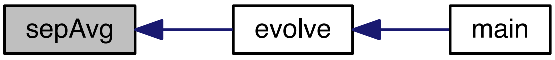

complexMult
delta_define
writeOutDouble
writeOut
initialise
appendData
newParam
complexMag2
complexMult
main
parseArgs
writeOutVortex
complexDiv
complexScale
findVortex
complexMag
conj
parSum
complexScale
evolve
psi2
sepAvg
initArr
sumAvg
readIn
vortAngle
minValue
vortArrange
coordSwap
vortCentre
vort.dist
vortPos
vortSepAvg
writeOutParam

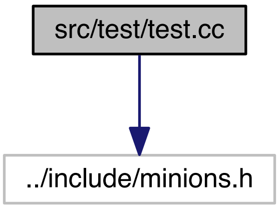

appendData
newParam
optLatSetup
vortSepAvg

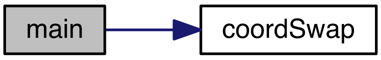

parseArgs
appendData
newParam

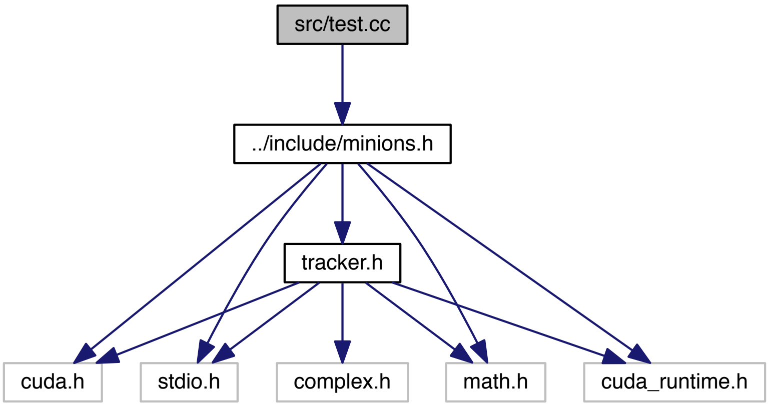

parseArgs
main

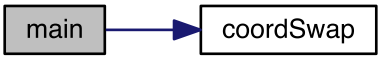

parSum
evolve
main

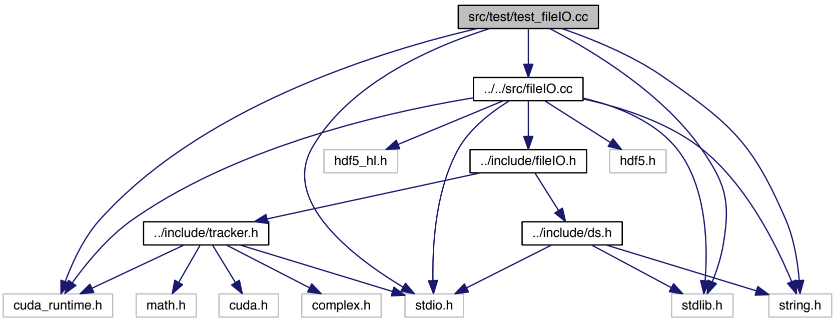

sepAvg
evolve
main

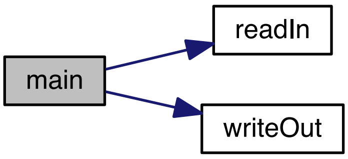

src/test/test.cc
../include/minions.h


main
coordSwap


src/test.cc
../include/minions.h
tracker.h
cuda.h
stdio.h
complex.h
math.h
cuda_runtime.h


main
coordSwap


src/test/test_fileIO.cc
../../src/fileIO.cc
hdf5_hl.h
../include/fileIO.h
hdf5.h
../include/tracker.h
../include/ds.h
cuda_runtime.h
math.h
cuda.h
complex.h
stdio.h
stdlib.h
string.h


main
readIn
writeOut