GP(U E): GPU Gross-Pitaevskii Equation

0

Generated by Doxygen 1.8.6

Mon Jul 6 2015 17:26:33

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 BEC2D Namespace Reference

Classes

- class Wavefunction

Functions

- class BEC2D::Wavefunction Wavefunction ()

4.1.1 Function Documentation

4.1.1.1 class BEC2D::Wavefunction BEC2D::Wavefunction ( )

Definition at line 54 of file wavefunction.cu.

```
00054                                     {
00055
00056     }
```

4.2 FileIO Namespace Reference

Check source file for information on functions.

Functions

- double2 ∗ readIn (char ∗, char ∗, int, int)
- int readState (char ∗)
- 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 writeOutParam (char ∗, Array, char ∗)
- void writeOutVortex (char ∗, char ∗, struct Tracker::Vortex ∗, int, int)

4.2.1 Detailed Description

Check source file for information on functions.

4.2.2 Function Documentation

4.2.2.1 double2 ∗ FileIO::readIn (char ∗ *fileR,* char ∗ *fileI,* int *xDim,* int *yDim*)

Definition at line 45 of file fileIO.cc.

References FILE, vis::i, and yDim.

Referenced by main().

```
                                                                 {
        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;
    }
```

Here is the caller graph for this function:

4.2.2.2 int FileIO::readState (char ∗ *name*)

Definition at line 156 of file fileIO.cc.

References FILE.

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

4.2.2.3 void FileIO::writeOut (char ∗ *buffer,* char ∗ *file,* double2 ∗ *data,* int *length,* int *step*)

Definition at line 84 of file fileIO.cc.

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

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

```
                                                                              {
        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);
    }
```

Here is the caller graph for this function:

4.2.2.4 void FileIO::writeOutDouble (char ∗ *buffer,* char ∗ *file,* double ∗ *data,* int *length,* int *step*)

Definition at line 103 of file fileIO.cc.

References FILE, and vis::i.

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

```
                                                                                    {
        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);
    }
```

Here is the caller graph for this function:

4.2.2.5 void FileIO::writeOutInt (char ∗ *buffer,* char ∗ *file,* int ∗ *data,* int *length,* int *step*)

Definition at line 116 of file fileIO.cc.

References FILE, and vis::i.

```
                                                                               {
        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);
    }
```

4.2.2.6 void FileIO::writeOutInt2 (char ∗ *buffer,* char ∗ *file,* int2 ∗ *data,* int *length,* int *step*)

Definition at line 129 of file fileIO.cc.

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

```
                                                                                 {
        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);
    }
```

4.2.2.7 void FileIO::writeOutParam (char ∗ *buffer,* Array *arr,* char ∗ *file*)

Definition at line 69 of file fileIO.cc.

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

Referenced by evolve(), and main().

```
                                                            {
        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);
    }
```

Here is the caller graph for this function:

4.2.2.8 void FileIO::writeOutVortex (char ∗ *buffer,* char ∗ *file,* struct Tracker::Vortex ∗ *data,* int *length,* int *step*)

Definition at line 142 of file fileIO.cc.

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

Referenced by evolve().

```
                                                                                                        {
        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);
    }
```

Here is the caller graph for this function:

4.3 hist3d Namespace Reference

Functions

- def plot_hist_pcolor
- def plot_xyz_histogram

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple dt = (c.getfloat('Params','dt'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple incr = int(c.getfloat('Params','print_out'))
- int num_vort = 0
- tuple sep = (c.getfloat('Params','dx'))
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple yMax = (c.getfloat('Params','yMax'))

4.3.1 Detailed Description

4.3.2 Function Documentation

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

Definition at line 87 of file hist3d.py.

Referenced by vis.overlap().

```
00087
00088 def plot_hist_pcolor(start,fin,incr, barcolor):
00089     fig = plt.figure()
00090
00091     data =[]
00092     for i in range(start, fin, incr):
00093         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',' )
00094         datatmp=[]
00095         count=0
00096
00097         for i1 in range(0,v_arr.size/2):
00098             for i2 in range(i1,v_arr.size/2):
00099                 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 )
00100                 datatmp.append( m_tmp )
00101                 count = count + 1
00102         hist=np.histogram(datatmp,bins=np.arange(0.0,240.0,0.1))
00103         data.append(hist[:][0])
00104
00105       #  print data
00106         ax = fig.add_subplot(111)
00107         ax.imshow(data)
00108     plt.gca().invert_yaxis()
00109         ax.set_aspect('auto')
00110 #         plt.jet()
00111     fig.savefig("HIST_PCOLOR.pdf")
00112
00113 #plot_xyz_histogram(0,100000,100,'b')
00114 #plot_hist_pcolor(0,100000,100,'b')
00115
```

Here is the caller graph for this function:

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

Definition at line 57 of file hist3d.py.

```
00057
00058 def plot_xyz_histogram(start,fin,incr, barcolor):
00059     fig = plt.figure()
00060     ax = Axes3D(fig)
00061     data =[]
00062     for i in range(start, fin, incr):
00063         v_arr=genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',' )
00064         datatmp=[]
00065         count=0
00066
00067         for i1 in range(0,v_arr.size/2):
00068             for i2 in range(i1,v_arr.size/2):
00069                 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 ))
00070                 count = count + 1
00071         hist=np.histogram(datatmp,bins=np.arange(1.0,m.sqrt(xDim**2 + yDim**2),1.0))
00072         data.append(hist[:][0])
00073     """ Takes in a matrix (see structure above) and generate a pseudo-3D histogram by overlaying close,
       semitransparent bars. """
00074     for time, occurrence in zip(range(len(data)), data):
00075         dist = range(len(occurrence))
00076         barband = range(-45, 45, 5)
00077         #for modifier in barband:
00078         ax.bar(dist, occurrence, zs=time, zdir='y', color=np.random.rand(3,1), alpha=0.8)
00079             #ax.bar(current, occurrence, zs=duration+(float(modifier)/100), zdir='y',
       color=np.random.rand(3,1), alpha=0.6)
00080
00081     ax.set_xlabel('Dist')
00082     ax.set_ylabel('Time')
00083     ax.set_zlabel('Occurrances')
00084
00085     plt.savefig("HIST_N.pdf")
00086     plt.show()
```

4.3.3 Variable Documentation

4.3.3.1 tuple hist3d.c = ConfigParser.ConfigParser()

Definition at line 41 of file hist3d.py.

Referenced by Minions.complexDiv(), and Minions.conj().

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

Definition at line 51 of file hist3d.py.

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

Definition at line 50 of file hist3d.py.

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

Definition at line 47 of file hist3d.py.

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

Definition at line 46 of file hist3d.py.

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

Definition at line 48 of file hist3d.py.

4.3.3.7 int hist3d.num_vort = 0

Definition at line 54 of file hist3d.py.

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

Definition at line 49 of file hist3d.py.

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

Definition at line 44 of file hist3d.py.

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

Definition at line 52 of file hist3d.py.

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

Definition at line 45 of file hist3d.py.

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

Definition at line 53 of file hist3d.py.

4.4 hist_it Namespace Reference

4.4.1 Detailed Description

4.5 Minions Namespace Reference

Functions

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

    id magic hackery
- double maxValue (double ∗, int)
- double minValue (double ∗, int)
- double psi2 (double2)
- double sumAvg (double ∗in, int len)

4.5.1 Function Documentation

4.5.1.1 double2 Minions::complexDiv (double2 *num,* double2 *den*)

Definition at line 118 of file minions.cc.

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

Referenced by Tracker::findVortex(), and Tracker::phaseTest().

```
00118                                                   {
00119         double2 c = conj(den);
00120         return complexScale(complexMult(num,c),(1.0/
    complexMag2(den)));
00121     }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.5.1.2 double Minions::complexMag (double2 *in*)

Definition at line 90 of file minions.cc.

Referenced by Tracker::findVortex(), and Tracker::phaseTest().

```
00090                                    {
00091         return sqrt(in.x*in.x + in.y*in.y);
00092     }
```

Here is the caller graph for this function:

4.5.1.3 double Minions::complexMag2 (double2 *in*)

Definition at line 94 of file minions.cc.

Referenced by complexDiv().

```
00094                                     {
00095         return in.x*in.x + in.y*in.y;
00096     }
```

Here is the caller graph for this function:

4.5.1.4 double2 Minions::complexMult (double2 *in1*, double2 *in2*)

Definition at line 98 of file minions.cc.

References result.

Referenced by complexDiv().

```
                                                {
        double2 result;
        result.x = (in1.x*in2.x - in1.y*in2.y);
        result.y = (in1.x*in2.y + in1.y*in2.x);
        return result;
    }
```

Here is the caller graph for this function:

4.5.1.5 double2 Minions::complexScale (double2 *comp*, double *scale*)

Definition at line 105 of file minions.cc.

References result.

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

```
                                                    {
        double2 result;
        result.x = comp.x*scale;
        result.y = comp.y*scale;
        return result;
    }
```

Here is the caller graph for this function:

4.5.1.6 double2 Minions::conj (double2 *c*)

Definition at line 112 of file minions.cc.

References hist3d::c, and result.

Referenced by complexDiv().

```
                            {
        double2 result = c;
        result.y = -result.y;
        return result;
    }
```

Here is the caller graph for this function:

4.5.1.7 void Minions::coordSwap (struct Tracker::Vortex * *vCoords*, int *src*, int *dest*)

Definition at line 84 of file minions.cc.

Referenced by Tracker::vortArrange().

```
                                                                        {
        struct Tracker::Vortex d = vCoords[dest];
        vCoords[dest] = vCoords[src];
        vCoords[src] = d;
    }
```

Here is the caller graph for this function:

4.5.1.8 double Minions::fInvSqRt (double *in*)

id magic hackery

Definition at line 69 of file minions.cc.

References in(), and l.

```
                        {
        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;
    }
```

Here is the call graph for this function:

4.5.1.9 double Minions::maxValue (double ∗ *grid,* int *len*)

Definition at line 41 of file minions.cc.

References vis::i.

Referenced by Tracker::findOLMaxima().

```
                                    {
        double max = grid[0];
        for (unsigned int i=1;i<len-1;++i){
            if(max<grid[i]){
                max=grid[i];
            }
        }
        return max;
    }
```

Here is the caller graph for this function:

4.5.1.10 double Minions::minValue (double ∗ *grid,* int *len*)

Definition at line 51 of file minions.cc.

References vis::i.

```
                                    {
        double min = grid[0];
        for (unsigned int i=1;i<len-1;++i){
            if(min>grid[i])
                min=grid[i];
        }
        return min;
    }
```

4.5.1.11 double Minions::psi2 (double2 *in*)

Definition at line 37 of file minions.cc.

Referenced by evolve().

```
                        {
        return in.x*in.x + in.y*in.y;
    }
```

Here is the caller graph for this function:

4.5.1.12 double Minions::sumAvg (double ∗ *in,* int *len*)

Definition at line 60 of file minions.cc.

References vis::i.

Referenced by evolve().

```
                                             {
          double avg = 0.0;

          for (unsigned int i=0; i<len; ++i){
              avg += in[i];
          }
          return avg/len;
      }
```

Here is the caller graph for this function:

4.6 observables Namespace Reference

Functions

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

  Kinetic energy spectrum = kinertrum.
- def kinertrum_loop

Variables

- tuple c = ConfigParser.ConfigParser()
- tuple data = numpy.ndarray(shape=(xDim,yDim))
- tuple dkx = (c.getfloat('Params','dpx'))
- tuple dky = (c.getfloat('Params','dpy'))
- tuple dt = (c.getfloat('Params','dt'))
- tuple dx = (c.getfloat('Params','dx'))
- tuple dy = (c.getfloat('Params','dy'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple g = (0.5∗N)
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- float HBAR = 1.05457148e-34
- float hbar = 1.05457e-34
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple K = np.array(open('K_0').read().splitlines(),dtype='f8')
- tuple k_mag = np.sqrt(kx∗∗2 + ky∗∗2)
- tuple km_mag = np.sqrt(kxm∗∗2 + kym∗∗2)
- tuple kMax = max(max(k_mag))
- tuple kx = np.reshape(np.array([np.linspace(0, (xDim/2-1)∗dkx, xDim/2), np.linspace((-xDim/2-1)∗dkx, -dkx, xDim/2)]), (xDim,1))

- tuple ky = np.reshape(np.array([np.linspace(0, (yDim/2-1)∗dky, yDim/2), np.linspace((-yDim/2-1)∗dky, -dky, yDim/2)]), (yDim,1))
- float m = 1.4431607e-25
- tuple mass = (c.getfloat('Params','Mass'))
- tuple N = int(c.getfloat('Params','atoms'))
- tuple num_vort = int(c.getfloat('Params','Num_vort'))
- tuple omega = (c.getfloat('Params','omega'))
- tuple omegaX = (c.getfloat('Params','omegaX'))
- tuple omegaZ = (c.getfloat('Params','omegaZ'))
- float PI = 3.141592653589793
- tuple V = np.array(open('V_0').read().splitlines(),dtype='f8')
- tuple x = np.asarray(open('x_0').read().splitlines(),dtype='f8')
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple xPy = np.array(open('xPy_0').read().splitlines(),dtype='f8')
- tuple y = np.asarray(open('y_0').read().splitlines(),dtype='f8')
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple yMax = (c.getfloat('Params','yMax'))
- tuple yPx = np.array(open('yPx_0').read().splitlines(),dtype='f8')

4.6.1 Detailed Description

4.6.2 Function Documentation

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

Definition at line 293 of file observables.py.

Referenced by expec_val_().

```
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()
```

Here is the caller graph for this function:

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

Definition at line 194 of file observables.py.

References kinertrum().

Referenced by expec_val_().

```
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

            kinertrum(np.reshape(a,(xDim,yDim)),dx,i,0)
            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()
```

Here is the call graph for this function:

Here is the caller graph for this function:

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

Definition at line 268 of file observables.py.

Referenced by expec_val_().

```

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
```

Here is the caller graph for this function:

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

Definition at line 290 of file observables.py.

```

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

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

Definition at line 235 of file observables.py.

Referenced by expec_val_().

```

def energy_total(dataName, initValue, finalValue, increment):
    E=np.zeros((finalValue,1))
    E_k=np.zeros((finalValue,1))
    E_vi=np.zeros((finalValue,1))
    E_l=np.zeros((finalValue,1))
    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 = np.array(real,dtype='f8') #64-bit double
            a_i = np.array(img,dtype='f8') #64-bit double
            wfcr = np.reshape(a_r[:] + 1j*a_i[:],(xDim,yDim))
            wfcp = np.array(np.fft.fft2(wfcr))
            wfcr_c = np.conj(wfcr)

            E1 = np.fft.ifft2(K*wfcp)
            E2 = (V + 0.5*g*np.abs(wfcr)**2)*wfcr
            E3 = -(omega*omegaX)*(np.fft.ifft(xPy*np.fft.fft(wfcr,axis=0),axis=0) - np.fft.ifft(yPx*
      np.fft.fft(wfcr,axis=1),axis=1)  )

```

```
00255             E_k[i/incr] = np.trapz(np.trapz(wfcr_c*E1))*dx*dy
00256             E_vi[i/incr] = np.trapz(np.trapz(wfcr_c*E2))*dx*dy
00257             E_l[i/incr] = np.trapz(np.trapz(wfcr_c*E3))*dx*dy
00258             E[i/incr] = E_k[i/incr] + E_vi[i/incr] + E_l[i/incr]
00259             print (i/float(evMaxVal))
00260     np.savetxt('E_'+ str(i) + '.csv',E,delimiter=',')
00261     np.savetxt('E_k_'+ str(i) + '.csv',E_k,delimiter=',')
00262     np.savetxt('E_vi_'+ str(i) + '.csv',E_vi,delimiter=',')
00263     np.savetxt('E_l_'+ str(i) + '.csv',E_l,delimiter=',')
00264     t = np.array(range(initValue,finalValue,incr))/dt
00265     plt.plot(t,E,'r-',t,E_k,'g-',t,E_vi,'b-',t,E_l,'y-')
00266     plt.savefig("EnergyVst.pdf",dpi=200)
00267     plt.close()
```

Here is the caller graph for this function:

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

Definition at line 382 of file observables.py.

References ang_mom(), dens_struct_fact(), energy_kinetic(), energy_total(), expec_val_monopole(), expec_val_-quadrupole(), and kinertrum_loop().

```
00382
00383 def expec_val_(quant_name, quantity, dataName, initValue, finalValue, incr):
00384     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00385     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00386 #   px=open('px_0')
00387 #   py=open('py_0')
00388     xm, ym = np.meshgrid(x, y)
00389     result = []
00390     for i in range(initValue,finalValue,incr):
00391         if not os.path.exists(dataName):
00392             real=open(dataName + '_' + str(i)).read().splitlines()
00393             img=open(dataName + 'i_' + str(i)).read().splitlines()
00394             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00395             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00396             a = a_r[:] + 1j*a_i[:]
00397             wfc = np.reshape(a,(xDim,yDim))
00398             conjwfc = np.conj(wfc)
00399
00400             d1 = np.multiply( quantity, wfc )
00401             d2 = np.multiply( conjwfc, d1)
00402             result.append( np.real( np.sum( np.sum( d2  ) ) )*dx*dx )
00403         print str(100*float(i)/finalValue) + '%'
00404     np.savetxt(quant_name + '.csv',result,delimiter=',')
00405     plt.plot(range(initValue,finalValue,incr),result)
00406     plt.savefig(quant_name + ".pdf",dpi=200)
00407     plt.close()
```

Here is the call graph for this function:

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

Definition at line 330 of file observables.py.

Referenced by expec_val_().

```
00330
00331 def expec_val_monopole(dataName, initValue, finalValue, incr):
00332     x=np.asarray(open('x_0').read().splitlines(),dtype='f8')
00333     y=np.asarray(open('y_0').read().splitlines(),dtype='f8')
00334 #   px=open('px_0')
00335 #   py=open('py_0')
00336     xm, ym = np.meshgrid(x, y)
00337     result = []
00338     for i in range(initValue,finalValue,incr):
00339         if not os.path.exists(dataName):
00340             real=open(dataName + '_' + str(i)).read().splitlines()
00341             img=open(dataName + 'i_' + str(i)).read().splitlines()
00342             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00343             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00344             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()
```

Here is the caller graph for this function:

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

Definition at line 356 of file observables.py.

Referenced by expec_val_().

```
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()
```

Here is the caller graph for this function:

4.6.2.9 def observables.kinertrum (*Psi*, *dx*, *i*, *quOn*)

Kinetic energy spectrum = kinertrum.

Calculates the spectrum for compressible and incompressible kinetic energies.

Parameters

Psi	The wavefunction
dx	Increment along x
i	The current step number
quOn	Boolean to turn on quantum kinetic energy spectrum (includes phase term).

Definition at line 104 of file observables.py.

Referenced by dens_struct_fact(), and kinertrum_loop().

```
def kinertrum(Psi, dx, i, quOn):

    kMax = np.max(np.max(kx))
    Psi[np.where(Psi==0)] = 1e-100
    n_r = np.abs(Psi)**2
    n_r[np.where(n_r==0)] = 1e-100
    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(ph1,dx,dy)
    vel_ph2_x, vel_ph2_y = np.gradient(ph2,dx,dy)

    v_x = (hbar/m)*vel_ph1_x;
    v_y = (hbar/m)*vel_ph2_y;
    v_x[np.where(v_x==0)] = 1e-100
    v_y[np.where(v_y==0)] = 1e-100

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

    if quOn:
        u_x = np.multiply(u_x,np.exp(1j*np.angle(Psi)))
        u_y = np.multiply(u_y,np.exp(1j*np.angle(Psi)))

    u_kx = np.fft.fftn(u_x)
    u_ky = np.fft.fftn(u_y)

    uc_kx = ( kxm**2*u_kx + kxm*kym*u_ky ) / ( km_mag**2 + 1e-100 )
    uc_ky = ( kym*kxm*u_kx + kym**2*u_ky ) / ( km_mag**2 + 1e-100 )

    ui_kx = u_kx - uc_kx
    ui_ky = u_ky - 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.abs(np.square(uc_x) + np.square(uc_y))
    Ei = 0.5*np.abs(np.square(ui_x) + np.square(ui_y))

    fig, ax = plt.subplots()
    f = plt.imshow((Ec),cmap=plt.get_cmap('gnuplot2'))
    cbar = fig.colorbar(f)
    plt.gca().invert_yaxis()
    plt.savefig("Ec_" + str(i/incr) + ".png",dpi=200)
    plt.close()
    fig, ax = plt.subplots()
    f = plt.imshow((Ei),cmap=plt.get_cmap('gnuplot2'))
    cbar = fig.colorbar(f)
    plt.gca().invert_yaxis()
    plt.savefig("Ei_" + str(i/incr) + ".png",dpi=200)
    plt.close()

    print Ec
    #exit()
    ekc = np.zeros((xDim/2-1,1))
    eki = np.zeros((xDim/2-1,1))
    for i1 in np.arange(0,np.size(k_mag)/2 -2):
        iX = np.array(np.where(np.logical_and( k_mag[i1] >= km_mag, k_mag[i1+1] < km_mag )))
#        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]))
        ekc[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(uc_kx[iX]**2 + uc_ky[iX]**2)))/len(iX)
        eki[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(ui_kx[iX]**2 + ui_ky[iX]**2)))/len(iX)
    print i1
    np.savetxt('ekc_' + str(i) + '.csv',ekc,delimiter=',')
    np.savetxt('eki_' + str(i) + '.csv',eki,delimiter=',')
    fig, ax = plt.subplots()
    print eki[0:(xDim/2-2)]
    f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),eki[0:(xDim/2-2)])
    plt.savefig("eki_" + str(i) + ".png",dpi=200)
    f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),np.ravel(ekc[0:(xDim/2-2)]))
    plt.savefig("ekc_" + str(i) + ".png",dpi=200)
    plt.close()

```

Here is the caller graph for this function:

4.6.2.10 def observables.kinertrum_loop (*dataName,* *initValue,* *finalValue,* *incr*)

Definition at line 183 of file observables.py.

References kinertrum().

Referenced by expec_val_().

```
def kinertrum_loop(dataName, initValue, 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[:]

            kinertrum(np.reshape(a,(xDim,yDim)),dx,i,1)
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.6.3 Variable Documentation

4.6.3.1 tuple observables.c = ConfigParser.ConfigParser()

Definition at line 55 of file observables.py.

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

Definition at line 82 of file observables.py.

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

Definition at line 66 of file observables.py.

4.6.3.4 tuple observables.dky = (c.getfloat('Params','dpy'))

Definition at line 67 of file observables.py.

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

Definition at line 68 of file observables.py.

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

Definition at line 64 of file observables.py.

4.6.3.7 tuple observables.dy = (c.getfloat('Params','dy'))

Definition at line 65 of file observables.py.

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

Definition at line 61 of file observables.py.

4.6.3.9 tuple observables.g = (0.5∗N)

Definition at line 233 of file observables.py.

Referenced by Tracker.findVortex().

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

Definition at line 60 of file observables.py.

4.6.3.11 float observables.HBAR = 1.05457148e-34

Definition at line 51 of file observables.py.

4.6.3.12 float observables.hbar = 1.05457e-34

Definition at line 96 of file observables.py.

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

Definition at line 62 of file observables.py.

4.6.3.14 tuple observables.K = np.array(open('K_0').read().splitlines(),dtype='f8')

Definition at line 227 of file observables.py.

4.6.3.15 tuple observables.k_mag = np.sqrt(kx∗∗2 + ky∗∗2)

Definition at line 93 of file observables.py.

Referenced by optLatSetup().

4.6.3.16 tuple observables.km_mag = np.sqrt(kxm∗∗2 + kym∗∗2)

Definition at line 92 of file observables.py.

4.6.3.17 tuple observables.kMax = max(max(k_mag))

Definition at line 94 of file observables.py.

4.6.3.18 tuple observables.kx = np.reshape(np.array([np.linspace(0, (xDim/2-1)∗dkx, xDim/2), np.linspace((-xDim/2-1)∗dkx, -dkx, xDim/2)]), (xDim,1))

Definition at line 89 of file observables.py.

4.6.3.19 tuple observables.ky = np.reshape(np.array([np.linspace(0, (yDim/2-1)∗dky, yDim/2), np.linspace((-yDim/2-1)∗dky, -dky, yDim/2)]), (yDim,1))

Definition at line 90 of file observables.py.

4.6.3.20 float observables.m = 1.4431607e-25

Definition at line 97 of file observables.py.

4.6.3.21 tuple observables.mass = (c.getfloat('Params','Mass'))

Definition at line 72 of file observables.py.

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

Definition at line 80 of file observables.py.

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

Definition at line 77 of file observables.py.

4.6.3.24 tuple observables.omega = (c.getfloat('Params','omega'))

Definition at line 73 of file observables.py.

4.6.3.25 tuple observables.omegaX = (c.getfloat('Params','omegaX'))

Definition at line 74 of file observables.py.

4.6.3.26 tuple observables.omegaZ = (c.getfloat('Params','omegaZ'))

Definition at line 71 of file observables.py.

4.6.3.27 float observables.PI = 3.141592653589793

Definition at line 52 of file observables.py.

4.6.3.28 tuple observables.V = np.array(open('V_0').read().splitlines(),dtype='f8')

Definition at line 225 of file observables.py.

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

Definition at line 84 of file observables.py.

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

Definition at line 58 of file observables.py.

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

Definition at line 69 of file observables.py.

4.6.3.32 tuple observables.xPy = np.array(open('xPy_0').read().splitlines(),dtype='f8')

Definition at line 229 of file observables.py.

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

Definition at line 85 of file observables.py.

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

Definition at line 59 of file observables.py.

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

Definition at line 70 of file observables.py.

4.6.3.36 tuple observables.yPx = np.array(open('yPx_0').read().splitlines(),dtype='f8')

Definition at line 231 of file observables.py.

4.7 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'))

4.7.1 Detailed Description

```
stats.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
```

4.7.2 Function Documentation

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

Definition at line 57 of file stats.py.

Referenced by vis.overlap().

```
00057
00058 def lsFit(start,end,incr):
00059     L = np.matrix([
00060             [0,0,1],
00061             [1,0,1],
00062             [0,1,1],
00063             [1,1,1]
00064             ])
00065     LSQ = np.linalg.inv(np.transpose(L)*L)*np.transpose(L)
00066     for i in range(start,end,incr):
00067         v_arr=genfromtxt('vort_arr_' + str(i),delimiter=',' )
00068         real=open('wfc_ev_' + str(i)).read().splitlines()
00069         img=open('wfc_evi_' + str(i)).read().splitlines()
00070         a_r = np.asanyarray(real,dtype='f8') #64-bit double
00071         a_i = np.asanyarray(img,dtype='f8') #64-bit double
00072         a = a_r[:] + 1j*a_i[:]
00073         wfc = (np.reshape(a,(xDim,yDim)))
00074 
00075         indX = [row[0] for row in v_arr]
00076         indY = [row[1] for row in v_arr]
00077         wind = [row[2] for row in v_arr]
00078         sign = [row[3] for row in v_arr]
00079         data=[]
00080         for ii in range(0,len(indX)):
00081             p=np.matrix([[0],[0],[0],[0]],dtype=np.complex)
00082             p[0]=(wfc[indX[ii], indY[ii]])
00083             p[1]=(wfc[indX[ii]+1, indY[ii]])
00084             p[2]=(wfc[indX[ii], indY[ii]+1])
00085             p[3]=(wfc[indX[ii]+1, indY[ii]+1])
00086             rc = LSQ * np.real(p)
00087             ic = LSQ * np.imag(p)
00088 
00089             A=np.squeeze([row[0:2] for row in [rc,ic]])
00090             B=-np.squeeze([row[2] for row in [rc,ic]])
00091             r=np.linalg.lstsq(A,B)[0]
00092             data.append([indX[ii]+r[0],indY[ii]+r[1],sign[ii]])
00093 
00094 #       f = plt.imshow(abs(wfc)**2)
00095 #       plt.jet()
00096 #       plt.gca().invert_yaxis()
00097 #       plt.hold(True)
00098 #       X = [row[0] for row in data]
00099 #       Y = [row[1] for row in data]
00100 #       plt.scatter(Y,X,s=0.2,marker='.',c='red',lw=0)
00101 #       plt.scatter(indY,indX,s=0.2,marker='.',c='yellow',lw=0)
00102 #       plt.savefig("fig.png",dpi=1200)
00103 #       plt.close()
00104         np.savetxt('vort_lsq_'+str(i)+'.csv',data,delimiter=',')
```

Here is the caller graph for this function:

4.7.3 Variable Documentation

4.7.3.1 tuple stats.c = ConfigParser.ConfigParser()

Definition at line 50 of file stats.py.

4.7.3.2 tuple stats.incr = int(c.getfloat('Params','print_out'))

Definition at line 53 of file stats.py.

4.7.3.3 tuple stats.xDim = int(c.getfloat('Params','xDim'))

Definition at line 54 of file stats.py.

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

Definition at line 55 of file stats.py.

4.8 Tracker Namespace Reference

See the source file for info on functions.

Classes

- struct Vortex

  Vortex is used to track specific individual vortices. More...

Functions

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

  Finds the maxima of the optical lattice.
- int findVortex (int ∗marker, double2 ∗wfc, double radius, int xDim, double ∗x, int timestep)

  Phase winding method to determine vortex positions.
- 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)

  Tests the phase winding of the wavefunction, looking for vortices.
- double sigVOL (int2 ∗vArr, int2 ∗opLatt, double ∗x, int numVort)
- double sigVOL (struct Tracker::Vortex ∗vArr, int2 ∗opLatt, double ∗x, int numVort)

  Sigma of vortex lattice and optical lattice.
- double vortAngle (struct Tracker::Vortex ∗vortCoords, struct Vortex central, int numVort)

  Determines the angle of the vortex lattice relative to the x-axis.
- void vortArrange (struct Tracker::Vortex ∗vCoordsC, struct Vortex ∗vCoordsP, int length)

  Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.
- struct Vortex vortCentre (struct Tracker::Vortex ∗cArray, int length, int xDim)

  Determines the coords of the vortex closest to the central position.
- void vortPos (int ∗marker, struct Tracker::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.
- struct Vortex ∗ vortPosDelta (int ∗cMarker, int2 ∗pMarker, double ∗x, double tolerance, int numVortices, int xDim)
- double vortSepAvg (struct Vortex ∗vArray, struct Tracker::Vortex centre, int length)

  Determines the vortex separation at the centre of the lattice.

Variables

- char bufferT [1024]

4.8.1 Detailed Description

See the source file for info on functions. Contains all the glorious info you need to track vortices and see what they are up to.

4.8.2 Class Documentation

4.8.2.1 struct Tracker::Vortex

Vortex is used to track specific individual vortices.

coords tracks x,y positions. sign indicates direction of vortex rotation. wind indicates the unit charge of the vortex.

Definition at line 54 of file tracker.h.

Collaboration diagram for Tracker::Vortex:

Class Members

int2	coords	
int	sign	
int	wind	

4.8.3 Function Documentation

4.8.3.1 int Tracker::findOLMaxima (int ∗ *marker,* double ∗ *Vopt,* double *radius,* int *xDim,* double ∗ *x*)

Finds the maxima of the optical lattice.

Deprecated.

Definition at line 67 of file tracker.cc.

References vis::i, and Minions::maxValue().

```
00067                                                                                  {
00068         double gridValues[9];
00069         int2 mIndex[1024];
00070         int2 index;
00071         int i,j,found;
00072         found=0;
00073         for (i=1; i<xDim-1; ++i ){
00074             for(j=1; j<xDim-1;++j){
00075                 if(sqrt(x[i]*x[i] + x[j]*x[j]) < radius){
00076                     gridValues[0] = Vopt[(i-1)*xDim + (j-1)];
00077                     gridValues[1] = Vopt[(i-1)*xDim + j];
00078                     gridValues[2] = Vopt[(i-1)*xDim + (j+1)];
00079                     gridValues[3] = Vopt[i*xDim + (j-1)];
00080                     gridValues[4] = Vopt[i*xDim + j];
00081                     gridValues[5] = Vopt[i*xDim + (j+1)];
00082                     gridValues[6] = Vopt[(i+1)*xDim + (j-1)];
00083                     gridValues[7] = Vopt[(i+1)*xDim + j];
00084                     gridValues[8] = Vopt[(i+1)*xDim + (j+1)];
00085                     if(fabs((gridValues[4]-Minions::maxValue(gridValues,9))/gridValues[4])
       <= 1e-7){
00086                         //printf ("%d,%d\n",i,j);
00087                         (marker)[i*xDim + j] = 1;
00088                         index.x=i;
00089                         index.y=j;
00090                         mIndex[found] = index;
00091                         ++found;
00092                     }
00093                 }
00094             }
00095         }
```

```
        return found;
    }
```

Here is the call graph for this function:

4.8.3.2 int Tracker::findVortex (int ∗ *marker,* double2 ∗ *wfc,* double *radius,* int *xDim,* double ∗ *x,* int *timestep*)

Phase winding method to determine vortex positions.

Calculates the phase around a loop and checks if ∼ +/-2Pi.

Definition at line 136 of file tracker.cc.

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

Referenced by evolve().

```
                                                                                                                        {
            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] = Minions::complexScale(
      Minions::complexDiv( wfc[i*xDim + j],         wfc[(i+1)*xDim + j] ),        (
      Minions::complexMag( wfc[(i+1)*xDim + j])      /
      Minions::complexMag( wfc[i*xDim + j] )));
                                    g[1] = Minions::complexScale(
      Minions::complexDiv( wfc[(i+1)*xDim + j],     wfc[(i+1)*xDim + (j+1)] ),    (
      Minions::complexMag( wfc[(i+1)*xDim + (j+1)]) /
      Minions::complexMag( wfc[(i+1)*xDim + j] )));
                                    g[2] = Minions::complexScale(
      Minions::complexDiv( wfc[(i+1)*xDim + (j+1)], wfc[i*xDim + (j+1)] ),        (
      Minions::complexMag( wfc[i*xDim + (j+1)])      /
      Minions::complexMag( wfc[(i+1)*xDim + (j+1)] )));
                                    g[3] = Minions::complexScale(
      Minions::complexDiv( wfc[i*xDim + (j+1)],     wfc[i*xDim + j] ),            (
      Minions::complexMag( wfc[i*xDim + j])          /
      Minions::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;
    }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.8.3.3 void Tracker::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 186 of file tracker.cc.

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

```
                                                        {
        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;
                }
            }
        }
    }
```

4.8.3.4 int Tracker::phaseTest (int2 *vLoc,* double2 ∗ *wfc,* int *xDim*)

Tests the phase winding of the wavefunction, looking for vortices.

Definition at line 203 of file tracker.cc.

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

```
                                                     {
        int result = 0;
        double2 gridValues[4];
        double phiDelta[4];
        double sum=0.0;
        int i=vLoc.x, j=vLoc.y;
        gridValues[0] = Minions::complexScale(
Minions::complexDiv(wfc[i*xDim + j],wfc[(i+1)*xDim + j]),          (
Minions::complexMag(wfc[(i+1)*xDim + j])    /
Minions::complexMag(wfc[i*xDim + j])));
        gridValues[1] = Minions::complexScale(
Minions::complexDiv(wfc[(i+1)*xDim + j],wfc[(i+1)*
xDim + (j+1)]),      (Minions::complexMag(wfc[(i+1)*
xDim + (j+1)])/ Minions::complexMag(wfc[(i+1)*xDim + j])));
        gridValues[2] = Minions::complexScale(
Minions::complexDiv(wfc[(i+1)*xDim + (j+1)],wfc[i*
xDim + (j+1)]),      (Minions::complexMag(wfc[i*xDim + (j+1)])  /
Minions::complexMag(wfc[(i+1)*xDim + (j+1)])));
        gridValues[3] = Minions::complexScale(
Minions::complexDiv(wfc[i*xDim + (j+1)],wfc[i*xDim + j]),          (
Minions::complexMag(wfc[i*xDim + j])        /
Minions::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;
        }
        return result;
    }
```

Here is the call graph for this function:

4.8.3.5 double Tracker::sigVOL (int2 ∗ *vArr,* int2 ∗ *opLatt,* double ∗ *x,* int *numVort*)

4.8.3.6 double Tracker::sigVOL (struct Tracker::Vortex ∗ *vArr,* int2 ∗ *opLatt,* double ∗ *x,* int *numVort*)

Sigma of vortex lattice and optical lattice.

Definition at line 312 of file tracker.cc.

References Tracker::Vortex::coords, dx, and vis::i.

```
00312                                                                                  {
00313         double sigma = 0.0;
00314         double dx = abs(x[1]-x[0]);
00315         for (int i=0; i<numVort; ++i){
00316             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);
00317         }
00318         sigma /= numVort;
00319         return sigma;
00320     }
```

4.8.3.7 double Tracker::vortAngle (struct Vortex ∗ *vortCoords,* struct Vortex *central,* int *numVort*)

Determines the angle of the vortex lattice relative to the x-axis.

Definition at line 291 of file tracker.cc.

References Tracker::Vortex::coords, vis::i, and PI.

Referenced by evolve().

```
00291                                                                                         {
00292         int location;
00293         double sign=1.0;
00294         double minVal=1e300;//(pow(central.x - vortCoords[0].x,2) + pow(central.y - vortCoords[0].y,2));
00295         for (int i=0; i < numVort; ++i){
00296             if (minVal > (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) > 2e-6 && abs(central.coords.y -
      vortCoords[i].coords.y) > 2e-6){
00297                 minVal = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.coords.y -
      vortCoords[i].coords.y,2));
00298                 location = i;
00299             }
00300         }
00301         double ang=(fmod(atan2( (vortCoords[location].coords.y - central.coords.y), (vortCoords[
      location].coords.x - central.coords.x) ),PI/3));
00302         printf("Angle=%e\n",ang);
00303         return PI/3 - ang;
00304 
00305         //return PI/2 + fmod(atan2(vortCoords[location].y-central.y, vortCoords[location].x - central.x),
       PI/3);
00306         //return PI/2 - sign*acos( ( (central.x - vortCoords[location].x)*(central.x -
       vortCoords[location].x) ) / ( minVal*(central.x - vortCoords[location].x) ) );
00307     }
```

Here is the caller graph for this function:

4.8.3.8 void Tracker::vortArrange (struct Vortex ∗ *vCoordsC,* struct Vortex ∗ *vCoordsP,* int *length*)

Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.

Definition at line 249 of file tracker.cc.

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

Referenced by evolve().

```
00249                                                                                        {
00250         int dist, dist_t;
00251         int i, j, index;
00252         for ( i = 0; i < length; ++i ){
00253             dist = 0x7FFFFFFF; //arbitrary big value
00254             index = i;
00255             for ( j = i; j < length ; ++j){
00256                 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) );
00257                 if(dist > dist_t ){
```

```
                        dist = dist_t;
                        index = j;
                    }
                }
                Minions::coordSwap(vCoordsC,index,i);
            }
        }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.8.3.9 struct Vortex Tracker::vortCentre (struct Tracker::Vortex ∗ *cArray,* int *length,* int *xDim*)

Determines the coords of the vortex closest to the central position.

Useful for centering the optical lattice over v. lattice∗

Definition at line 269 of file tracker.cc.

References Tracker::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:

4.8.3.10 void Tracker::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 230 of file tracker.cc.

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

Referenced by evolve().

```
                                                                                                      {
            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;
                    }
                }
            }
        }
```

Here is the caller graph for this function:

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

4.8.3.12 double Tracker::vortSepAvg (struct Vortex ∗ *vArray,* struct Vortex *centre,* int *length*)

Determines the vortex separation at the centre of the lattice.

Definition at line 48 of file tracker.cc.

References Tracker::Vortex::coords, and result.

Referenced by evolve(), and optLatSetup().

```
00048                                                                                          {
00049         double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y -
       v_array[0].y,2));
00050         double min = 0.0;
00051         double min_tmp = 0.0;
00052         int index=0;
00053         min = sqrt( pow(centre.coords.x - vArray[0].coords.x,2) + pow(centre.coords.y - vArray[0].coords.y,
      2));
00054         for (int j=1; j<length; ++j){
00055             min_tmp = sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.coords.y - vArray[j].
      coords.y,2));
00056             if(min > min_tmp && min_tmp > 1e-7){
00057                 min = min_tmp;
00058                 index = j;
00059             }
00060         }
00061         return min;
00062     }
```

Here is the caller graph for this function:

4.8.4 Variable Documentation

4.8.4.1 char Tracker::bufferT[1024]

Definition at line 43 of file tracker.cc.

4.9 vis Namespace Reference

Functions

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

Variables

- tuple c = ConfigParser.ConfigParser()
- list CPUs = os.environ['SLURM_JOB_CPUS_PER_NODE']
- tuple data = numpy.ndarray(shape=(xDim,yDim))

- tuple dt = (c.getfloat('Params','dt'))
- tuple dx = (c.getfloat('Params','dx'))
- list ev_proc = [ ]
- list evImgList = [ ]
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- list gnd_proc = [ ]
- list gndImgList = [ ]
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple i = gndImgList.pop()
- tuple incr = int(c.getfloat('Params','print_out'))
- int num_vort = 0
- tuple p = proc.pop()
- proc = gnd_proc+ev_proc
- tuple sep = (c.getfloat('Params','dx'))
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple yMax = (c.getfloat('Params','yMax'))

4.9.1 Detailed Description

4.9.2 Function Documentation

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

Definition at line 78 of file vis.py.

Referenced by overlap().

```
00078
00079 def delaunay(dataName,dataType,value):
00080     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=',' )
00081     data = np.array([[row[0],row[1]] for row in v_arr])
00082     dln = sp.spatial.Delaunay(data)
00083     plt.triplot(data[:,0],data[:,1],dln.simplices.copy(),linewidth=0.5,color='b',marker='.')
00084     plt.xlim(300,700);plt.ylim(300,700);
00085     plt.savefig('delaun_' + str(value) + '.png',dpi=200)
00086     print 'Saved Delaunay @ t=' + str(value)
```

Here is the caller graph for this function:

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

Definition at line 133 of file vis.py.

```
00133
00134 def hist_gen(name,value,num_bins):
00135     v_arr=genfromtxt('vort_arr_' + str(value),delimiter=',' )
00136     H=[]
00137     count=0
00138
00139     for i1 in range(0,v_arr.size/2):
00140         for i2 in range(i1,v_arr.size/2):
00141             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 ))
00142             count = count + 1
00143     plt.title('Vortex lattice @ t=' + str(value*dt))
00144     plt.ticklabel_format(style='scientific')
00145     plt.ticklabel_format(style='scientific',axis='x', scilimits=(0,0))
00146     h = plt.hist(H, bins=num_bins)
00147     plt.savefig(name + "_" + str(value) + ".pdf")
00148     plt.close()
```

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

Definition at line 149 of file vis.py.

```
00149
00150 def image_gen(dataName, initValue, finalValue, increment,imgdpi):
00151     for i in range(initValue,finalValue,increment):
00152         if not os.path.exists(dataName+"r_"+str(i)+"_abspsi2.png"):
00153             real=open(dataName + '_' + str(i)).read().splitlines()
00154             img=open(dataName + 'i_' + str(i)).read().splitlines()
00155             a_r = numpy.asanyarray(real,dtype='f8') #64-bit double
00156             a_i = numpy.asanyarray(img,dtype='f8') #64-bit double
00157             a = a_r[:] + 1j*a_i[:]
00158             b = np.reshape(a,(xDim,yDim))
00159             f = plt.imshow(abs(b)**2)
00160             plt.jet()
00161             plt.gca().invert_yaxis()
00162             plt.savefig(dataName+"r_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00163             plt.close()
00164             g = plt.imshow(np.angle(b))
00165             plt.gca().invert_yaxis()
00166             plt.savefig(dataName+"r_"+str(i)+"_phi.png",dpi=imgdpi)
00167             plt.close()
00168             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))**2)
00169             plt.gca().invert_yaxis()
00170             plt.jet()
00171             plt.savefig(dataName+"p_"+str(i)+"_abspsi2.png",dpi=imgdpi)
00172             plt.close()
00173             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00174             plt.gca().invert_yaxis()
00175             plt.savefig(dataName+"p_"+str(i)+"_phi.png",dpi=imgdpi)
00176             plt.close()
00177             print "Saved figure: " + str(i) + ".png"
00178             plt.close()
00179         else:
00180             print "File(s) " + str(i) +".png already exist."
```

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

Definition at line 181 of file vis.py.

References laplacian(), and struct_fact().

```
00181
00182 def image_gen_single(dataName, value, imgdpi,opmode):
00183     real=open(dataName + '_' + str(0)).read().splitlines()
00184     img=open(dataName + 'i_' + str(0)).read().splitlines()
00185     a1_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00186     a1_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00187     a1 = a1_r[:] + 1j*a1_i[:]
00188     b1 = np.reshape(a1,(xDim,yDim))
00189
00190     if not os.path.exists(dataName+"r_"+str(value)+"_abspsi2.png"):
00191         real=open(dataName + '_' + str(value)).read().splitlines()
00192         img=open(dataName + 'i_' + str(value)).read().splitlines()
00193         a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00194         a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00195         a = a_r[:] + 1j*a_i[:]
00196         b = np.reshape(a,(xDim,yDim))
00197         m_val=np.max(np.abs(b)**2)
00198         #scaleAxis(b,dataName,"_abspsi2",value,imgdpi)
00199         if opmode & 0b100000 > 0:
00200 #             fig, ax = plt.subplots()
00201 #             #plt.rc('text',usetex=True)
00202 #             #plt.rc('font',family='serif')
00203 #             f = plt.imshow((abs(b)**2 - abs(b1)**2),cmap='gnuplot2',vmin=-6,vmax=6)
00204 #             plt.title(r'$\left(\rho( r,t ) - \rho( r,t_0 )\right),t=$' + str(value*dt))
00205 #             cbar = fig.colorbar(f)
00206 #             plt.gca().set_xlabel('x '+ str((dx)))
00207 #             plt.gca().set_ylabel('x '+ str(dx))
00208 #             plt.gca().invert_yaxis()
00209 #             plt.savefig(dataName+"r_"+str(value)+"_diffabspsi2.png",dpi=imgdpi)
00210 #             plt.close()
00211 #             #plt.rc('text',usetex=True)
00212 #             #plt.rc('font',family='serif')
00213
00214             fig, ax = plt.subplots()
00215             f = plt.imshow((abs(b)**2),cmap='gnuplot2',vmin=0,vmax=1e7)
00216             plt.title('rho(r) @ t=' + str(value*dt))
00217             plt.title(r'$\\rho \left( r,t \right),\,t=$' + str(value*dt))
00218
00219             #plugins.connect(fig, plugins.MousePosition(fontsize=14))
00220
00221             cbar = fig.colorbar(f)
00222             plt.gca().set_xlabel('x '+ str((dx)))
00223             plt.gca().set_ylabel('x '+ str(dx))
00224             plt.gca().invert_yaxis()
00225             plt.savefig(dataName+"r_"+str(value)+"_abspsi2.png",dpi=imgdpi)
00226             plt.axis('off')
00227             plt.savefig(dataName+"r_"+str(value)+"_abspsi2_axis0.pdf",bbox_inches='tight',dpi=imgdpi)
00228             plt.close()
00229
00230         if opmode & 0b010000 > 0:
00231             fig, ax = plt.subplots()
00232             g = plt.imshow(np.angle(b))
00233             cbar = fig.colorbar(g)
00234             plt.gca().invert_yaxis()
00235             plt.title('theta(r) @ t=' + str(value*dt))
00236             plt.savefig(dataName+"r_"+str(value)+"_phi.png",dpi=imgdpi)
00237             plt.close()
00238
00239         if opmode & 0b001000 > 0:
00240             fig, ax = plt.subplots()
00241             f = plt.imshow(abs(np.fft.fftshift(np.fft.fft2(b)))**2)
00242             cbar = fig.colorbar(f)
00243             plt.gca().invert_yaxis()
00244             plt.jet()
00245             plt.title('rho(p) @ t=' + str(value*dt))
00246             plt.savefig(dataName+"p_"+str(value)+"_abspsi2.png",dpi=imgdpi)
00247             plt.close()
00248
00249         if opmode & 0b000100 > 0:
00250             fig, ax = plt.subplots()
00251             g = plt.imshow(np.angle(np.fft.fftshift(np.fft.fft2(b))))
00252             cbar = fig.colorbar(g)
00253             plt.gca().invert_yaxis()
00254             plt.title('theta(p) @ t=' + str(value*dt))
00255             plt.savefig(dataName+"p_"+str(value)+"_phi.png",dpi=imgdpi)
00256             plt.close()
00257
00258         if opmode & 0b000010 > 0:
00259             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."
```

Here is the call graph for this function:

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

Definition at line 96 of file vis.py.

Referenced by image_gen_single().

```

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()
```

Here is the caller graph for this function:

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

Definition at line 121 of file vis.py.

Referenced by overlap().

```

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()
```

Here is the caller graph for this function:

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

Definition at line 309 of file vis.py.

References delaunay(), stats.lsFit(), opPot(), hist3d.plot_hist_pcolor(), and vort_traj().

```

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
```

```
00314     a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00315     wfc0 = a_r[:] + 1j*a_i[:]
00316     for i in range(initValue,finalValue,increment):
00317         real=open(dataName + '_' + str(value)).read().splitlines()
00318         img=open(dataName + 'i_' + str(value)).read().splitlines()
00319         a_r = numpy.asanyarray(real,dtype='f8') #128-bit complex
00320         a_i = numpy.asanyarray(img,dtype='f8') #128-bit complex
00321         a = a_r[:] + 1j*a_i[:]
00322         b = np.dot(wfc0,a)
00323         print i, np.sum(b)
```

Here is the call graph for this function:

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

Definition at line 298 of file vis.py.

```
00298
00299 def scaleAxis(data,dataName,label,value,imgdpi):
00300     fig, ax = plt.subplots()
00301     ax.xaxis.set_major_locator(ScaledLocator(dx=dx))
00302     ax.xaxis.set_major_formatter(ScaledLocator(dx=dx))
00303     f = plt.imshow(abs(data)**2)
00304     cbar = fig.colorbar(f)
00305     plt.gca().invert_yaxis()
00306     plt.jet()
00307     plt.savefig(dataName+"r_"+str(value)+"_"+label +".png",dpi=imgdpi)
00308     plt.close()
```

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

Definition at line 111 of file vis.py.

Referenced by image_gen_single().

```
00111
00112 def struct_fact(density,name,imgdpi):
00113     fig, ax = plt.subplots()
00114     #f = plt.quiver(gx,gy)
00115     f = plt.imshow((np.abs(np.fft.fftshift(np.fft.fft2(density)))),cmap=plt.get_cmap('prism'))
00116     cbar = fig.colorbar(f)
00117     cbar.set_clim(1e6,1e11)
00118     plt.jet()
00119     plt.savefig(name + "_struct_log10.png",dpi=imgdpi)
00120     plt.close()
```

Here is the caller graph for this function:

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

Definition at line 87 of file vis.py.

```
00087
00088 def voronoi(dataName,dataType,value):
00089     v_arr=genfromtxt(dataName + str(value) + dataType,delimiter=',' )
00090     data = [[row[0],row[1]] for row in v_arr]
00091     vor = Voronoi(data)
00092     voronoi_plot_2d(vor)
00093     plt.xlim(300,700);plt.ylim(300,700);
00094     plt.savefig('voronoi_' + str(value) + '.png',dpi=200)
00095     print 'Saved Voronoi @ t=' + str(value)
```

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

Definition at line 268 of file vis.py.

Referenced by overlap().

```
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."
```

Here is the caller graph for this function:

4.9.3 Variable Documentation

4.9.3.1 tuple vis.c = ConfigParser.ConfigParser()

Definition at line 59 of file vis.py.

4.9.3.2 list vis.CPUs = os.environ['SLURM_JOB_CPUS_PER_NODE']

Definition at line 34 of file vis.py.

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

Definition at line 76 of file vis.py.

4.9.3.4 tuple vis.dt = (c.getfloat('Params','dt'))

Definition at line 71 of file vis.py.

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

Definition at line 70 of file vis.py.

4.9.3.6 list vis.ev_proc = [ ]

Definition at line 342 of file vis.py.

4.9.3.7 list vis.evImgList = [ ]

Definition at line 336 of file vis.py.

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

Definition at line 67 of file vis.py.

4.9.3.9 list vis.gnd_proc = [ ]

Definition at line 341 of file vis.py.

4.9.3.10 list vis.gndImgList = [ ]

Definition at line 335 of file vis.py.

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

Definition at line 66 of file vis.py.

4.9.3.12 tuple vis.i = gndImgList.pop()

Definition at line 344 of file vis.py.

Referenced by delta_define(), energy_angmom(), evolve(), Tracker.findOLMaxima(), Tracker.findVortex(), initialise(), Minions.maxValue(), Minions.minValue(), multipass(), Tracker.olPos(), optLatSetup(), pSum(), pSumT(), FileIO.-readIn(), sepAvg(), Tracker.sigVOL(), Minions.sumAvg(), Tracker.vortAngle(), Tracker.vortArrange(), Tracker.vort-Centre(), Tracker.vortPos(), FileIO.writeOut(), FileIO.writeOutDouble(), FileIO.writeOutInt(), FileIO.writeOutInt2(), FileIO.writeOutParam(), and FileIO.writeOutVortex().

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

Definition at line 68 of file vis.py.

4.9.3.14 int vis.num_vort = 0

Definition at line 74 of file vis.py.

4.9.3.15 tuple vis.p = proc.pop()

Definition at line 360 of file vis.py.

Referenced by appendData(), and newParam().

4.9.3.16 vis.proc = gnd_proc+ev_proc

Definition at line 354 of file vis.py.

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

Definition at line 69 of file vis.py.

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

Definition at line 64 of file vis.py.

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

Definition at line 72 of file vis.py.

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

Definition at line 65 of file vis.py.

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

Definition at line 73 of file vis.py.

4.10 vort Namespace Reference

Classes

- class Vortex
- class VtxList

Functions

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

Variables

- tuple c = ConfigParser.ConfigParser()
- int counter = 0
- tuple current = self.element(pos-1)
- list data = [ ]
- tuple dcp = set(uid_c)

- tuple dpc = set(uid_p)
- tuple dt = (c.getfloat('Params','dt'))
- list dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]
- tuple dx = (c.getfloat('Params','dx'))
- tuple evMaxVal = int(c.getfloat('Params','esteps'))
- tuple gndMaxVal = int(c.getfloat('Params','gsteps'))
- int i = 0
- tuple incr = int(c.getfloat('Params','print_out'))
- tuple index_r = vorts_c.idx_min_dist(vorts_p.element(i3))
- tuple max_uid = vorts_p.max_uid()
- int pos = 0
- int pos_l = 0
- tuple r = m.sqrt((self.x - vtx.x)∗∗2 + (self.y - vtx.y)∗∗2)
- ret_idx = counter
- 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 v0c = vorts_c.element(index_r[0])
- tuple v0p = vorts_p.element(i3)
- tuple v1c = vorts_c.element(index_r[0])
- tuple v_arr_c = genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
- tuple v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])
- tuple v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])
- tuple v_arr_p = genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')

  v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')
- tuple v_arr_p_coords = np.array([[a for a in v][:2] for v in v_arr_p])
- tuple v_arr_p_sign = np.array([[a for a in v][2] for v in v_arr_p])
- int val = 0
- tuple vorts_c = VtxList()
- tuple vorts_c_update = sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])
- tuple vorts_p = VtxList()
- vtx = self.head
- 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 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_pos = self.vtx_uid(uid_i)
- list vtx_pos_c = [ ]
- list vtx_pos_p = [ ]
- tuple xDim = int(c.getfloat('Params','xDim'))
- tuple xMax = (c.getfloat('Params','xMax'))
- tuple yDim = int(c.getfloat('Params','yDim'))
- tuple yMax = (c.getfloat('Params','yMax'))

4.10.1 Detailed Description

```
vort.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
```

4.10.2 Class Documentation

4.10.2.1 class vort::Vortex

Definition at line 56 of file vort.py.

Collaboration diagram for vort.Vortex:

4.10.2.2 class vort::VtxList

Definition at line 90 of file vort.py.

Collaboration diagram for vort.VtxList:

4.10.3 Function Documentation

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

Definition at line 59 of file vort.py.

Referenced by __init__().

```
00059
    def __init__(self,uid,x,y,isOn,sign=1):
```

Here is the caller graph for this function:

4.10.3.2 def vort.__init__ (*self*)

Definition at line 93 of file vort.py.

References __init__().

```
00093
    def __init__(self):
```

Here is the call graph for this function:

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

Definition at line 141 of file vort.py.

```
00141
    def add(self,Vtx,index=None): #Add a vtx at index, otherwise end
```

4.10.3.4 def vort.as_np (*self*)

Definition at line 157 of file vort.py.

```
00157
    def as_np(self): #Return numpy array with format x,y,sign,uid,isOn
```

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

Definition at line 84 of file vort.py.

Referenced by Tracker.vortArrange().

```
00084
    def dist(self,vtx): #Distance between self and vtx
```

Here is the caller graph for this function:

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

Definition at line 236 of file vort.py.

```
00236
def do_the_thing(start,fin,incr): #Performs the tracking
```

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

Definition at line 100 of file vort.py.

```
00100
    def element(self,pos): #Get vtx at position pos
```

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

Definition at line 175 of file vort.py.

```
00175
    def idx_min_dist(self,vortex, isSelf=False): #Closest vtx to self
```

4.10.3.9 def vort.max_uid (*self*)

Definition at line 124 of file vort.py.

References max_uid.

```
00124
    def max_uid(self): #Return position and value of largest uid
```

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

Definition at line 191 of file vort.py.

```
00191
    def remove(self,pos): #Remove vortices outside articificial boundary
```

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

Definition at line 210 of file vort.py.

```
00210
    def swap_uid(self,uid_i,uid_f): #Swap uid between vtx
```

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

Definition at line 79 of file vort.py.

```
00079
    def update_next(self,next): #Get next vortex
```

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

Definition at line 74 of file vort.py.

```
00074
    def update_on(self,isOn): #Vortex is trackable
```

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

Definition at line 69 of file vort.py.

Referenced by vort_increase().

```
00069
    def update_uid(self,uid):
```

Here is the caller graph for this function:

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

Definition at line 217 of file vort.py.

```
00217
    def vort_decrease(self,positions,vorts_p): #Turn off vortex timeline
```

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

Definition at line 227 of file vort.py.

References update_uid().

```
00227
    def vort_increase(self,positions,vorts_p): #Add new vtx to tracking
```

Here is the call graph for this function:

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

Definition at line 114 of file vort.py.

```
00114
    def vtx_uid(self,uid): #Get vtx with identifier uid
```

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

Definition at line 170 of file vort.py.

```
00170
    def write_out(self,time,data): #Write out CSV file as  x,y,sign,uid,isOn
```

4.10.4 Variable Documentation

4.10.4.1 tuple vort.c = ConfigParser.ConfigParser()

Definition at line 42 of file vort.py.

4.10.4.2 int vort.counter = 0

Definition at line 177 of file vort.py.

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

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

Definition at line 194 of file vort.py.

4.10.4.4 list vort.data = [ ]

Definition at line 160 of file vort.py.

4.10.4.5 tuple vort.dcp = set(uid_c)

Definition at line 274 of file vort.py.

4.10.4.6 tuple vort.dpc = set(uid_p)

Definition at line 273 of file vort.py.

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

Definition at line 51 of file vort.py.

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

Definition at line 159 of file vort.py.

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

Definition at line 50 of file vort.py.

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

Definition at line 48 of file vort.py.

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

Definition at line 47 of file vort.py.

4.10.4.12 int vort.i = 0

Definition at line 161 of file vort.py.

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

Definition at line 49 of file vort.py.

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

Definition at line 258 of file vort.py.

4.10.4.15 tuple vort.max_uid = vorts_p.max_uid()

Definition at line 219 of file vort.py.

Referenced by max_uid().

4.10.4.16 int vort.pos = 0

Definition at line 117 of file vort.py.

4.10.4.17 int vort.pos_l = 0

Definition at line 102 of file vort.py.

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

Definition at line 86 of file vort.py.

4.10.4.19 vort.ret_idx = counter

Definition at line 178 of file vort.py.

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

Definition at line 269 of file vort.py.

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

Definition at line 270 of file vort.py.

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

Definition at line 260 of file vort.py.

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

Definition at line 261 of file vort.py.

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

Definition at line 262 of file vort.py.

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

Definition at line 244 of file vort.py.

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

Definition at line 246 of file vort.py.

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

Definition at line 248 of file vort.py.

4.10.4.28 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 239 of file vort.py.

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

Definition at line 245 of file vort.py.

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

Definition at line 247 of file vort.py.

4.10.4.31 vort.val = 0

Definition at line 126 of file vort.py.

4.10.4.32 tuple vort.vorts_c = VtxList()

Definition at line 242 of file vort.py.

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

Definition at line 285 of file vort.py.

4.10.4.34 tuple vort.vorts_p = VtxList()

Definition at line 241 of file vort.py.

4.10.4.35 tuple vort.vtx = self.head

Definition at line 104 of file vort.py.

4.10.4.36 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 254 of file vort.py.

4.10.4.37 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 250 of file vort.py.

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

Definition at line 212 of file vort.py.

4.10.4.39 tuple vort.vtx_pos_c = [ ]

Definition at line 276 of file vort.py.

4.10.4.40 tuple vort.vtx_pos_p = [ ]

Definition at line 275 of file vort.py.

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

Definition at line 45 of file vort.py.

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

Definition at line 52 of file vort.py.

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

Definition at line 46 of file vort.py.

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

Definition at line 53 of file vort.py.

Chapter 5

Class Documentation

5.1 BEC2D::Wavefunction Class Reference

Collaboration diagram for BEC2D::Wavefunction:

Public Member Functions

- Wavefunction ()
- Wavefunction (int xDim, int yDim, double xMax, double yMax)
- int2 getGridSize (int xDim, int yDim)
- double2 & getWfc ()
- double2 initWfc ()
- bool setGridSize (int xDim, int yDim)

Private Attributes

- double2 dimMax
- int2 gridSize
- double2 $*$ wfc = new double2[xDim$*$yDim]

5.1.1 Detailed Description

Definition at line 39 of file wavefunction.cu.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 BEC2D::Wavefunction::Wavefunction ( )

5.1.2.2 BEC2D::Wavefunction::Wavefunction (int *xDim*, int *yDim*, double *xMax*, double *yMax*)

Definition at line 57 of file wavefunction.cu.

```
                                                                          {

         }
```

5.1.3 Member Function Documentation

5.1.3.1 int2 BEC2D::Wavefunction::getGridSize (int *xDim*, int *yDim*)

5.1.3.2 double2& BEC2D::Wavefunction::getWfc ( )

5.1.3.3 double2 BEC2D::Wavefunction::initWfc ( )

5.1.3.4 BEC2D::Wavefunction::setGridSize (int *xDim*, int *yDim*)

Definition at line 60 of file wavefunction.cu.

```
00060                                                   {
00061
00062     }
```

5.1.4 Member Data Documentation

5.1.4.1 double2 BEC2D::Wavefunction::dimMax [private]

Definition at line 42 of file wavefunction.cu.

5.1.4.2 int2 BEC2D::Wavefunction::gridSize [private]

Definition at line 41 of file wavefunction.cu.

5.1.4.3 double2* BEC2D::Wavefunction::wfc = new double2[xDim*yDim] [private]

Definition at line 43 of file wavefunction.cu.

The documentation for this class was generated from the following file:

- src/wavefunction.cu

Chapter 6

File Documentation

6.1 bin/batch_run.sh File Reference

6.2 batch_run.sh

```
#GPUE: Split Operator based GPU solver for Nonlinear
#Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
#<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
#Redistribution and use in source and binary forms, with or without
#modification, are permitted provided that the following conditions are
#met:
#
#1. Redistributions of source code must retain the above copyright
#notice, this list of conditions and the following disclaimer.
#
#2. Redistributions in binary form must reproduce the above copyright
#notice, this list of conditions and the following disclaimer in the
#documentation and/or other materials provided with the distribution.
#
#3. Neither the name of the copyright holder nor the names of its
#contributors may be used to endorse or promote products derived from
#this software without specific prior written permission.
#
#THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
#PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
#TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
#PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
#LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
#NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
#SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#!/bin/bash

#SBATCH --job-name=
#SBATCH --partition=
#SBATCH --mem=
#SBATCH --cpus-per-task=
#SBATCH --ntasks=
#SBATCH --error=
#SBATCH --mail-type=ALL
#SBATCH --mail-user=
```

6.3 bin/path.sh File Reference

6.4 path.sh

```
#GPUE: Split Operator based GPU solver for Nonlinear
#Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
#<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
#Redistribution and use in source and binary forms, with or without
```

```
#modification, are permitted provided that the following conditions are
#met:
#
#1. Redistributions of source code must retain the above copyright
#notice, this list of conditions and the following disclaimer.
#
#2. Redistributions in binary form must reproduce the above copyright
#notice, this list of conditions and the following disclaimer in the
#documentation and/or other materials provided with the distribution.
#
#3. Neither the name of the copyright holder nor the names of its
#contributors may be used to endorse or promote products derived from
#this software without specific prior written permission.
#
#THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
#PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
#TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
#PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
#LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
#NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
#SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#!/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
```

6.5 bin/run.sh File Reference

6.6 run.sh

```
#GPUE: Split Operator based GPU solver for Nonlinear
#Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
#<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
#Redistribution and use in source and binary forms, with or without
#modification, are permitted provided that the following conditions are
#met:
#
#1. Redistributions of source code must retain the above copyright
#notice, this list of conditions and the following disclaimer.
#
#2. Redistributions in binary form must reproduce the above copyright
#notice, this list of conditions and the following disclaimer in the
#documentation and/or other materials provided with the distribution.
#
#3. Neither the name of the copyright holder nor the names of its
#contributors may be used to endorse or promote products derived from
#this software without specific prior written permission.
#
#THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
#PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
#TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
#PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
#LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
#NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
#SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#!/bin/bash
i=0
EMAIL=mymail@addr.com
count=0
NAME=$1
PARAMS=$2
declare -a JOBS=(-1 -1 -1 -1 -1 -1 -1 -1)
function run_gpue_test {
    echo $1
}

function run_gpue {
    if [[ $(echo $1 | head -c 1) == "#" ]];then
        return;
    elif [[ $(echo $1 | head -c 1) == "" ]];then
        return
    fi

    if [ -n  "$NAME" ];then
```

```
        NAME=$(echo $NAME)_
    fi
    sleep 1
    A=$(date '+%y/%m/%d/%H_%M_%S')
    if [ -d ./$A ]; then
        echo "Exists"
        A=$A-$i
        i=$((i+1))
    fi
    echo "$NAME$A"
    mkdir -p $NAME$A
    cp ./gpue ./$NAME$A; cp -r ./src ./$NAME$A; cp -r ./include ./$NAME$A; cp ./Makefile ./$NAME$A; cp -r ./py ./$NAME$A; cp -r ./bin ./$NAME$A; cp ./wfc_load ./$NAME$A; cp ./wfci_load ./$NAME$A;
    cd ./$NAME$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
    rm -rf ./*.png
    #python ./py/hist3d.py
    rm wfc*
    mail -s "#Completed GPU Job# $A" $EMAIL < $(echo $(cat result.log; cat ./Params.dat))
    cd ../../../../..
    sleep 1
}

while read line ; do
    run_gpue "$line" &
    #echo "Running $line"
    JOBS[$count]=$!
    let count+=1

    if [ $count -gt 7 ]; then
        wait
        count=0
    fi
done < $PARAMS
```

6.7 bin/sanity_test.sh File Reference

Variables

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

6.7.1 Variable Documentation

6.7.1.1 FILE

Initial value:

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

Definition at line 32 of file sanity_test.sh.

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

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

Definition at line 39 of file sanity_test.sh.

6.8 sanity_test.sh

```
00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030
00031 #!/bin/bash
00032 FILE=$1
00033 COUNTER=0
00034 POSITION=-1
00035 ARR[0]=0
00036 for i in $(cat $FILE);
00037 do
00038     let POSITION++
00039     if [ "$i" != "0.0000000000000000e+00" ];
00040     then
00041         ARR[$COUNTER]=$POSITION
00042         let COUNTER++
00043     fi
00044
00045 done
00046 echo Non-zero elements $COUNTER
00047 echo "Elements located at:"
00048
00049 for item in ${ARR[*]}
00050 do
00051     printf "%s\n" $item
00052 done
```

6.9 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

6.9.1 Function Documentation

6.9.1.1 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`)

6.9.2 Variable Documentation

6.9.2.1 OLDPWD

Initial value:

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

Definition at line 31 of file upload_vids.sh.

6.10 upload_vids.sh

```
00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031 OLDPWD=$(pwd)
00032 for i in $(cat ./ogg.txt | grep wfc);
00033 do
00034     echo $(if [[ $(basename $(dirname $i))=='images' ]];
00035         then
00036             cd $(dirname $i)/../bin;
00037             TITLE=$(head -n 1 run_params.conf)
00038             SUMMARY=$(head -n 20 ../result.log)
00039             cd -
00040             google youtube post --category Tech $i --title "$TITLE" --summary "$SUMMARY" --access=
      unlisted $i
00041         fi);
00042 done
00043
```

6.11 bin/zippit.sh File Reference

Functions

- for i in (cat manifest.txt)

Variables

- do echo Working on $i

6.11.1 Function Documentation

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

Referenced by conjugate(), and Minions::fInvSqRt().

Here is the caller graph for this function:

6.11.2 Variable Documentation

6.11.2.1 $HOME builds bin pigz p r $i

Definition at line 31 of file zippit.sh.

6.12 zippit.sh

```
00001 #GPUE: Split Operator based GPU solver for Nonlinear
00002 #Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 #<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.
00004 #Redistribution and use in source and binary forms, with or without
00005 #modification, are permitted provided that the following conditions are
00006 #met:
00007 #
00008 #1. Redistributions of source code must retain the above copyright
00009 #notice, this list of conditions and the following disclaimer.
00010 #
00011 #2. Redistributions in binary form must reproduce the above copyright
00012 #notice, this list of conditions and the following disclaimer in the
00013 #documentation and/or other materials provided with the distribution.
00014 #
00015 #3. Neither the name of the copyright holder nor the names of its
00016 #contributors may be used to endorse or promote products derived from
00017 #this software without specific prior written permission.
00018 #
00019 #THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00020 #"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00021 #LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00022 #PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00023 #HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00024 #SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00025 #TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00026 #PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00027 #LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00028 #NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00029 #SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00030 #!/bin/bash
00031 for i in $(cat manifest.txt); do echo 'Working on $i'; $HOME/builds/bin/pigz -
      p 24 -r $i; done
```

6.13 include/constants.h File Reference

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

Macros

- #define EPSILON_0 8.854187817620e-12
- #define HBAR 1.05457148e-34
- #define INV_RT_2 0.7071067811865475
- #define MU_0 4∗PI∗1e-7

- #define MU_B 9.27400915e-24
- #define MU_N 5.05078324e-27
- #define PI 3.141592653589793
- #define Q 1.602176565e-19
- #define RT_2 1.4142135623730951

6.13.1 Macro Definition Documentation

6.13.1.1 #define EPSILON_0 8.854187817620e-12

Definition at line 42 of file constants.h.

6.13.1.2 #define HBAR 1.05457148e-34

Definition at line 37 of file constants.h.

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

6.13.1.3 #define INV_RT_2 0.7071067811865475

Definition at line 43 of file constants.h.

6.13.1.4 #define MU_0 4∗PI∗1e-7

Definition at line 41 of file constants.h.

6.13.1.5 #define MU_B 9.27400915e-24

Definition at line 39 of file constants.h.

6.13.1.6 #define MU_N 5.05078324e-27

Definition at line 38 of file constants.h.

6.13.1.7 #define PI 3.141592653589793

Definition at line 36 of file constants.h.

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

6.13.1.8 #define Q 1.602176565e-19

Definition at line 40 of file constants.h.

6.13.1.9 #define RT_2 1.4142135623730951

Definition at line 44 of file constants.h.

6.14 constants.h

```
/*** constants.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
#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
```

6.15 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 Array
- struct Param

Functions

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

6.15.1 Class Documentation

6.15.1.1 struct Array

Definition at line 46 of file ds.h.

Collaboration diagram for Array:

Class Members

Param ∗	array	
size_t	length	
size_t	used	

6.15.1.2 struct Param

Definition at line 40 of file ds.h.

Collaboration diagram for Param:

Class Members

double	data	
char	title[32]	

6.15.2 Function Documentation

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

Definition at line 42 of file ds.cc.

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

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

```
00042                                               {
00043     Param p = newParam(t,d);
00044     if(arr->used == arr->length){
00045         arr->length *= 2;
00046         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
      Param));
00047     }
00048     arr->array[arr->used] = p;
00049     arr->used = arr->used + 1;
00050 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.15.2.2 void freeArray (Array ∗ *arr*)

Definition at line 52 of file ds.cc.

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

```
00052                         {
00053     free(arr->array);
00054     arr->array = NULL;
00055     arr->used = 0;
00056     arr->length = 0;
00057 }
```

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

Definition at line 36 of file ds.cc.

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

Referenced by main().

```
00036                                        {
00037     arr->array = (Param*) malloc(initLen*sizeof(Param));
00038     arr->used = 0;
00039     arr->length = initLen;
00040 }
```

Here is the caller graph for this function:

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

Definition at line 59 of file ds.cc.

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

Referenced by appendData().

```
00059                                {
00060     Param p;
00061     strcpy(p.title,t);
00062     p.data = d;
00063     return p;
00064 }
```

Here is the caller graph for this function:

6.16 ds.h

```
/*** ds.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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
```

6.17 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:

Namespaces

- FileIO

  Check source file for information on functions.

Functions

- double2 ∗ FileIO::readIn (char ∗, char ∗, int, int)
- int FileIO::readState (char ∗)
- void FileIO::writeOut (char ∗, char ∗, double2 ∗, int, int)
- void FileIO::writeOutDouble (char ∗, char ∗, double ∗, int, int)
- void FileIO::writeOutInt (char ∗, char ∗, int ∗, int, int)
- void FileIO::writeOutInt2 (char ∗, char ∗, int2 ∗, int, int)
- void FileIO::writeOutParam (char ∗, Array, char ∗)
- void FileIO::writeOutVortex (char ∗, char ∗, struct Tracker::Vortex ∗, int, int)

6.18 fileIO.h

```
/*** fileIO.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
```

```
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#ifndef FILEIO_H
#define FILEIO_H
#include "../include/ds.h"
#include "../include/tracker.h"

namespace FileIO{
    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 Tracker::Vortex*, int, int);
    void writeOutParam(char*, Array, char*);
    int readState(char*);
}
#endif
```

6.19 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

- __global__ void angularOp (double, double, double2 ∗, double ∗, double2 ∗)
- __device__ double2 braKetMult (double2 in1, double2 in2)
- __global__ void cMult (cufftDoubleComplex ∗, cufftDoubleComplex ∗, cufftDoubleComplex ∗)
- __global__ void cMultDensity (double2 ∗, double2 ∗, double2 ∗, double, double, double, int, int)
- __device__ double complexMagnitudeSquared (double2)
- __device__ double2 conjugate (double2 in)
- __global__ void energyCalc (double2 ∗wfc, double2 ∗op, double dt, double2 ∗energy, int gnd_state, int op_-space, double sqrt_omegaz_mass)
- __device__ unsigned int getBid3d3d ()
- unsigned int getGid3d3d ()
- __device__ unsigned int getTid3d3d ()
- __global__ void multipass (cufftDoubleComplex ∗, cufftDoubleComplex ∗, int)
- __global__ void pinVortex (cufftDoubleComplex ∗, cufftDoubleComplex ∗, cufftDoubleComplex ∗)
- __global__ void pSum (double ∗in1, double ∗output, int pass)

  Routine for parallel summation.
- __device__ double2 realCompMult (double scalar, double2 comp)
- __global__ void reduce (double2 ∗, double ∗)
- __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.

6.19.1 Function Documentation

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

Definition at line 153 of file kernels.cu.

References getGid3d3d(), and result.

```
00153                                                                                             {
00154     unsigned int gid = getGid3d3d();
00155     double2 result;
00156     double op;
00157     op = exp( -omega*xpyypx[gid]*dt);
00158     result.x=wfc[gid].x*op;
00159     result.y=wfc[gid].y*op;
00160     out[gid]=result;
00161 }
```

Here is the call graph for this function:

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

Definition at line 88 of file kernels.cu.

References complexMultiply(), and conjugate().

Referenced by energyCalc().

```
00089 {
00090     return complexMultiply(conjugate(in1),in2);
00091 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

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

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

Definition at line 104 of file kernels.cu.

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

```
00104
                           {
00105     double2 result;
00106     double gDensity;
00107     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00108     gDensity = N*complexMagnitudeSquared(in2[tid])*4*
      HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*
      HBAR));
00109
00110     if(gstate == 0){
00111         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR) );
00112         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00113         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00114     }
00115     else{
00116         double2 tmp;
00117         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(
      dt/HBAR));
00118         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(
      dt/HBAR));
00119
00120         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00121         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00122     }
00123     out[tid] = result;
00124 }
```

Here is the call graph for this function:

6.19.1.5 __device__ double complexMagnitudeSquared (double2)

Definition at line 74 of file kernels.cu.

Referenced by cMultDensity(), and energyCalc().

```
00074                                                                   {
00075     return in.x*in.x + in.y*in.y;
00076 }
```

Here is the caller graph for this function:

6.19.1.6 __device__ double2 conjugate (double2 *in*)

Definition at line 56 of file kernels.cu.

References in(), and result.

Referenced by braKetMult().

```
00056                                         {
00057     double2 result = in;
00058     result.y = -result.y;
00059     return result;
00060 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

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

Definition at line 193 of file kernels.cu.

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

```
00193
                                  {
00194     unsigned int gid = getGid3d3d();
00195     double hbar_dt = HBAR/dt;
00196     double g_local = 0.0;
00197     double2 result;
00198     double opLocal;
00199     if(op_space)
00200         g_local = gDenConst*sqrt_omegaz_mass*complexMagnitudeSquared(
      wfc[gid]);
00201     if(!gnd_state){
00202         opLocal = -log(op[gid].x + g_local)*hbar_dt;
00203     }
00204     else{
00205         opLocal = cos(op[gid].x + g_local)*hbar_dt;
00206     }
00207     result = braKetMult(wfc[gid], realCompMult(opLocal,
      wfc[gid]));
00208     //printf("oplocal=%e    Resx=%e Resy=%e\n",opLocal,result.x,result.y);
00209     energy[gid].x += result.x;
00210     energy[gid].y += result.y;
00211 }
```

Here is the call graph for this function:

6.19.1.8 __device__ unsigned int getBid3d3d ( )

Definition at line 46 of file kernels.cu.

```
00046                                  {
00047     return blockIdx.x + gridDim.x*(blockIdx.y + gridDim.y * blockIdx.z);
00048 }
```

6.19.1.9 unsigned int getGid3d3d ( )

Definition at line 41 of file kernels.cu.

Referenced by angularOp(), cMult(), energyCalc(), multipass(), pSum(), pSumT(), scalarDiv(), and scalarDiv_wfc-Norm().

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

Here is the caller graph for this function:

6.19.1.10 __device__ unsigned int getTid3d3d ( )

Definition at line 52 of file kernels.cu.

```
00052                                {
00053     return blockDim.x * ( blockDim.y * ( blockDim.z + ( threadIdx.z * blockDim.y ) )  + threadIdx.y )  +
      threadIdx.x;
00054 }
```

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

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

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

Routine for parallel summation.

Can be looped over from host.

Definition at line 239 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
00239                                                      {
00240         unsigned int tid = threadIdx.x;
00241         unsigned int bid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x;// printf("bid0=%d\n",bid);
00242         unsigned int gid = getGid3d3d();
00243         extern __shared__ double sdata2[];
00244         for(int i = blockDim.x>>1; i > 0; i>>=1){
00245                 if(tid < blockDim.x>>1){
00246                         sdata2[tid] += sdata2[tid + i];
00247                 }
00248                 __syncthreads();
00249         }
00250         if(tid==0){
00251                 output[bid] = sdata2[0];
00252         }
00253 }
```

Here is the call graph for this function:

6.19.1.14 __device__ double2 realCompMult (double *scalar*, double2 *comp*)

Definition at line 62 of file kernels.cu.

References result.

Referenced by energyCalc().

```
00062                                                       {
00063     double2 result;
```

```
    result.x = scalar * comp.x;
    result.y = scalar * comp.y;
    return result;
}
```

Here is the caller graph for this function:

6.19.1.15 __global__ void reduce (double2 ∗ , double ∗)

6.19.1.16 __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 130 of file kernels.cu.

References getGid3d3d(), and result.

```
                                                                    {
    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;
}
```

Here is the call graph for this function:

6.19.1.17 __global__ void scalarDiv1D (double2 ∗ , double2 ∗)

6.19.1.18 __global__ void scalarDiv2D (double2 ∗ , double2 ∗)

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

As above, but normalises for wfc.

Definition at line 142 of file kernels.cu.

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

```
                                                                                   {
    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;
}
```

Here is the call graph for this function:

6.20 kernels.h

```
/*** kernels.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

```

```
2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#ifndef KERNELS_H
#define KERNELS_H
#include<stdio.h>
/* CUDA function declarations */

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
```

6.21 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:

Namespaces

- Minions

Functions

- double2 Minions::complexDiv (double2 num, double2 den)

- double Minions::complexMag (double2 in)
- double Minions::complexMag2 (double2 in)
- double2 Minions::complexMult (double2 in1, double2 in2)
- double2 Minions::complexScale (double2 comp, double scale)
- double2 Minions::conj (double2 c)
- void Minions::coordSwap (struct Tracker::Vortex ∗vCoords, int src, int dest)
- double Minions::fInvSqRt (double)

  id magic hackery
- double Minions::maxValue (double ∗, int)
- double Minions::minValue (double ∗, int)
- double Minions::psi2 (double2)
- double Minions::sumAvg (double ∗in, int len)

6.22 minions.h

```
/*** minions.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#ifndef MINIONS_H
#define MINIONS_H

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

namespace Minions{
    /* 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 Tracker::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






```

6.23 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 "tracker.h"
```

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

Functions

- 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.
- int isError (int, char ∗)
- void optLatSetup (struct Tracker::Vortex centre, double ∗V, struct Tracker::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.
- void parSum (double2 ∗, double2 ∗, int, int, int)

Variables

- double a_s
- int ang_mom = 0
- long atoms
- double dt
- double dx
- double dy
- cufftDoubleComplex ∗ EappliedField
- cufftDoubleComplex ∗ EK
- double ∗ Energy
- double ∗ Energy_gpu
- cudaError_t err
- long esteps
- cufftDoubleComplex ∗ EV
- cufftDoubleComplex ∗ EV_opt
- cufftDoubleComplex ∗ ExPy
- cufftDoubleComplex ∗ EyPx

- double gdt
- cufftDoubleComplex ∗ GK
- int gpe = 0
- dim3 grid
- long gsteps
- cufftDoubleComplex ∗ GV
- cufftDoubleComplex ∗ GV_half
- cufftDoubleComplex ∗ GxPy
- cufftDoubleComplex ∗ GyPx
- double interaction
- double ∗ K
- cufftDoubleComplex ∗ K_gpu
- double l
- double laser_power
- double mass
- double omegaX
- double omegaY
- double omegaZ
- cufftDoubleComplex ∗ par_sum
- struct Params ∗ paramS
- double ∗ Phi
- cufftHandle plan_1d
- cufftHandle plan_2d
- int print
- double ∗ px
- double ∗ py
- double ∗ r
- int read_wfc
- cufftResult result
- cudaStream_t streamA
- cudaStream_t streamB
- cudaStream_t streamC
- cudaStream_t streamD
- int threads
- double ∗ V
- cufftDoubleComplex ∗ V_gpu
- double ∗ V_opt
- cufftDoubleComplex ∗ wfc
- cufftDoubleComplex ∗ wfc0
- cufftDoubleComplex ∗ wfc_backup
- cufftDoubleComplex ∗ wfc_gpu
- int write_it
- double ∗ x
- int xDim
- double xi
- double xMax
- double ∗ xp
- double ∗ xPy
- double ∗ xPy_gpu
- double ∗ y
- int yDim
- double yMax
- double ∗ yp
- double ∗ yPx
- double ∗ yPx_gpu

6.23.1 Function Documentation

6.23.1.1 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 655 of file split_op.cu.

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

```
00655
                                      {
00656     double renorm_factor_2d=1.0/pow(xDim*yDim,0.5);
00657     double result=0;
00658
00659     for (int i=0; i < xDim*yDim; ++i){
00660         Energy[i] = 0.0;
00661     }
00662
00663
00664 /*  cudaMalloc((void**) &energy_gpu, sizeof(double2) * xDim*yDim);
00665
00666     energyCalc<<<grid,threads>>>( gpuWfc, V_op, 0.5*dt, energy_gpu, gState,1,i 0.5*sqrt(omegaZ/mass));
00667     result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_FORWARD );
00668     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00669
00670     energyCalc<<<grid,threads>>>( gpuWfc, K_op, dt, energy_gpu, gState,0, 0.5*sqrt(omegaZ/mass));
00671     result = cufftExecZ2Z( plan_2d, gpuWfc, gpuWfc, CUFFT_INVERSE );
00672     scalarDiv<<<grid,threads>>>( gpuWfc, renorm_factor_2d, gpuWfc ); //Normalise
00673
00674     err=cudaMemcpy(energy, energy_gpu, sizeof(cufftDoubleComplex)*xDim*yDim, cudaMemcpyDeviceToHost);
00675
00676     for(int i=0; i<xDim*yDim; i++){
00677         result += energy[i].x;
00678         //printf("En=%E\n",result*dx*dy);
00679     }
00680     return result*dx*dy;
00681 */
00682
00683 }
```

6.23.1.2 int isError (int , char ∗)

Definition at line 58 of file split_op.cu.

References result.

```
00058                                         {
00059     if(result!=0){printf("Error has occurred for method %s with return type %d\n",
      c,result);
00060         exit(result);
00061     }
00062     return result;
00063 }
```

6.23.1.3 void optLatSetup (struct Tracker::Vortex *centre,* double ∗ *V,* struct Tracker::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 596 of file split_op.cu.

References appendData(), buffer, Tracker::Vortex::coords, dt, dx, dy, EV_opt, HBAR, vis::i, observables::k_mag, PI, sepMinEpsilon, Tracker::vortSepAvg(), FileIO::writeOut(), xDim, and yDim.

Referenced by evolve().

```
00596
                                                                                    {
00597     int i,j;
00598     double sepMin = Tracker::vortSepAvg(vArray,centre,num_vortices);
00599     sepMin = sepMin*(1 + sepMinEpsilon);
00600     appendData(&params,"Vort_sep",(double)sepMin);
00601     /*
00602     * Defining the necessary k vectors for the optical lattice
00603     */
00604     double k_mag = ((2*PI/(sepMin*dx))/2)*(2/sqrt(3)); // Additional /2 as a result of lambda/2
       period
00605     double2* k = (double2*) malloc(sizeof(double2)*3);
00606     appendData(&params,"kmag",(double)k_mag);
00607     k[0].x = k_mag * cos(0*PI/3 + theta_opt);
00608     k[0].y = k_mag * sin(0*PI/3 + theta_opt);
00609     k[1].x = k_mag * cos(2*PI/3 + theta_opt);
00610     k[1].y = k_mag * sin(2*PI/3 + theta_opt);
00611     k[2].x = k_mag * cos(4*PI/3 + theta_opt);
00612     k[2].y = k_mag * sin(4*PI/3 + theta_opt);
00613
00614     double2 *r_opt = (double2*) malloc(sizeof(double2)*xDim);
00615
00616 /*  for (int ii = 0; ii < xDim; ++ii){
00617         r_opt[ii].x = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.x)/(xDim-1);
00618         r_opt[ii].y = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.y)/(yDim-1);
00619     }
00620 */
00621     FileIO::writeOut(buffer,"r_opt",r_opt,xDim,0);
00622     appendData(&params,"k[0].x",(double)k[0].x);
00623     appendData(&params,"k[0].y",(double)k[0].y);
00624     appendData(&params,"k[1].x",(double)k[1].x);
00625     appendData(&params,"k[1].y",(double)k[1].y);
00626     appendData(&params,"k[2].x",(double)k[2].x);
00627     appendData(&params,"k[2].y",(double)k[2].y);
00628
00629     double x_shift = dx*(9+(0.5*xDim-1) - centre.coords.x);//sin(theta_opt)*(sepMin);
00630     double y_shift = dy*(0+(0.5*yDim-1) - centre.coords.y);//cos(theta_opt)*(sepMin);
00631
00632     printf("Xs=%e\nYs=%e\n",x_shift,y_shift);
00633
00634     //#pragma omp parallel for private(j)
00635     for ( j=0; j<yDim; ++j ){
00636         for ( i=0; i<xDim; ++i ){
00637             v_opt[j*xDim + i] = intensity*(
00638                                 pow( abs( cos( k[0].x*( x[i] + x_shift ) + k[0].
      y*( y[j] + y_shift ) ) ), 2)
00639                           + pow( abs( cos( k[1].x*( x[i] + x_shift ) + k[1].y*(
      y[j] + y_shift ) ) ), 2)
00640                           + pow( abs( cos( k[2].x*( x[i] + x_shift ) + k[2].y*(
      y[j] + y_shift ) ) ), 2)
00641             /*                  pow( abs( cos( k[0].x*( r_opt[i].x + x_shift ) + k[0].y*( r_opt[j].y +
       y_shift ) ) ), 2)
00642                           + pow( abs( cos( k[1].x*( r_opt[i].x + x_shift ) + k[1].y*( r_opt[j].y + y_shift )
       ) ), 2)
00643                           + pow( abs( cos( k[2].x*( r_opt[i].x + x_shift ) + k[2].y*( r_opt[j].y + y_shift )
       ) ), 2)
00644             */          );
00645             EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim +
      i])*(dt/(2*HBAR)));
00646             EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(
      dt/(2*HBAR)));
00647         }
00648     }
00649
00650 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

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

Definition at line 572 of file split_op.cu.

References dx, dy, threads, and yDim.

Referenced by evolve().

```
00572                                                                                  {
00573         int grid_tmp = xDim*yDim;
00574         int block = grid_tmp/threads;
00575         int thread_tmp = threads;
00576         int pass = 0;
00577         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);
}
```

Here is the caller graph for this function:

6.23.2 Variable Documentation

6.23.2.1 double a_s

Definition at line 66 of file split_op.h.

Referenced by evolve(), and initialise().

6.23.2.2 int ang_mom = 0

Definition at line 62 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.3 long atoms

Definition at line 74 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.4 double dt

Definition at line 70 of file split_op.h.

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

6.23.2.5 double dx

Definition at line 75 of file split_op.h.

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

6.23.2.6 double dy

Definition at line 75 of file split_op.h.

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

6.23.2.7 cufftDoubleComplex ∗ EappliedField

Definition at line 81 of file split_op.h.

Referenced by initialise().

6.23.2.8 cufftDoubleComplex ∗ EK

Definition at line 81 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.9 double∗ Energy

Definition at line 82 of file split_op.h.

Referenced by initialise().

6.23.2.10 double ∗ Energy_gpu

Definition at line 82 of file split_op.h.

Referenced by initialise().

6.23.2.11 cudaError_t err

Definition at line 58 of file split_op.h.

Referenced by main().

6.23.2.12 long esteps

Definition at line 74 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.13 cufftDoubleComplex ∗ EV

Definition at line 81 of file split_op.h.

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

6.23.2.14 cufftDoubleComplex ∗ EV_opt

Definition at line 81 of file split_op.h.

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

6.23.2.15 cufftDoubleComplex ∗ ExPy

Definition at line 81 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.16 cufftDoubleComplex ∗ EyPx

Definition at line 81 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.17 double gdt

Definition at line 70 of file split_op.h.

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

6.23.2.18 cufftDoubleComplex * GK

Definition at line 81 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.19 int gpe = 0

Definition at line 63 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.20 dim3 grid

Definition at line 95 of file split_op.h.

Referenced by initialise().

6.23.2.21 long gsteps

Definition at line 74 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.22 cufftDoubleComplex * GV

Definition at line 81 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.23 cufftDoubleComplex * GV_half

Definition at line 81 of file split_op.h.

6.23.2.24 cufftDoubleComplex * GxPy

Definition at line 81 of file split_op.h.

6.23.2.25 cufftDoubleComplex * GyPx

Definition at line 81 of file split_op.h.

6.23.2.26 double interaction

Definition at line 91 of file split_op.h.

Referenced by evolve(), and parseArgs().

6.23.2.27 double ∗ K

Definition at line 82 of file split_op.h.

Referenced by initialise().

6.23.2.28 cufftDoubleComplex ∗ K_gpu

Definition at line 85 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.29 double l

Definition at line 99 of file split_op.h.

Referenced by Minions::fInvSqRt(), initialise(), and parseArgs().

6.23.2.30 double laser_power

Definition at line 92 of file split_op.h.

Referenced by evolve(), and parseArgs().

6.23.2.31 double mass

Definition at line 66 of file split_op.h.

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

6.23.2.32 double omegaX

Definition at line 66 of file split_op.h.

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

6.23.2.33 double omegaY

Definition at line 66 of file split_op.h.

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

6.23.2.34 double omegaZ

Definition at line 66 of file split_op.h.

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

6.23.2.35 cufftDoubleComplex ∗ par_sum

Definition at line 85 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.36 struct Params∗ paramS

Definition at line 50 of file split_op.cu.

6.23.2.37 double ∗ Phi

Definition at line 82 of file split_op.h.

Referenced by initialise().

6.23.2.38 cufftHandle plan_1d

Definition at line 78 of file split_op.h.

Referenced by evolve(), and initialise().

6.23.2.39 cufftHandle plan_2d

Definition at line 78 of file split_op.h.

Referenced by evolve(), and initialise().

6.23.2.40 int print

Definition at line 73 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.41 double ∗ px

Definition at line 75 of file split_op.h.

6.23.2.42 double ∗ py

Definition at line 75 of file split_op.h.

6.23.2.43 double ∗ r

Definition at line 82 of file split_op.h.

Referenced by initialise().

6.23.2.44 int read_wfc

Definition at line 73 of file split_op.h.

Referenced by main(), and parseArgs().

6.23.2.45 cufftResult result

Definition at line 59 of file split_op.h.

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

6.23.2.46 cudaStream_t streamA

Definition at line 88 of file split_op.h.

6.23.2.47 cudaStream_t streamB

Definition at line 88 of file split_op.h.

6.23.2.48 cudaStream_t streamC

Definition at line 88 of file split_op.h.

6.23.2.49 cudaStream_t streamD

Definition at line 88 of file split_op.h.

6.23.2.50 int threads

Definition at line 96 of file split_op.h.

Referenced by initialise(), and parSum().

6.23.2.51 double ∗ V

Definition at line 82 of file split_op.h.

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

6.23.2.52 cufftDoubleComplex ∗ V_gpu

Definition at line 85 of file split_op.h.

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

6.23.2.53 double ∗ V_opt

Definition at line 82 of file split_op.h.

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

6.23.2.54 cufftDoubleComplex∗ wfc

Definition at line 81 of file split_op.h.

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

6.23.2.55 cufftDoubleComplex ∗ wfc0

Definition at line 81 of file split_op.h.

6.23.2.56 cufftDoubleComplex ∗ wfc_backup

Definition at line 81 of file split_op.h.

Referenced by initialise().

6.23.2.57 cufftDoubleComplex∗ wfc_gpu

Definition at line 85 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.58 int write_it

Definition at line 73 of file split_op.h.

Referenced by evolve(), and parseArgs().

6.23.2.59 double∗ x

Definition at line 75 of file split_op.h.

Referenced by cMultDensity(), energyCalc(), evolve(), initialise(), main(), scalarDiv_wfcNorm(), FileIO::writeOut(), and FileIO::writeOutInt2().

6.23.2.60 int xDim

Definition at line 73 of file split_op.h.

Referenced by delta_define(), energy_angmom(), evolve(), initialise(), main(), Tracker::olPos(), optLatSetup(), parseArgs(), and Tracker::vortPos().

6.23.2.61 double xi

Definition at line 67 of file split_op.h.

Referenced by evolve().

6.23.2.62 double xMax

Definition at line 75 of file split_op.h.

Referenced by initialise().

6.23.2.63 double ∗ xp

Definition at line 75 of file split_op.h.

Referenced by initialise().

6.23.2.64 double ∗ xPy

Definition at line 82 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.65 double ∗ xPy_gpu

Definition at line 82 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.66 double ∗ y

Definition at line 75 of file split_op.h.

Referenced by cMultDensity(), evolve(), Tracker::findVortex(), initialise(), main(), Tracker::olPos(), Tracker::phaseTest(), scalarDiv_wfcNorm(), FileIO::writeOut(), and FileIO::writeOutInt2().

6.23.2.67 int yDim

Definition at line 73 of file split_op.h.

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

6.23.2.68 double yMax

Definition at line 75 of file split_op.h.

Referenced by initialise().

6.23.2.69 double ∗ yp

Definition at line 75 of file split_op.h.

Referenced by initialise().

6.23.2.70 double ∗ yPx

Definition at line 82 of file split_op.h.

Referenced by initialise(), and main().

6.23.2.71 double ∗ yPx_gpu

Definition at line 82 of file split_op.h.

Referenced by initialise(), and main().

6.24 split_op.h

```
/*** split_op.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
```

```
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#ifndef SPLIT_OP_H
#define SPLIT_OP_H

#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 "tracker.h"
#ifdef __linux
    #include<omp.h>
#elif __APPLE__
    //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 parSum(double2* , double2* , int , int , int );
void optLatSetup(struct Tracker::Vortex centre, double*
V, struct Tracker::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


/*class SplitOp{

}*/
```

6.25 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 Tracker::Vortex

    Vortex is used to track specific individual vortices. More...

Namespaces

- Tracker

    See the source file for info on functions.

Functions

- int Tracker::findOLMaxima (int ∗marker, double ∗V, double radius, int xDim, double ∗x)

    Finds the maxima of the optical lattice.
- int Tracker::findVortex (int ∗marker, double2 ∗wfc, double radius, int xDim, double ∗x, int timestep)

    Phase winding method to determine vortex positions.
- void Tracker::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 Tracker::phaseTest (int2 vLoc, double2 ∗wfc, int xDim)

    Tests the phase winding of the wavefunction, looking for vortices.
- double Tracker::sigVOL (int2 ∗vArr, int2 ∗opLatt, double ∗x, int numVort)
- double Tracker::vortAngle (struct Tracker::Vortex ∗vortCoords, struct Vortex central, int numVort)

    Determines the angle of the vortex lattice relative to the x-axis.
- void Tracker::vortArrange (struct Tracker::Vortex ∗vCoordsC, struct Vortex ∗vCoordsP, int length)

    Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.
- struct Vortex Tracker::vortCentre (struct Tracker::Vortex ∗cArray, int length, int xDim)

Determines the coords of the vortex closest to the central position.

- void Tracker::vortPos (int ∗marker, struct Tracker::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.
- struct Vortex ∗ Tracker::vortPosDelta (int ∗cMarker, int2 ∗pMarker, double ∗x, double tolerance, int numVortices, int xDim)
- double Tracker::vortSepAvg (struct Vortex ∗vArray, struct Tracker::Vortex centre, int length)

  Determines the vortex separation at the centre of the lattice.

6.25.1 Class Documentation

6.25.1.1 struct Tracker::Vortex

Vortex is used to track specific individual vortices.

coords tracks x,y positions. sign indicates direction of vortex rotation. wind indicates the unit charge of the vortex.

Definition at line 54 of file tracker.h.

Collaboration diagram for Tracker::Vortex:

Class Members

int2	coords	
int	sign	
int	wind	

6.26 tracker.h

```
/*** tracker.h - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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>

namespace Tracker{

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

    int findVortex(int*,double2*, double, int, double*, int);
    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
```

6.27 py/hist3d.py File Reference

Namespaces

- hist3d

Functions

- def hist3d.plot_hist_pcolor
- def hist3d.plot_xyz_histogram

Variables

- tuple hist3d.c = ConfigParser.ConfigParser()
- tuple hist3d.dt = (c.getfloat('Params','dt'))
- tuple hist3d.dx = (c.getfloat('Params','dx'))
- tuple hist3d.evMaxVal = int(c.getfloat('Params','esteps'))
- tuple hist3d.gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple hist3d.incr = int(c.getfloat('Params','print_out'))
- int hist3d.num_vort = 0
- tuple hist3d.sep = (c.getfloat('Params','dx'))
- tuple hist3d.xDim = int(c.getfloat('Params','xDim'))
- tuple hist3d.xMax = (c.getfloat('Params','xMax'))
- tuple hist3d.yDim = int(c.getfloat('Params','yDim'))
- tuple hist3d.yMax = (c.getfloat('Params','yMax'))

6.28 hist3d.py

```
'''
hist3d.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

```

```
Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''

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 = 0#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')

```

6.29 py/hist_it.py File Reference

Namespaces

- hist_it

6.30 hist_it.py

```
'''
hist_it.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''
```

6.31 py/observables.py File Reference

Namespaces

- observables

Functions

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

  Kinetic energy spectrum = kinertrum.
- def observables.kinertrum_loop

Variables

- tuple observables.c = ConfigParser.ConfigParser()
- tuple observables.data = numpy.ndarray(shape=(xDim,yDim))
- tuple observables.dkx = (c.getfloat('Params','dpx'))
- tuple observables.dky = (c.getfloat('Params','dpy'))
- tuple observables.dt = (c.getfloat('Params','dt'))
- tuple observables.dx = (c.getfloat('Params','dx'))
- tuple observables.dy = (c.getfloat('Params','dy'))
- tuple observables.evMaxVal = int(c.getfloat('Params','esteps'))
- tuple observables.g = (0.5∗N)
- tuple observables.gndMaxVal = int(c.getfloat('Params','gsteps'))
- float observables.HBAR = 1.05457148e-34
- float observables.hbar = 1.05457e-34
- tuple observables.incr = int(c.getfloat('Params','print_out'))
- tuple observables.K = np.array(open('K_0').read().splitlines(),dtype='f8')
- tuple observables.k_mag = np.sqrt(kx∗∗2 + ky∗∗2)
- tuple observables.km_mag = np.sqrt(kxm∗∗2 + kym∗∗2)
- tuple observables.kMax = max(max(k_mag))
- tuple observables.kx = np.reshape(np.array([np.linspace(0, (xDim/2-1)∗dkx, xDim/2), np.linspace((-xDim/2-1)∗dkx, -dkx, xDim/2)]), (xDim,1))
- tuple observables.ky = np.reshape(np.array([np.linspace(0, (yDim/2-1)∗dky, yDim/2), np.linspace((-yDim/2-1)∗dky, -dky, yDim/2)]), (yDim,1))
- float observables.m = 1.4431607e-25
- tuple observables.mass = (c.getfloat('Params','Mass'))
- tuple observables.N = int(c.getfloat('Params','atoms'))
- tuple observables.num_vort = int(c.getfloat('Params','Num_vort'))
- tuple observables.omega = (c.getfloat('Params','omega'))
- tuple observables.omegaX = (c.getfloat('Params','omegaX'))
- tuple observables.omegaZ = (c.getfloat('Params','omegaZ'))
- float observables.PI = 3.141592653589793
- tuple observables.V = np.array(open('V_0').read().splitlines(),dtype='f8')
- tuple observables.x = np.asarray(open('x_0').read().splitlines(),dtype='f8')
- tuple observables.xDim = int(c.getfloat('Params','xDim'))
- tuple observables.xMax = (c.getfloat('Params','xMax'))
- tuple observables.xPy = np.array(open('xPy_0').read().splitlines(),dtype='f8')

- tuple observables.y = np.asarray(open('y_0').read().splitlines(),dtype='f8')
- tuple observables.yDim = int(c.getfloat('Params','yDim'))
- tuple observables.yMax = (c.getfloat('Params','yMax'))
- tuple observables.yPx = np.array(open('yPx_0').read().splitlines(),dtype='f8')

6.32 observables.py

```
'''
observables.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''
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

HBAR = 1.05457148e-34
PI = 3.141592653589793

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'))
dy = (c.getfloat('Params','dy'))
dkx = (c.getfloat('Params','dpx'))
dky = (c.getfloat('Params','dpy'))
dt = (c.getfloat('Params','dt'))
xMax = (c.getfloat('Params','xMax'))
yMax = (c.getfloat('Params','yMax'))
omegaZ = (c.getfloat('Params','omegaZ'))
mass = (c.getfloat('Params','Mass'))
omega = (c.getfloat('Params','omega'))
omegaX = (c.getfloat('Params','omegaX'))
```

```

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’)

kx = np.reshape( np.array( [np.linspace( 0, (xDim/2-1)*dkx, xDim/2), np.linspace( (-xDim/2-1)*dkx, -dkx, xDim/2)]), (xDim,1) )
ky = np.reshape( np.array( [np.linspace( 0, (yDim/2-1)*dky, yDim/2), np.linspace( (-yDim/2-1)*dky, -dky, yDim/2)]), (yDim,1) )
kxm, kym = np.meshgrid(kx,ky)
km_mag = np.sqrt( kxm**2 + kym**2 )
k_mag = np.sqrt( kx**2 + ky**2 )
kMax = max(max(k_mag))

hbar = 1.05457e-34
m = 1.4431607e-25

## Kinetic energy spectrum = kinertrum. Calculates the spectrum for compressible and incompressible kinetic energies.
# @param Psi The wavefunction
# @param dx Increment along x
# @param i The current step number
# @param quOn Boolean to turn on quantum kinetic energy spectrum (includes phase term).
def kinertrum(Psi, dx, i, quOn):

    kMax = np.max(np.max(kx))
    Psi[np.where(Psi==0)] = 1e-100
    n_r = np.abs(Psi)**2
    n_r[np.where(n_r==0)] = 1e-100
    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(ph1,dx,dy)
    vel_ph2_x, vel_ph2_y = np.gradient(ph2,dx,dy)

    v_x = (hbar/m)*vel_ph1_x;
    v_y = (hbar/m)*vel_ph2_y;
    v_x[np.where(v_x==0)] = 1e-100
    v_y[np.where(v_y==0)] = 1e-100

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

    if quOn:
        u_x = np.multiply(u_x,np.exp(1j*np.angle(Psi)))
        u_y = np.multiply(u_y,np.exp(1j*np.angle(Psi)))

    u_kx = np.fft.fftn(u_x)
    u_ky = np.fft.fftn(u_y)

    uc_kx = ( kxm**2*u_kx + kxm*kym*u_ky ) / ( km_mag**2 + 1e-100 )
    uc_ky = ( kym*kxm*u_kx + kym**2*u_ky ) / ( km_mag**2 + 1e-100 )

    ui_kx = u_kx - uc_kx
    ui_ky = u_ky - 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.abs(np.square(uc_x) + np.square(uc_y))
    Ei = 0.5*np.abs(np.square(ui_x) + np.square(ui_y))

    fig, ax = plt.subplots()
    f = plt.imshow((Ec),cmap=plt.get_cmap(’gnuplot2’))
    cbar = fig.colorbar(f)
    plt.gca().invert_yaxis()
    plt.savefig("Ec_" + str(i/incr) + ".png",dpi=200)
    plt.close()
    fig, ax = plt.subplots()
    f = plt.imshow((Ei),cmap=plt.get_cmap(’gnuplot2’))
    cbar = fig.colorbar(f)
    plt.gca().invert_yaxis()
    plt.savefig("Ei_" + str(i/incr) + ".png",dpi=200)
    plt.close()

    print Ec
    #exit()
    ekc = np.zeros((xDim/2-1,1))
    eki = np.zeros((xDim/2-1,1))
    for i1 in np.arange(0,np.size(k_mag)/2 -2):
        iX = np.array(np.where(np.logical_and( k_mag[i1] >= km_mag, k_mag[i1+1] < km_mag )))
#        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]))
        ekc[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(uc_kx[iX]**2 + uc_ky[iX]**2)))/len(iX)
        eki[i1] = (0.5*m*k_mag[i1]) * (np.sum(np.abs(ui_kx[iX]**2 + ui_ky[iX]**2)))/len(iX)
    print i1
    np.savetxt(’ekc_’ + str(i) + ’.csv’,ekc,delimiter=’,’)
    np.savetxt(’eki_’ + str(i) + ’.csv’,eki,delimiter=’,’)
    fig, ax = plt.subplots()
    print eki[0:(xDim/2-2)]
    f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),eki[0:(xDim/2-2)])
    plt.savefig("eki_" + str(i) + ".png",dpi=200)
    f = plt.loglog(np.ravel(k_mag[0:(xDim/2 -2)]),np.ravel(ekc[0:(xDim/2-2)]))
    plt.savefig("ekc_" + str(i) + ".png",dpi=200)
    plt.close()


def kinertrum_loop(dataName, initValue, 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[:]

            kinertrum(np.reshape(a,(xDim,yDim)),dx,i,1)

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

            kinertrum(np.reshape(a,(xDim,yDim)),dx,i,0)
            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()

V = np.array(open(’V_0’).read().splitlines(),dtype=’f8’)
V = np.reshape(V,(xDim,yDim))
K = np.array(open(’K_0’).read().splitlines(),dtype=’f8’)
K = np.reshape(K,(xDim,yDim))
xPy = np.array(open(’xPy_0’).read().splitlines(),dtype=’f8’)
xPy = np.reshape(xPy,(xDim,yDim))
yPx = np.array(open(’yPx_0’).read().splitlines(),dtype=’f8’)
yPx = np.reshape(yPx,(xDim,yDim))
g = (0.5*N)*4.0*HBAR*HBAR*PI*(4.67e-9/mass)*np.sqrt(mass*omegaZ/(2.0*PI*HBAR))

def energy_total(dataName, initValue, finalValue, increment):
    E=np.zeros((finalValue,1))
    E_k=np.zeros((finalValue,1))
    E_vi=np.zeros((finalValue,1))
    E_l=np.zeros((finalValue,1))
    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 = np.array(real,dtype=’f8’) #64-bit double
            a_i = np.array(img,dtype=’f8’) #64-bit double
            wfcr = np.reshape(a_r[:] + 1j*a_i[:],(xDim,yDim))
            wfcp = np.array(np.fft.fft2(wfcr))
            wfcr_c = np.conj(wfcr)

            E1 = np.fft.ifft2(K*wfcp)
            E2 = (V + 0.5*g*np.abs(wfcr)**2)*wfcr
            E3 = -(omega*omegaX)*(np.fft.ifft(xPy*np.fft.fft(wfcr,axis=0),axis=0) - np.fft.ifft(yPx*
np.fft.fft(wfcr,axis=1),axis=1)  )

            E_k[i/incr] = np.trapz(np.trapz(wfcr_c*E1))*dx*dy
            E_vi[i/incr] = np.trapz(np.trapz(wfcr_c*E2))*dx*dy
            E_l[i/incr] = np.trapz(np.trapz(wfcr_c*E3))*dx*dy
            E[i/incr] = E_k[i/incr] + E_vi[i/incr] + E_l[i/incr]
            print (i/float(evMaxVal))
    np.savetxt('E_'+ str(i) + '.csv',E,delimiter=',')
    np.savetxt('E_k_'+ str(i) + '.csv',E_k,delimiter=',')
    np.savetxt('E_vi_'+ str(i) + '.csv',E_vi,delimiter=',')
    np.savetxt('E_l_'+ str(i) + '.csv',E_l,delimiter=',')
    t = np.array(range(initValue,finalValue,incr))/dt
    plt.plot(t,E,'r-',t,E_k,'g-',t,E_vi,'b-',t,E_l,'y-')
    plt.savefig("EnergyVst.pdf",dpi=200)
    plt.close()

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__':
    kinertrum_loop('wfc_ev', 0, evMaxVal, incr)
    exit()
    energy_total('wfc_ev',0,evMaxVal,incr)
    dens_struct_fact('wfc_ev', 0, evMaxVal, 500)

    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)
```

6.33 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'))

6.34 stats.py

```
'''
stats.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''
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=',')
```

6.35 py/vis.py File Reference

Namespaces

- vis

Functions

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

Variables

- tuple vis.c = ConfigParser.ConfigParser()
- list vis.CPUs = os.environ['SLURM_JOB_CPUS_PER_NODE']
- tuple vis.data = numpy.ndarray(shape=(xDim,yDim))
- tuple vis.dt = (c.getfloat('Params','dt'))
- tuple vis.dx = (c.getfloat('Params','dx'))
- list vis.ev_proc = [ ]
- list vis.evImgList = [ ]
- tuple vis.evMaxVal = int(c.getfloat('Params','esteps'))
- list vis.gnd_proc = [ ]
- list vis.gndImgList = [ ]
- tuple vis.gndMaxVal = int(c.getfloat('Params','gsteps'))
- tuple vis.i = gndImgList.pop()
- tuple vis.incr = int(c.getfloat('Params','print_out'))
- int vis.num_vort = 0
- tuple vis.p = proc.pop()
- vis.proc = gnd_proc+ev_proc
- tuple vis.sep = (c.getfloat('Params','dx'))
- tuple vis.xDim = int(c.getfloat('Params','xDim'))
- tuple vis.xMax = (c.getfloat('Params','xMax'))
- tuple vis.yDim = int(c.getfloat('Params','yDim'))
- tuple vis.yMax = (c.getfloat('Params','yMax'))

6.36 vis.py

```
'''
vis.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''
import os
CPUs = os.environ['SLURM_JOB_CPUS_PER_NODE']
print "Number of cores: " + str(CPUs)
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
import mpld3
from mpld3 import plugins

getcontext().prec = 4
c = ConfigParser.ConfigParser()
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 = 0#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()
```

```python

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))
        m_val=np.max(np.abs(b)**2)
        #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=1e7)
            plt.title('rho(r) @ t=' + str(value*dt))
            plt.title(r'$\\rho \left( r,t \right),\,t=$' + str(value*dt))

            #plugins.connect(fig, plugins.MousePosition(fontsize=14))

            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__':
    try:
        delaunay('vort_arr_',0)
        stats.lsFit(0,evMaxVal,incr)
        hist3d.plot_hist_pcolor(0,evMaxVal,incr,'b')
        vort_traj('traj_plot',200)
    except:
        print "Unhandled error occurred. Blame Lee."
    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,200,0b110000)))
        gnd_proc.append(Process(target=image_gen_single,args=("wfc_0_const",i,200,0b110000)))
    while evImgList:
        i=evImgList.pop()
        ev_proc.append(Process(target=image_gen_single,args=("wfc_ev",i,200,0b101000)))
        #ev_proc.append(Process(target=mpld3.show,))
        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()):
        if int(CPUs) > len(mp.active_children()):
            print len(mp.active_children())
            try:
                p=proc.pop()
                p.start()
            except:
                print "Failed to execute ", p
```

6.37 py/vort.py File Reference

Classes

- class vort.Vortex
- class vort.VtxList

Namespaces

- vort

Functions

- def vort.__init__

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

Variables

- tuple vort.c = ConfigParser.ConfigParser()
- int vort.counter = 0
- tuple vort.current = self.element(pos-1)
- list vort.data = [ ]
- tuple vort.dcp = set(uid_c)
- tuple vort.dpc = set(uid_p)
- tuple vort.dt = (c.getfloat('Params','dt'))
- list vort.dtype = [('x',float),('y',float),('sign',int),('uid',int),('isOn',int)]
- tuple vort.dx = (c.getfloat('Params','dx'))
- tuple vort.evMaxVal = int(c.getfloat('Params','esteps'))
- tuple vort.gndMaxVal = int(c.getfloat('Params','gsteps'))
- int vort.i = 0
- tuple vort.incr = int(c.getfloat('Params','print_out'))
- tuple vort.index_r = vorts_c.idx_min_dist(vorts_p.element(i3))
- tuple vort.max_uid = vorts_p.max_uid()
- int vort.pos = 0
- int vort.pos_l = 0
- tuple vort.r = m.sqrt((self.x - vtx.x)∗∗2 + (self.y - vtx.y)∗∗2)
- vort.ret_idx = counter
- 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.v0c = vorts_c.element(index_r[0])
- tuple vort.v0p = vorts_p.element(i3)
- tuple vort.v1c = vorts_c.element(index_r[0])
- tuple vort.v_arr_c = genfromtxt('vort_lsq_' + str(i) + '.csv',delimiter=',')
- tuple vort.v_arr_c_coords = np.array([[a for a in v][:2] for v in v_arr_c])
- tuple vort.v_arr_c_sign = np.array([[a for a in v][2] for v in v_arr_c])
- tuple vort.v_arr_p = genfromtxt('vort_lsq_' + str(0) + '.csv',delimiter=',')

  v_arr_p=genfromtxt('vort_lsq_' + str(0) + '.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_p_sign = np.array([[a for a in v][2] for v in v_arr_p])
- int vort.val = 0
- tuple vort.vorts_c = VtxList()
- tuple vort.vorts_c_update = sorted(vorts_c.as_np(),key=lambda vtx: vtx[3])

- tuple vort.vorts_p = VtxList()
- vort.vtx = self.head
- 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.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_pos = self.vtx_uid(uid_i)
- list vort.vtx_pos_c = [ ]
- list vort.vtx_pos_p = [ ]
- tuple vort.xDim = int(c.getfloat('Params','xDim'))
- tuple vort.xMax = (c.getfloat('Params','xMax'))
- tuple vort.yDim = int(c.getfloat('Params','yDim'))
- tuple vort.yMax = (c.getfloat('Params','yMax'))

6.37.1 Class Documentation

6.37.1.1 class vort::Vortex

Definition at line 56 of file vort.py.

Collaboration diagram for vort.Vortex:

6.37.1.2 class vort::VtxList

Definition at line 90 of file vort.py.

Collaboration diagram for vort.VtxList:

6.38 vort.py

```
'''
vort.py - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
'''
##############################################################################
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: #Tracks indivisual vortices over time.
##############################################################################
##############################################################################
    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): #Vortex is trackable
##############################################################################
        self.isOn = isOn

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

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

##############################################################################
class VtxList: #Linked-list for tracking vortices
##############################################################################
##############################################################################
    def __init__(self):
##############################################################################
        self.head = None
        self.tail = None
        self.length = 0

##############################################################################
    def element(self,pos): #Get vtx at position 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): #Get vtx with identifier uid
##############################################################################
        vtx = self.head
        pos = 0
        while vtx.uid != uid:
            vtx = vtx.next
            pos = pos +1
        return [vtx,pos]

##############################################################################
    def max_uid(self): #Return position and value of largest uid
##############################################################################
        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): #Add a vtx at index, otherwise end
###############################################################################
        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): #Return numpy array with format x,y,sign,uid,isOn
###############################################################################
        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): #Write out CSV file as  x,y,sign,uid,isOn
###############################################################################
        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): #Closest vtx to self
###############################################################################
        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): #Remove vortices outside articificial boundary
###############################################################################
        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): #Swap uid between vtx
###############################################################################
        vtx_pos = self.vtx_uid(uid_i)
        self.remove(pos_i)
        self.add(vtx,index=pos_f)

```

```
################################################################################
    def vort_decrease(self,positions,vorts_p): #Turn off vortex timeline
################################################################################
        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): #Add new vtx to tracking
################################################################################
        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): #Performs the tracking
################################################################################
    #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()]

        #Check the difference between current and previous vtx data
        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)
```

6.39 src/ds.cc File Reference

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

Include dependency graph for ds.cc:

Functions

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

6.39.1 Function Documentation

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

Definition at line 42 of file ds.cc.

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

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

```
                                                       {
    Param p = newParam(t,d);
    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;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.39.1.2 void freeArray (Array ∗ *arr*)

Definition at line 52 of file ds.cc.

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

```
                             {
    free(arr->array);
    arr->array = NULL;
    arr->used = 0;
    arr->length = 0;
}
```

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

Definition at line 36 of file ds.cc.

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

Referenced by main().

```
                                                {
    arr->array = (Param*) malloc(initLen*sizeof(Param));
    arr->used = 0;
    arr->length = initLen;
}
```

Here is the caller graph for this function:

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

Definition at line 59 of file ds.cc.

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

Referenced by appendData().

```
00059                                   {
00060     Param p;
00061     strcpy(p.title,t);
00062     p.data = d;
00063     return p;
00064 }
```

Here is the caller graph for this function:

6.40 ds.cc

```
00001 /*** ds.cc - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
00012
00013 2. Redistributions in binary form must reproduce the above copyright
00014 notice, this list of conditions and the following disclaimer in the
00015 documentation and/or other materials provided with the distribution.
00016
00017 3. Neither the name of the copyright holder nor the names of its
00018 contributors may be used to endorse or promote products derived from
00019 this software without specific prior written permission.
00020
00021 THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
00022 "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
00023 LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
00024 PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
00025 HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
00026 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
00027 TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
00028 PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
00029 LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
00030 NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
00031 SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
00032 */
00033
00034 #include "../include/ds.h"
00035
00036 void initArr(Array *arr, size_t initLen){
00037     arr->array = (Param*) malloc(initLen*sizeof(Param));
00038     arr->used = 0;
00039     arr->length = initLen;
00040 }
00041
00042 void appendData(Array *arr, char* t, double d){
00043     Param p = newParam(t,d);
00044     if(arr->used == arr->length){
00045         arr->length *= 2;
00046         arr->array = (Param*)realloc(arr->array, arr->length*sizeof(
      Param));
00047     }
00048     arr->array[arr->used] = p;
00049     arr->used = arr->used + 1;
00050 }
00051
00052 void freeArray(Array *arr){
00053     free(arr->array);
00054     arr->array = NULL;
00055     arr->used = 0;
00056     arr->length = 0;
00057 }
00058
00059 Param newParam(char* t,double d){
00060     Param p;
```

```
    strcpy(p.title,t);
    p.data = d;
    return p;
}
```

6.41 src/fileIO.cc File Reference

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

Include dependency graph for fileIO.cc:

Namespaces

- FileIO

    Check source file for information on functions.

Functions

- double2 ∗ FileIO::readIn (char ∗, char ∗, int, int)
- int FileIO::readState (char ∗)
- void FileIO::writeOut (char ∗, char ∗, double2 ∗, int, int)
- void FileIO::writeOutDouble (char ∗, char ∗, double ∗, int, int)
- void FileIO::writeOutInt (char ∗, char ∗, int ∗, int, int)
- void FileIO::writeOutInt2 (char ∗, char ∗, int2 ∗, int, int)
- void FileIO::writeOutParam (char ∗, Array, char ∗)
- void FileIO::writeOutVortex (char ∗, char ∗, struct Tracker::Vortex ∗, int, int)

6.42 fileIO.cc

```
/*** fileIO.c - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```

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

namespace FileIO{

    /*
     * Reads datafile into memory.
     */
    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;
    }

    /*
     * Writes out the parameter file.
     */
    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);
    }

    /*
     * Writes out double2 complex data files.
     */
    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);
    }

    /*
     * Writes out double type data files.
     */
    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);
    }

    /*
     * Writes out int type data files.
     */
    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);
    }

    /*
     * Writes out int2 data type.
     */
    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);
    }

    /*
     * Writes out tracked vortex data.
     */
    void writeOutVortex(char* buffer, char *file, struct
Tracker::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);
    }

    /*
     * Opens and closes file. Nothing more. Nothing less.
     */
    int readState(char* name){
        FILE *f;
        f = fopen(name,"r");
        fclose(f);
        return 0;
    }
}
```

6.43 src/kernels.cu File Reference

```
#include "../include/constants.h"
#include <stdio.h>
```

Include dependency graph for kernels.cu:

Functions

- __global__ void angularOp (double omega, double dt, double2 *wfc, double *xpyypx, double2 *out)
- __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)
- __device__ double complexMagnitude (double2 in)
- __host__ __device__ double complexMagnitudeSquared (double2 in)
- __host__ __device__ double2 complexMultiply (double2 in1, double2 in2)
- __device__ double2 conjugate (double2 in)
- __global__ void energyCalc (double2 *wfc, double2 *op, double dt, double2 *energy, int gnd_state, int op_-space, double sqrt_omegaz_mass)
- __device__ unsigned int getBid3d3d ()
- __device__ unsigned int getGid3d3d ()
- __device__ unsigned int getTid3d3d ()

- __global__ void multipass (double2 ∗input, double2 ∗output, int pass)

  Routine for parallel summation.
- __global__ void pSum (double ∗in1, double ∗output, int pass)

  Routine for parallel summation.
- template<typename T >
  __global__ void pSumT (T ∗in1, T ∗output, int pass)

  Routine for parallel summation.
- __device__ double2 realCompMult (double scalar, double2 comp)
- __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.

Variables

- __constant__ double gDenConst = 2.535425438831619e-59

6.43.1 Function Documentation

6.43.1.1 __global__ void angularOp (double *omega,* double *dt,* double2 ∗ *wfc,* double ∗ *xpyypx,* double2 ∗ *out*)

Definition at line 153 of file kernels.cu.

References getGid3d3d(), and result.

```
                                                                                  {
    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;
}
```

Here is the call graph for this function:

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

Definition at line 88 of file kernels.cu.

References complexMultiply(), and conjugate().

Referenced by energyCalc().

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

Here is the call graph for this function:

Here is the caller graph for this function:

6.43.1.3 __global__ void cMult (double2 ∗ *in1,* double2 ∗ *in2,* double2 ∗ *out*)

Performs complex multiplication of in1 and in2, giving result as out.

Definition at line 96 of file kernels.cu.

References getGid3d3d(), and result.

```
00096                                                                    {
00097     double2 result;
00098     unsigned int gid = getGid3d3d();
00099     result.x = (in1[gid].x*in2[gid].x - in1[gid].y*in2[gid].y);
00100     result.y = (in1[gid].x*in2[gid].y + in1[gid].y*in2[gid].x);
00101     out[gid] = result;
00102 }
```

Here is the call graph for this function:

6.43.1.4 __global__ void cMultDensity (double2 ∗ *in1,* double2 ∗ *in2,* double2 ∗ *out,* double *dt,* double *mass,* double *omegaZ,* int *gstate,* int *N*)

Definition at line 104 of file kernels.cu.

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

```
00104
                          {
00105     double2 result;
00106     double gDensity;
00107     int tid = blockIdx.y*gridDim.x*blockDim.x + blockIdx.x*blockDim.x + threadIdx.x;
00108     gDensity = N*complexMagnitudeSquared(in2[tid])*4*
      HBAR*HBAR*PI*(4.67e-9/mass)*sqrt(mass*(omegaZ)/(2*PI*
      HBAR));
00109
00110     if(gstate == 0){
00111         double tmp = in1[tid].x*exp(-gDensity*(dt/HBAR) );
00112         result.x = (tmp)*in2[tid].x - (in1[tid].y)*in2[tid].y;
00113         result.y = (tmp)*in2[tid].y + (in1[tid].y)*in2[tid].x;
00114     }
00115     else{
00116         double2 tmp;
00117         tmp.x = in1[tid].x*cos(-gDensity*(dt/HBAR)) - in1[tid].y*sin(-gDensity*(
      dt/HBAR));
00118         tmp.y = in1[tid].y*cos(-gDensity*(dt/HBAR)) + in1[tid].x*sin(-gDensity*(
      dt/HBAR));
00119
00120         result.x = (tmp.x)*in2[tid].x - (tmp.y)*in2[tid].y;
00121         result.y = (tmp.x)*in2[tid].y + (tmp.y)*in2[tid].x;
00122     }
00123     out[tid] = result;
00124 }
```

Here is the call graph for this function:

6.43.1.5 __device__ double complexMagnitude (double2 *in*)

Definition at line 70 of file kernels.cu.

```
00070                                                 {
00071     return sqrt(in.x*in.x + in.y*in.y);
00072 }
```

6.43.1.6 __host__ __device__ double complexMagnitudeSquared (double2 *in*)

Definition at line 74 of file kernels.cu.

Referenced by cMultDensity(), and energyCalc().

```
00074                                                                  {
00075     return in.x*in.x + in.y*in.y;
00076 }
```

Here is the caller graph for this function:

6.43.1.7 __host__ __device__ double2 complexMultiply (double2 *in1,* double2 *in2*)

Definition at line 78 of file kernels.cu.

References result.

Referenced by braKetMult().

```
{
    double2 result;
    result.x = (in1.x*in2.x - in1.y*in2.y);
    result.y = (in1.x*in2.y + in1.y*in2.x);
    return result;
}
```

Here is the caller graph for this function:

6.43.1.8 __device__ double2 conjugate (double2 *in*)

Definition at line 56 of file kernels.cu.

References in(), and result.

Referenced by braKetMult().

```
{
    double2 result = in;
    result.y = -result.y;
    return result;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

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

Definition at line 193 of file kernels.cu.

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

```
{
    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;
}
```

Here is the call graph for this function:

6.43.1.10 __device__ unsigned int getBid3d3d ( )

Definition at line 46 of file kernels.cu.

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

6.43.1.11 __device__ unsigned int getGid3d3d ( )

Definition at line 41 of file kernels.cu.

Referenced by angularOp(), cMult(), energyCalc(), multipass(), pSum(), pSumT(), scalarDiv(), and scalarDiv_wfcNorm().

```
                               {
    return blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
threadIdx.y ) + blockIdx.x ) + threadIdx.x;
}
```

Here is the caller graph for this function:

6.43.1.12 __device__ unsigned int getTid3d3d ( )

Definition at line 52 of file kernels.cu.

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

6.43.1.13 __global__ void multipass (double2 ∗ *input,* double2 ∗ *output,* int *pass*)

Routine for parallel summation.

Can be looped over from host.

Definition at line 166 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
                                                                  {
    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];
    }
}
```

Here is the call graph for this function:

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

Routine for parallel summation.

Can be looped over from host.

Definition at line 239 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
                                                          {
             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];
             }
      }
```

Here is the call graph for this function:

6.43.1.15 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 220 of file kernels.cu.

References getGid3d3d(), and vis::i.

```
                                                                      {
             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];
             }
      }
```

Here is the call graph for this function:

6.43.1.16 __device__ double2 realCompMult (double *scalar,* double2 *comp*)

Definition at line 62 of file kernels.cu.

References result.

Referenced by energyCalc().

```
                                                          {
         double2 result;
         result.x = scalar * comp.x;
         result.y = scalar * comp.y;
         return result;
      }
```

Here is the caller graph for this function:

6.43.1.17 __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 130 of file kernels.cu.

References getGid3d3d(), and result.

```
                                                                                      {
    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;
}
```

Here is the call graph for this function:

6.43.1.18 __global__ void scalarDiv_wfcNorm (double2 ∗ *in,* double *dr,* double2 ∗ *pSum,* double2 ∗ *out*)

As above, but normalises for wfc.

Definition at line 142 of file kernels.cu.

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

```
                                                                                                              {
    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;
}
```

Here is the call graph for this function:

6.43.2 Variable Documentation

6.43.2.1 __constant__ double gDenConst = 2.535425438831619e-59

Definition at line 38 of file kernels.cu.

Referenced by energyCalc().

6.44 kernels.cu

```
/*** kernels.cu - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.
```

```

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "../include/constants.h"
#include <stdio.h>


__constant__ double gDenConst = 2.535425438831619e-59;//Evaluted in MATLAB:
 HBAR*(4.67e-9)*sqrt(8*HBAR*PI)*;
//inline __device__ unsigned int getGid3d3d(){

__device__ unsigned int getGid3d3d(){
    return blockDim.x * ( ( blockDim.y * ( ( blockIdx.z * blockDim.z + threadIdx.z ) + blockIdx.y ) +
threadIdx.y ) + blockIdx.x ) + threadIdx.x;
}

//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 = 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];
        }
}



//##############################################################################################
//##############################################################################################
```

6.45 src/minions.cc File Reference

`#include "../include/minions.h"`

Include dependency graph for minions.cc:

Namespaces

- Minions

Functions

- double2 Minions::complexDiv (double2 num, double2 den)
- double Minions::complexMag (double2 in)
- double Minions::complexMag2 (double2 in)
- double2 Minions::complexMult (double2 in1, double2 in2)

- double2 Minions::complexScale (double2 comp, double scale)
- double2 Minions::conj (double2 c)
- void Minions::coordSwap (struct Tracker::Vortex *vCoords, int src, int dest)
- double Minions::fInvSqRt (double)

    id magic hackery
- double Minions::maxValue (double *, int)
- double Minions::minValue (double *, int)
- double Minions::psi2 (double2)
- double Minions::sumAvg (double *in, int len)

6.46 minions.cc

```
/*** minions.cc - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "../include/minions.h"

namespace Minions{
    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 Tracker::Vortex *vCoords, int src, int dest){
        struct Tracker::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;
        }
    }
*/
```

6.47 src/multigpu.cu File Reference

6.48 multigpu.cu

6.49 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

- void delta_define (double ∗x, double ∗y, double x0, double y0, double ∗delta)
- 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.
- 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)
- int initialise (double omegaX, double omegaY, int N)
- int isError (int result, char ∗c)
- int main (int argc, char ∗∗argv)
- void optLatSetup (struct Tracker::Vortex centre, double ∗V, struct Tracker::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.
- int parseArgs (int argc, char ∗∗argv)
- void parSum (double2 ∗gpuWfc, double2 ∗gpuParSum, int xDim, int yDim, int threads)
- template<typename T >
  void parSum (T ∗gpuToSumArr, T ∗gpuParSum, int xDim, int yDim, int threads)

Variables

- double a0x
- double a0y
- double angle_sweep
- char buffer [100]
- int device
- double gammaY
- int kick_it
- double omega
- Params ∗ paramS
- Array params
- double Rxy
- double sepMinEpsilon =0.0
- double timeTotal
- int verbose
- double x0_shift
- double y0_shift

6.49.1 Function Documentation

6.49.1.1 void delta_define (double ∗ *x*, double ∗ *y*, double *x0*, double *y0*, double ∗ *delta*)

Definition at line 865 of file split_op.cu.

References dt, dx, EV_opt, HBAR, vis::i, V, xDim, and yDim.

```
                                                                              {
    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)));
        }
    }
}
```

6.49.1.2 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 655 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;
*/

}
```

6.49.1.3 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 293 of file split_op.cu.

References a_s, angle_sweep, appendData(), buffer, Tracker::Vortex::coords, dt, dx, dy, EV, EV_opt, Tracker::findVortex(), gdt, HBAR, vis::i, interaction, kick_it, laser_power, mass, omega, omegaX, omegaY, omegaZ, optLatSetup(), parSum(), PI, plan_1d, plan_2d, Minions::psi2(), result, sepAvg(), Tracker::Vortex::sign, Minions::sumAvg(), V, V_gpu, V_opt, Tracker::vortAngle(), Tracker::vortArrange(), Tracker::vortCentre(), Tracker::vortPos(), Tracker::vortSepAvg(), wfc, Tracker::Vortex::wind, write_it, FileIO::writeOut(), FileIO::writeOutDouble(), FileIO::writeOutParam(), FileIO::writeOutVortex(), x, xDim, xi, y, 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] = Minions::psi2(wfc[ ( (
yDim/2) - (gridSum/2) + j )*yDim  + ( (xDim/2)  - (gridSum/2) + k )]);
        }
    }
    xi = 1/sqrt(8*PI*a_s*Minions::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 Tracker::Vortex central_vortex; //vortex closest to the central position
    double vort_angle; //Angle of vortex lattice. Add to optical lattice for alignment.
    struct Tracker::Vortex *vortCoords = NULL; //array of vortex coordinates from
 vortexLocation 1's
    struct Tracker::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] = Tracker::findVortex(vortexLocation, wfc, 1e-4, xDim, x, i);
                    if(i==0){
                        vortCoords = (struct Tracker::Vortex*) malloc(sizeof(struct Tracker::Vortex)*(2*num_vortices[0]));
                        vortCoordsP = (struct Tracker::Vortex*) malloc(sizeof(struct Tracker::Vortex)*(2*num_vortices[0]));
                        Tracker::vortPos(vortexLocation, vortCoords, xDim, wfc);
                        central_vortex = Tracker::vortCentre(vortCoords, num_vortices[0], xDim);
                        vort_angle = Tracker::vortAngle(vortCoords,central_vortex, num_vortices[0]);
                        appendData(&params,"Vort_angle",vort_angle);
                        optLatSetup(central_vortex, V, vortCoords, num_vortices[0], vort_angle + PI*angle_sweep/180.0, laser_power*HBAR*sqrt(omegaX*omegaY), V_opt, x, y);
                        sepAvg = Tracker::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);
                        }
                        FileIO::writeOutDouble(buffer,"V_opt_1",V_opt,xDim*yDim,0);
                        FileIO::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]);
                        FileIO::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]);
                        Tracker::vortPos(vortexLocation, vortCoords, xDim,wfc);
                    }
                    else{
                        Tracker::vortPos(vortexLocation, vortCoords, xDim,wfc);
                        Tracker::vortArrange(vortCoords, vortCoordsP, num_vortices[0]);
                    }
            /*      num_latt_max = Tracker::findOLMaxima(olMaxLocation, V_opt, 1e-4, xDim, x);
                    if(num_latt_max == num_vortices[0]){
                        olCoords = (int2*) malloc(sizeof(int2)*num_latt_max);
                        Tracker::olPos(olMaxLocation, olCoords, xDim);
                        vortOLSigma = Tracker::sigVOL(vortCoords, olCoords, x, num_latt_max);
                        FileIO::writeOutInt2(buffer, "opt_max_arr", olCoords, num_latt_max, i);
                        free(olCoords);
                    }*/
                    FileIO::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)
                FileIO::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: Cuda memcpy EV into GPU\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;
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.49.1.4 int initialise (double *omegaX*, double *omegaY*, int *N*)

Definition at line 64 of file split_op.cu.

References a0x, a0y, a_s, appendData(), 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, FileIO::writeOut(), FileIO-::writeOutDouble(), x, xDim, xMax, xp, xPy, xPy_gpu, y, yDim, yMax, yp, yPx, and yPx_gpu.

Referenced by main().

```
                                                        {
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
    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);
    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++ ){
            Phi[(i*yDim + j)] = fmod(l*atan2(y[j], x[i]),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]);

            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];

            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);
00233             EyPx[(i*yDim + j)].x=cos(-omega*omegaX*yPx[(i*xDim + j)]*
      dt);
00234             EyPx[(i*yDim + j)].y=sin(-omega*omegaX*yPx[(i*xDim + j)]*
      dt);
00235
00236             sum+=sqrt(wfc[(i*xDim + j)].x*wfc[(i*xDim + j)].x + wfc[(i*xDim + j)].
      y*wfc[(i*xDim + j)].y);
00237         }
00238     }
00239     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00240     //hdfWriteDouble(xDim, V, 0, "V_0");
00241     //hdfWriteComplex(xDim, wfc, 0, "wfc_0");
00242     FileIO::writeOutDouble(buffer,"V",V,xDim*yDim,0);
00243     //FileIO::writeOutDouble(buffer,"V_opt",V_opt,xDim*yDim,0);
00244     FileIO::writeOutDouble(buffer,"K",K,xDim*yDim,0);
00245     FileIO::writeOutDouble(buffer,"xPy",xPy,xDim*yDim,0);
00246     FileIO::writeOutDouble(buffer,"yPx",yPx,xDim*yDim,0);
00247     FileIO::writeOut(buffer,"WFC",wfc,xDim*yDim,0);
00248     FileIO::writeOut(buffer,"ExPy",ExPy,xDim*yDim,0);
00249     FileIO::writeOut(buffer,"EyPx",EyPx,xDim*yDim,0);
00250     FileIO::writeOutDouble(buffer,"Phi",Phi,xDim*yDim,0);
00251     FileIO::writeOutDouble(buffer,"r",r,xDim*yDim,0);
00252     FileIO::writeOutDouble(buffer,"x",x,xDim,0);
00253     FileIO::writeOutDouble(buffer,"y",y,yDim,0);
00254     FileIO::writeOutDouble(buffer,"px",xp,xDim,0);
00255     FileIO::writeOutDouble(buffer,"py",yp,yDim,0);
00256     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00257
00258     //free(V);
00259     free(K); free(r); //free(Phi);
00260
00261     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00262
00263     sum=sqrt(sum*dx*dy);
00264     //#pragma omp parallel for reduction(+:sum) private(j)
00265     for (i = 0; i < xDim; i++){
00266         for (j = 0; j < yDim; j++){
00267             wfc[(i*yDim + j)].x = (wfc[(i*yDim + j)].x)/(sum);
00268             wfc[(i*yDim + j)].y = (wfc[(i*yDim + j)].y)/(sum);
00269         }
00270     }
00271
00272     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00273
00274     result = cufftPlan2d(&plan_2d, xDim, yDim, CUFFT_Z2Z);
00275     if(result != CUFFT_SUCCESS){
00276         printf("Result:=%d\n",result);
00277         printf("Error: Could not execute cufftPlan2d(%s ,%d, %d).\n", "plan_2d", (unsigned int)xDim, (
      unsigned int)yDim);
00278         return -1;
00279     }
00280
00281     result = cufftPlan1d(&plan_1d, xDim, CUFFT_Z2Z, yDim);
00282     if(result != CUFFT_SUCCESS){
00283         printf("Result:=%d\n",result);
00284         printf("Error: Could not execute cufftPlan3d(%s ,%d ,%d ).\n", "plan_1d", (unsigned int)xDim, (
      unsigned int)yDim);
00285         return -1;
00286     }
00287
00288     //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//
00289
00290     return 0;
00291 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.49.1.5 int isError (int *result,* char ∗ *c*)

Definition at line 58 of file split_op.cu.

References result.

```
00058                                 {
00059     if(result!=0){printf("Error has occurred for method %s with return type %d\n",
      c,result);
00060         exit(result);
00061     }
00062     return result;
00063 }
```

6.49.1.6 int main (int *argc*, char ∗∗ *argv*)

Definition at line 876 of file split_op.cu.

References ang_mom, atoms, buffer, device, EK, err, esteps, EV, evolve(), ExPy, EyPx, GK, gpe, gsteps, GV, initArr(), initialise(), K_gpu, omegaX, omegaY, par_sum, parseArgs(), print, read_wfc, FileIO::readIn(), timeTotal, V_gpu, V_opt, wfc, wfc_gpu, FileIO::writeOutDouble(), FileIO::writeOutParam(), x, xDim, xPy, xPy_gpu, y, yDim, yPx, and yPx_gpu.

```
00876                                  {
00877 
00878     time_t start,fin;
00879     time(&start);
00880     printf("Start: %s\n", ctime(&start));
00881     initArr(&params,32);
00882     //appendData(&params,ctime(&start),0.0);
00883     parseArgs(argc,argv);
00884     cudaSetDevice(device);
00885     //************************************************************//
00886     /*
00887     * Initialise the Params data structure to track params and variables
00888     */
00889     //************************************************************//
00890     //paramS = (Params *) malloc(sizeof(Params));
00891     //strcpy(paramS->data,"INIT");
00892     //paramS->next=NULL;
00893 
00894     initialise(omegaX,omegaY,atoms);
00895     timeTotal = 0.0;
00896     //************************************************************//
00897     /*
00898     * Groundstate finder section
00899     */
00900     //************************************************************//
00901     FileIO::writeOutParam(buffer, params, "Params.dat");
00902     if(read_wfc == 1){
00903         printf("Loading wavefunction...");
00904         wfc=FileIO::readIn("wfc_load","wfci_load",xDim,
      yDim);
00905         printf("Wavefunction loaded.\n");
00906     }
00907 
00908     double2 ph;
00909     double x_0,y_0;
00910     x_0 = 0;//(0.5*xDim)*dx;
00911     y_0 = 0;//(0.5*yDim)*dy;
00912 /*  for(int i=0; i < xDim; i++ ){
00913         for(int j=0; j < yDim; j++ ){
00914             ph.x = cos( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
00915             ph.y = -sin( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
00916             wfc[(i*yDim + j)] = Minions::complexMult( wfc[(i*yDim + j)], ph );
00917         }
00918     }
00919     printf("l=%e\n",l);
00920 */  if(gsteps > 0){
00921         err=cudaMemcpy(K_gpu, GK, sizeof(cufftDoubleComplex)*xDim*
      yDim, cudaMemcpyHostToDevice);
00922         if(err!=cudaSuccess)
00923             exit(1);
00924         err=cudaMemcpy(V_gpu, GV, sizeof(cufftDoubleComplex)*xDim*
      yDim, cudaMemcpyHostToDevice);
00925         if(err!=cudaSuccess)
00926             exit(1);
00927         err=cudaMemcpy(xPy_gpu, xPy, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice)
      ;
00928         if(err!=cudaSuccess)
00929             exit(1);
00930         err=cudaMemcpy(yPx_gpu, yPx, sizeof(double)*xDim*yDim, cudaMemcpyHostToDevice)
      ;
00931         if(err!=cudaSuccess)
00932             exit(1);
00933         err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00934         if(err!=cudaSuccess)
00935             exit(1);
00936 
00937         evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
      xPy_gpu, par_sum, xDim*yDim, gsteps, 128, 0, ang_mom,
      gpe, print, atoms, 0);
00938             cudaMemcpy(wfc, wfc_gpu, sizeof(cufftDoubleComplex)*xDim*
      yDim, cudaMemcpyDeviceToHost);
00939         }
00940 
00941         free(GV); free(GK); free(xPy); free(yPx);
00942 
00943         //************************************************************//
00944         /*
00945         * Evolution
00946         */
00947         //************************************************************//
00948         if(esteps > 0){
00949             err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00950             if(err!=cudaSuccess)
00951                 exit(1);
00952             err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00953             if(err!=cudaSuccess)
00954                 exit(1);
00955             err=cudaMemcpy(xPy_gpu, ExPy, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00956             if(err!=cudaSuccess)
00957                 exit(1);
00958             err=cudaMemcpy(yPx_gpu, EyPx, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00959             if(err!=cudaSuccess)
00960                 exit(1);
00961             err=cudaMemcpy(K_gpu, EK, sizeof(cufftDoubleComplex)*xDim*
      yDim, cudaMemcpyHostToDevice);
00962             if(err!=cudaSuccess)
00963                 exit(1);
00964             err=cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*
      yDim, cudaMemcpyHostToDevice);
00965             if(err!=cudaSuccess)
00966                 exit(1);
00967             err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00968             if(err!=cudaSuccess)
00969                 exit(1);
00970 
00971             //delta_define(x, y, (523.6667 - 512 + x0_shift)*dx, (512.6667 - 512  + y0_shift)*dy, V_opt);
00972             FileIO::writeOutDouble(buffer,"V_opt",V_opt,
      xDim*yDim,0);
00973             evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
      xPy_gpu, par_sum, xDim*yDim, esteps, 128, 1, ang_mom,
      gpe, print, atoms, 0);
00974 
00975         }
00976         free(EV); free(EK); free(ExPy); free(EyPx);
00977         free(x);free(y);
00978         cudaFree(wfc_gpu); cudaFree(K_gpu); cudaFree(V_gpu); cudaFree(
      yPx_gpu); cudaFree(xPy_gpu); cudaFree(par_sum);
00979 
00980         time(&fin);
00981         //appendData(&params,ctime(&fin),0.0);
00982         printf("Finish: %s\n", ctime(&fin));
00983         printf("Total time: %ld seconds\n ",(long)fin-start);
00984         //appendData(&params,"t_duration",fin-start);
00985         return 0;
00986 }
```

Here is the call graph for this function:

6.49.1.7 void optLatSetup (struct Tracker::Vortex *centre*, double ∗ *V*, struct Tracker::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 596 of file split_op.cu.

References appendData(), buffer, Tracker::Vortex::coords, dt, dx, dy, EV_opt, HBAR, vis::i, observables::k_mag, PI, sepMinEpsilon, Tracker::vortSepAvg(), FileIO::writeOut(), xDim, and yDim.

Referenced by evolve().

```
00596                                                                                   {
00597     int i,j;
00598     double sepMin = Tracker::vortSepAvg(vArray,centre,num_vortices);
00599     sepMin = sepMin*(1 + sepMinEpsilon);
00600     appendData(&params,"Vort_sep",(double)sepMin);
00601     /*
00602     * Defining the necessary k vectors for the optical lattice
00603     */
00604     double k_mag = ((2*PI/(sepMin*dx))/2)*(2/sqrt(3)); // Additional /2 as a result of lambda/2
       period
00605     double2* k = (double2*) malloc(sizeof(double2)*3);
00606     appendData(&params,"kmag",(double)k_mag);
00607     k[0].x = k_mag * cos(0*PI/3 + theta_opt);
00608     k[0].y = k_mag * sin(0*PI/3 + theta_opt);
00609     k[1].x = k_mag * cos(2*PI/3 + theta_opt);
00610     k[1].y = k_mag * sin(2*PI/3 + theta_opt);
00611     k[2].x = k_mag * cos(4*PI/3 + theta_opt);
00612     k[2].y = k_mag * sin(4*PI/3 + theta_opt);
00613 
00614     double2 *r_opt = (double2*) malloc(sizeof(double2)*xDim);
00615 
00616 /*  for (int ii = 0; ii < xDim; ++ii){
00617         r_opt[ii].x = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.x)/(xDim-1);
00618         r_opt[ii].y = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.y)/(yDim-1);
00619     }
00620 */
00621     FileIO::writeOut(buffer,"r_opt",r_opt,xDim,0);
00622     appendData(&params,"k[0].x",(double)k[0].x);
00623     appendData(&params,"k[0].y",(double)k[0].y);
00624     appendData(&params,"k[1].x",(double)k[1].x);
00625     appendData(&params,"k[1].y",(double)k[1].y);
00626     appendData(&params,"k[2].x",(double)k[2].x);
00627     appendData(&params,"k[2].y",(double)k[2].y);
00628 
00629     double x_shift = dx*(9+(0.5*xDim-1) - centre.coords.x);//sin(theta_opt)*(sepMin);
00630     double y_shift = dy*(0+(0.5*yDim-1) - centre.coords.y);//cos(theta_opt)*(sepMin);
00631 
00632     printf("Xs=%e\nYs=%e\n",x_shift,y_shift);
00633 
00634     //#pragma omp parallel for private(j)
00635     for ( j=0; j<yDim; ++j ){
00636         for ( i=0; i<xDim; ++i ){
00637             v_opt[j*xDim + i] = intensity*(
00638                                 pow( abs( cos( k[0].x*( x[i] + x_shift ) + k[0].
      y*( y[j] + y_shift ) ) ), 2)
00639                           + pow( abs( cos( k[1].x*( x[i] + x_shift ) + k[1].y*(
      y[j] + y_shift ) ) ), 2)
00640                           + pow( abs( cos( k[2].x*( x[i] + x_shift ) + k[2].y*(
      y[j] + y_shift ) ) ), 2)
00641             /*                  pow( abs( cos( k[0].x*( r_opt[i].x + x_shift ) + k[0].y*( r_opt[j].y +
       y_shift ) ) ), 2)
00642                           + pow( abs( cos( k[1].x*( r_opt[i].x + x_shift ) + k[1].y*( r_opt[j].y + y_shift )
       ) ), 2)
00643                           + pow( abs( cos( k[2].x*( r_opt[i].x + x_shift ) + k[2].y*( r_opt[j].y + y_shift )
       ) ), 2)
00644             */          );
00645             EV_opt[(j*xDim + i)].x=cos( -(V[(j*xDim + i)] + v_opt[j*xDim +
      i])*(dt/(2*HBAR)));
00646             EV_opt[(j*xDim + i)].y=sin( -(V[(j*xDim + i)] + v_opt[j*xDim + i])*(
      dt/(2*HBAR)));
00647         }
00648     }
00649 
00650 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

6.49.1.8 int parseArgs (int *argc*, char ∗∗ *argv*)

Definition at line 714 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, sepMinEpsilon, write_it, x0_shift, xDim, y0_shift, and yDim.

Referenced by main().

```
00714                                 {
00715     int opt;
00716     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:S:")) != -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 'S':
                sepMinEpsilon = atof(optarg);
                printf("Argument for sepMinEpsilon is %lf\n",sepMinEpsilon);
                appendData(&params,"sepMinEpsilon",sepMinEpsilon);
                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:

6.49.1.9 void parSum (double2 ∗ *gpuWfc*, double2 ∗ *gpuParSum*, int *xDim*, int *yDim*, int *threads*)

Definition at line 572 of file split_op.cu.

References dx, dy, threads, and yDim.

Referenced by evolve().

```
                                                                                    {
        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);
```

```
00580             }
00581             else{
00582                 multipass<<<block,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0],pass
      );
00583             }
00584             grid_tmp /= threads;
00585             block = (int) ceil((double)grid_tmp/threads);
00586             pass++;
00587         }
00588         thread_tmp = grid_tmp;
00589         multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00590         scalarDiv_wfcNorm<<<grid,threads>>>(gpuWfc, dx*dy, gpuParSum, gpuWfc);
00591 }
```

Here is the caller graph for this function:

6.49.1.10 template<typename T > void parSum (T * *gpuToSumArr*, T * *gpuParSum*, int *xDim*, int *yDim*, int *threads*)

Definition at line 692 of file split_op.cu.

References dx, dy, threads, and yDim.

```
00692                                                                                                 {
00693                 int grid_tmp = xDim*yDim;
00694                 int block = grid_tmp/threads;
00695                 int thread_tmp = threads;
00696                 int pass = 0;
00697                 while((double)grid_tmp/threads > 1.0){
00698                         if(grid_tmp == xDim*yDim){
00699                                 multipass<<<block,threads,threads*sizeof(T)>>>(&gpuToSumArr[0],&gpuParSum[0
      ],pass);
00700                         }
00701                         else{
00702                                 multipass<<<block,thread_tmp,thread_tmp*sizeof(T)>>>(&gpuParSum[0],&
      gpuParSum[0],pass);
00703                         }
00704                         grid_tmp /= threads;
00705                         block = (int) ceil((double)grid_tmp/threads);
00706                         pass++;
00707                 }
00708                 thread_tmp = grid_tmp;
00709                 multipass<<<1,thread_tmp,thread_tmp*sizeof(double2)>>>(&gpuParSum[0],&gpuParSum[0], pass);
00710                 scalarDiv_wfcNorm<<<grid,threads>>>(gpuToSumArr, dx*dy, gpuParSum, gpuToSumArr);
00711 }
```

6.49.2 Variable Documentation

6.49.2.1 double a0x

Definition at line 54 of file split_op.cu.

Referenced by initialise().

6.49.2.2 double a0y

Definition at line 54 of file split_op.cu.

Referenced by initialise().

6.49.2.3 double angle_sweep

Definition at line 49 of file split_op.cu.

Referenced by evolve(), and parseArgs().

6.49.2.4 char buffer[100]

Definition at line 42 of file split_op.cu.

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

6.49.2.5 int device

Definition at line 44 of file split_op.cu.

Referenced by main(), and parseArgs().

6.49.2.6 double gammaY

Definition at line 46 of file split_op.cu.

Referenced by initialise(), and parseArgs().

6.49.2.7 int kick_it

Definition at line 45 of file split_op.cu.

Referenced by evolve(), and parseArgs().

6.49.2.8 double omega

Definition at line 47 of file split_op.cu.

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

6.49.2.9 Params∗ paramS

Definition at line 50 of file split_op.cu.

6.49.2.10 Array params

Definition at line 51 of file split_op.cu.

6.49.2.11 double Rxy

Definition at line 53 of file split_op.cu.

Referenced by initialise().

6.49.2.12 double sepMinEpsilon =0.0

Definition at line 55 of file split_op.cu.

Referenced by optLatSetup(), and parseArgs().

6.49.2.13 double timeTotal

Definition at line 48 of file split_op.cu.

Referenced by main().

6.49.2.14 int verbose

Definition at line 43 of file split_op.cu.

6.49.2.15 double x0_shift

Definition at line 52 of file split_op.cu.

Referenced by parseArgs().

6.49.2.16 double y0_shift

Definition at line 52 of file split_op.cu.

Referenced by parseArgs().

6.50 split_op.cu

```
/*
* split_op.cu - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#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;
double sepMinEpsilon=0.0;
/* 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);
    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++ ){
            Phi[(i*yDim + j)] = fmod(l*atan2(y[j], x[i]),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]);

            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];

            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");
    FileIO::writeOutDouble(buffer,"V",V,xDim*yDim,0);
    //FileIO::writeOutDouble(buffer,"V_opt",V_opt,xDim*yDim,0);
    FileIO::writeOutDouble(buffer,"K",K,xDim*yDim,0);
    FileIO::writeOutDouble(buffer,"xPy",xPy,xDim*yDim,0);
    FileIO::writeOutDouble(buffer,"yPx",yPx,xDim*yDim,0);
    FileIO::writeOut(buffer,"WFC",wfc,xDim*yDim,0);
    FileIO::writeOut(buffer,"ExPy",ExPy,xDim*yDim,0);
    FileIO::writeOut(buffer,"EyPx",EyPx,xDim*yDim,0);
    FileIO::writeOutDouble(buffer,"Phi",Phi,xDim*yDim,0);
    FileIO::writeOutDouble(buffer,"r",r,xDim*yDim,0);
    FileIO::writeOutDouble(buffer,"x",x,xDim,0);
    FileIO::writeOutDouble(buffer,"y",y,yDim,0);
    FileIO::writeOutDouble(buffer,"px",xp,xDim,0);
    FileIO::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] = Minions::psi2(wfc[ ( (yDim/2) - (gridSum/2) + j )*yDim  + ( (xDim/2)  - (gridSum/2) + k )]);
        }
    }
    xi = 1/sqrt(8*PI*a_s*Minions::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 Tracker::Vortex central_vortex; //vortex closest to the central position
    double vort_angle; //Angle of vortex lattice. Add to optical lattice for alignment.
    struct Tracker::Vortex *vortCoords = NULL; //array of vortex coordinates from vortexLocation 1's
    struct Tracker::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] = Tracker::findVortex(vortexLocation, wfc, 1e-4, xDim, x, i);
                    if(i==0){
                        vortCoords = (struct Tracker::Vortex*) malloc(sizeof(struct Tracker::Vortex)*(2*num_vortices[0]));
                        vortCoordsP = (struct Tracker::Vortex*) malloc(sizeof(struct Tracker::Vortex)*(2*num_vortices[0]));
                        Tracker::vortPos(vortexLocation, vortCoords, xDim, wfc);
                        central_vortex = Tracker::vortCentre(vortCoords, num_vortices[0], xDim);
                        vort_angle = Tracker::vortAngle(vortCoords,central_vortex, num_vortices[0]);
                        appendData(&params,"Vort_angle",vort_angle);
                        optLatSetup(central_vortex, V, vortCoords, num_vortices[0], vort_angle + PI*angle_sweep/180.0, laser_power*HBAR*sqrt(omegaX*omegaY), V_opt, x, y);
                        sepAvg = Tracker::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);
                        }
                        FileIO::writeOutDouble(buffer,"V_opt_1",
V_opt,xDim*yDim,0);
                        FileIO::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]);
                        FileIO::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]
);
                        Tracker::vortPos(vortexLocation, vortCoords,
xDim,wfc);
                    }
                    else{
                        Tracker::vortPos(vortexLocation, vortCoords,
xDim,wfc);
                        Tracker::vortArrange(vortCoords, vortCoordsP, num_vortices[0]);
                    }
            /*      num_latt_max = Tracker::findOLMaxima(olMaxLocation, V_opt, 1e-4, xDim, x);
                    if(num_latt_max == num_vortices[0]){
                        olCoords = (int2*) malloc(sizeof(int2)*num_latt_max);
                        Tracker::olPos(olMaxLocation, olCoords, xDim);
                        vortOLSigma = Tracker::sigVOL(vortCoords, olCoords, x, num_latt_max);
                        FileIO::writeOutInt2(buffer, "opt_max_arr", olCoords, num_latt_max, i);
                        free(olCoords);
                    }*/
                    FileIO::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)
                FileIO::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: Cuda memcpy EV into GPU\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 Tracker::Vortex centre, double* V, struct Tracker::Vortex *vArray, int num_vortices, double theta_opt, double intensity, double* v_opt, double *x, double *y){
    int i,j;
    double sepMin = Tracker::vortSepAvg(vArray,centre,num_vortices);
    sepMin = sepMin*(1 + sepMinEpsilon);
    appendData(&params,"Vort_sep",(double)sepMin);
    /*
    * Defining the necessary k vectors for the optical lattice
    */
    double k_mag = ((2*PI/(sepMin*dx))/2)*(2/sqrt(3)); // Additional /2 as a result of lambda/2 period
    double2* k = (double2*) malloc(sizeof(double2)*3);
    appendData(&params,"kmag",(double)k_mag);
    k[0].x = k_mag * cos(0*PI/3 + theta_opt);
    k[0].y = k_mag * sin(0*PI/3 + theta_opt);
    k[1].x = k_mag * cos(2*PI/3 + theta_opt);
    k[1].y = k_mag * sin(2*PI/3 + theta_opt);
    k[2].x = k_mag * cos(4*PI/3 + theta_opt);
    k[2].y = k_mag * sin(4*PI/3 + theta_opt);

    double2 *r_opt = (double2*) malloc(sizeof(double2)*xDim);

/*  for (int ii = 0; ii < xDim; ++ii){
        r_opt[ii].x = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.x)/(xDim-1);
        r_opt[ii].y = 0.0 + (xDim/sepMin)*PI*(ii-centre.coords.y)/(yDim-1);
    }
*/
    FileIO::writeOut(buffer,"r_opt",r_opt,xDim,0);
    appendData(&params,"k[0].x",(double)k[0].x);
    appendData(&params,"k[0].y",(double)k[0].y);
    appendData(&params,"k[1].x",(double)k[1].x);
    appendData(&params,"k[1].y",(double)k[1].y);
    appendData(&params,"k[2].x",(double)k[2].x);
    appendData(&params,"k[2].y",(double)k[2].y);

    double x_shift = dx*(9+(0.5*xDim-1) - centre.coords.x);//sin(theta_opt)*(sepMin);
    double y_shift = dy*(0+(0.5*yDim-1) - centre.coords.y);//cos(theta_opt)*(sepMin);

    printf("Xs=%e\nYs=%e\n",x_shift,y_shift);

    //#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( k[0].x*( x[i] + x_shift ) + k[0].y*( y[j] + y_shift ) ) ), 2)
                        + pow( abs( cos( k[1].x*( x[i] + x_shift ) + k[1].y*( y[j] + y_shift ) ) ), 2)
                        + pow( abs( cos( k[2].x*( x[i] + x_shift ) + k[2].y*( y[j] + y_shift ) ) ), 2)
            /*                pow( abs( cos( k[0].x*( r_opt[i].x + x_shift ) + k[0].y*( r_opt[j].y + y_shift ) ) ), 2)
                        + pow( abs( cos( k[1].x*( r_opt[i].x + x_shift ) + k[1].y*( r_opt[j].y + y_shift ) ) ), 2)
                        + pow( abs( cos( k[2].x*( r_opt[i].x + x_shift ) + k[2].y*( r_opt[j].y + 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:S:")) != -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 'S':
                sepMinEpsilon = atof(optarg);
                printf("Argument for sepMinEpsilon is %lf\n",sepMinEpsilon);
                appendData(&params,"sepMinEpsilon",sepMinEpsilon);
                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
    */
    //******************************************************************//
    FileIO::writeOutParam(buffer, params, "Params.dat");
    if(read_wfc == 1){
        printf("Loading wavefunction...");
        wfc=FileIO::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( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
            ph.y = -sin( fmod( 0*atan2( y[j] - y_0, x[i] - x_0 ), 2*PI) );
            wfc[(i*yDim + j)] = Minions::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)
00963             exit(1);
00964         err=cudaMemcpy(V_gpu, EV, sizeof(cufftDoubleComplex)*xDim*yDim,
      cudaMemcpyHostToDevice);
00965         if(err!=cudaSuccess)
00966             exit(1);
00967         err=cudaMemcpy(wfc_gpu, wfc, sizeof(cufftDoubleComplex)*
      xDim*yDim, cudaMemcpyHostToDevice);
00968         if(err!=cudaSuccess)
00969             exit(1);
00970 
00971         //delta_define(x, y, (523.6667 - 512 + x0_shift)*dx, (512.6667 - 512  + y0_shift)*dy, V_opt);
00972         FileIO::writeOutDouble(buffer,"V_opt",V_opt,
      xDim*yDim,0);
00973         evolve(wfc_gpu, K_gpu, V_gpu, yPx_gpu,
      xPy_gpu, par_sum, xDim*yDim, esteps, 128, 1, ang_mom,
      gpe, print, atoms, 0);
00974 
00975     }
00976     free(EV); free(EK); free(ExPy); free(EyPx);
00977     free(x);free(y);
00978     cudaFree(wfc_gpu); cudaFree(K_gpu); cudaFree(V_gpu); cudaFree(
      yPx_gpu); cudaFree(xPy_gpu); cudaFree(par_sum);
00979 
00980     time(&fin);
00981     //appendData(&params,ctime(&fin),0.0);
00982     printf("Finish: %s\n", ctime(&fin));
00983     printf("Total time: %ld seconds\n ",(long)fin-start);
00984     //appendData(&params,"t_duration",fin-start);
00985     return 0;
00986 }
```

6.51 src/srt.cc File Reference

Functions

- double sepAvg (int2 ∗vArray, int2 centre, int length)

6.51.1 Function Documentation

6.51.1.1 double sepAvg (int2 ∗ *vArray,* int2 *centre,* int *length*)

Definition at line 34 of file srt.cc.

References vis::i, and result.

Referenced by evolve().

```
00034                                                         {
00035     double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
00036     for (int i=0; i<length; ++i){
00037         result += sqrt( pow(centre.x - v_array[i].x,2) + pow(centre.y - v_array[i].y,2));
00038     }
00039     return result/length;
00040 }
```

Here is the caller graph for this function:

6.52 srt.cc

```
00001 /*** srt.cc - GPUE: Split Operator based GPU solver for Nonlinear
00002 Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
00003 <loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
00004 All rights reserved.
00005 
00006 Redistribution and use in source and binary forms, with or without
00007 modification, are permitted provided that the following conditions are
00008 met:
00009 
00010 1. Redistributions of source code must retain the above copyright
00011 notice, this list of conditions and the following disclaimer.
```

```

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

double sepAvg(int2 *vArray, int2 centre, int length){
    double result=0.0;// = sqrt( pow(centre.x - v_array[0].x,2) + pow(centre.y - v_array[0].y,2));
    for (int i=0; i<length; ++i){
        result += sqrt( pow(centre.x - v_array[i].x,2) + pow(centre.y - v_array[i].y,2));
    }
    return result/length;
}
```

6.53 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:

Namespaces

- Tracker

    See the source file for info on functions.

Functions

- int Tracker::findOLMaxima (int ∗marker, double ∗V, double radius, int xDim, double ∗x)

    Finds the maxima of the optical lattice.

- int Tracker::findVortex (int ∗marker, double2 ∗wfc, double radius, int xDim, double ∗x, int timestep)

    Phase winding method to determine vortex positions.

- void Tracker::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 Tracker::phaseTest (int2 vLoc, double2 ∗wfc, int xDim)

    Tests the phase winding of the wavefunction, looking for vortices.

- double Tracker::sigVOL (struct Tracker::Vortex ∗vArr, int2 ∗opLatt, double ∗x, int numVort)

    Sigma of vortex lattice and optical lattice.

- double Tracker::vortAngle (struct Tracker::Vortex ∗vortCoords, struct Vortex central, int numVort)

    Determines the angle of the vortex lattice relative to the x-axis.

- void Tracker::vortArrange (struct Tracker::Vortex ∗vCoordsC, struct Vortex ∗vCoordsP, int length)

    Ensures the vortices are tracked and arranged in the right order based on minimum distance between previous and current positions.

- struct Vortex Tracker::vortCentre (struct Tracker::Vortex ∗cArray, int length, int xDim)

Determines the coords of the vortex closest to the central position.

- void Tracker::vortPos (int ∗marker, struct Tracker::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.

- double Tracker::vortSepAvg (struct Vortex ∗vArray, struct Tracker::Vortex centre, int length)

  Determines the vortex separation at the centre of the lattice.

Variables

- char Tracker::bufferT [1024]

6.54 tracker.cc

```
/*** tracker.cc - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

#include "../include/tracker.h"
#include "../include/fileIO.h"
#include "../include/minions.h"
#include "../include/constants.h"

namespace Tracker {
    char bufferT[1024];

    double vortSepAvg(struct Vortex *vArray, struct
Tracker::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;
        double min_tmp = 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){
            min_tmp = sqrt( pow(centre.coords.x - vArray[j].coords.x,2) + pow(centre.
coords.y - vArray[j].coords.y,2));
            if(min > min_tmp && min_tmp > 1e-7){
                min = min_tmp;
                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]-Minions::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] = Minions::psi2(wfc[(i-1)*xDim + (j-1)]);
                    gridValues[1] = Minions::psi2(wfc[(i-1)*xDim + j]);
                    gridValues[2] = Minions::psi2(wfc[(i-1)*xDim + (j+1)]);
                    gridValues[3] = Minions::psi2(wfc[(i)*xDim + (j-1)]);
                    gridValues[4] = Minions::psi2(wfc[(i)*xDim + j]);
                    gridValues[5] = Minions::psi2(wfc[(i)*xDim + (j+1)]);
                    gridValues[6] = Minions::psi2(wfc[(i+1)*xDim + (j-1)]);
                    gridValues[7] = Minions::psi2(wfc[(i+1)*xDim + j]);
                    gridValues[8] = Minions::psi2(wfc[(i+1)*xDim + (j+1)]);
                    if(fabs((gridValues[4]-Minions::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

    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] = Minions::complexScale(
Minions::complexDiv( wfc[i*xDim + j],         wfc[(i+1)*xDim + j] ),       (
Minions::complexMag( wfc[(i+1)*xDim + j])         /
Minions::complexMag( wfc[i*xDim + j] )));
                                    g[1] = Minions::complexScale(
Minions::complexDiv( wfc[(i+1)*xDim + j],     wfc[(i+1)*xDim + (j+1)] ),   (
Minions::complexMag( wfc[(i+1)*xDim + (j+1)])     /
Minions::complexMag( wfc[(i+1)*xDim + j] )));
                                    g[2] = Minions::complexScale(
Minions::complexDiv( wfc[(i+1)*xDim + (j+1)],    wfc[i*xDim + (j+1)] ),      (
Minions::complexMag( wfc[i*xDim + (j+1)])        /
Minions::complexMag( wfc[(i+1)*xDim + (j+1)] )));
                                    g[3] = Minions::complexScale(
Minions::complexDiv( wfc[i*xDim + (j+1)],    wfc[i*xDim + j] ),          (
Minions::complexMag( wfc[i*xDim + j])            /
Minions::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] = Minions::complexScale(
Minions::complexDiv(wfc[i*xDim + j],wfc[(i+1)*xDim + j]),            (
Minions::complexMag(wfc[(i+1)*xDim + j])  /
Minions::complexMag(wfc[i*xDim + j])));
        gridValues[1] = Minions::complexScale(
Minions::complexDiv(wfc[(i+1)*xDim + j],wfc[(i+1)*xDim + (j+1)]),    (
Minions::complexMag(wfc[(i+1)*xDim + (j+1)])/
Minions::complexMag(wfc[(i+1)*xDim + j])));
        gridValues[2] = Minions::complexScale(
Minions::complexDiv(wfc[(i+1)*xDim + (j+1)],wfc[i*xDim + (j+1)]),    (
Minions::complexMag(wfc[i*xDim + (j+1)])  /
Minions::complexMag(wfc[(i+1)*xDim + (j+1)])));
        gridValues[3] = Minions::complexScale(
Minions::complexDiv(wfc[i*xDim + (j+1)],wfc[i*xDim + j]),            (
Minions::complexMag(wfc[i*xDim + j])      /
Minions::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;
        }
        return result;
    }

    void vortPos(int *marker, struct Tracker::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;
                }
            }
        }
    }

    void vortArrange(struct Tracker::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;
                }
            }
            Minions::coordSwap(vCoordsC,index,i);
        }
    }

    struct Vortex vortCentre(struct Tracker::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 Tracker::Vortex *vortCoords, struct Vortex central, int numVort){
        int location;
        double sign=1.0;
        double minVal=1e300;//(pow(central.x - vortCoords[0].x,2) + pow(central.y - vortCoords[0].y,2));
        for (int i=0; i < numVort; ++i){
            if (minVal > (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) > 2e-6 && abs(central.coords.y - vortCoords[i].coords.y) > 2e-6){
                minVal = (pow(central.coords.x - vortCoords[i].coords.x,2) + pow(central.coords.y - vortCoords[i].coords.y,2));
                location = i;
            }
        }
        double ang=(fmod(atan2( (vortCoords[location].coords.y - central.coords.y), (vortCoords[location].coords.x - central.coords.x) ),PI/3));
        printf("Angle=%e\n",ang);
        return PI/3 - ang;

        //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) ) / ( minVal*(central.x - vortCoords[location].x) ) );
    }

    double sigVOL(struct Tracker::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;
    }

}
```

6.55 src/wavefunction.cu File Reference

Classes

- class BEC2D::Wavefunction

Namespaces

- BEC2D

Functions

- int2 getGridSize (int xDim, int yDim)
- double2 & getWfc ()
- double2 initWfc ()
- bool setGridSize (int xDim, int yDim)
- class BEC2D::Wavefunction BEC2D::Wavefunction ()
- Wavefunction ()
- Wavefunction (int xDim, int yDim, double xMax, double yMax)

Variables

- double2 dimMax
- int2 gridSize
- double2 ∗ wfc = new double2[xDim∗yDim]

6.55.1 Function Documentation

6.55.1.1 int2 Wavefunction::getGridSize (int *xDim,* int *yDim*)

6.55.1.2 double2& Wavefunction::getWfc ( )

6.55.1.3 double2 Wavefunction::initWfc ( )

6.55.1.4 bool Wavefunction::setGridSize (int *xDim,* int *yDim*)

6.55.1.5 Wavefunction::Wavefunction ( )

Definition at line 54 of file wavefunction.cu.

```
                                        {

    }
```

6.55.1.6 Wavefunction::Wavefunction (int *xDim*, int *yDim*, double *xMax*, double *yMax*)

6.55.2 Variable Documentation

6.55.2.1 double2 dimMax

Definition at line 315 of file wavefunction.cu.

6.55.2.2 int2 gridSize

Definition at line 314 of file wavefunction.cu.

6.55.2.3 double2∗ wfc = new double2[xDim∗yDim]

Definition at line 316 of file wavefunction.cu.

6.56 wavefunction.cu

```
/*** wavefunction.cu - GPUE: Split Operator based GPU solver for Nonlinear
Schrodinger Equation, Copyright (C) 2011-2015, Lee J. O'Riordan
<loriordan@gmail.com>, Tadhg Morgan, Neil Crowley.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the copyright holder nor the names of its
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/


/*
Unused. Future extension.
*/
namespace BEC2D{
    class Wavefunction{
        private:
            int2 gridSize;
            double2 dimMax;
            double2 *wfc = new double2[xDim*yDim];

        public:
            Wavefunction();
            Wavefunction(int xDim, int yDim, double xMax, double
yMax);
            bool setGridSize(int xDim, int yDim);
            int2 getGridSize(int xDim, int yDim);
            double2 initWfc();
            double2 &getWfc();
    }

    Wavefunction::Wavefunction(){

    }
    Wavefunction::Wavefunction(int xDim, int yDim, double
xMax, double yMax){

    }
    Wavefunction::setGridSize(int xDim, int yDim){

    }
}


```