High Performance Scientific Computing with Python

By: George Lees Jr.

Python is a high level scripting language that is being recognized for its versatility and its overall easy of use. One of the areas most influenced by Python is the field of Scientific Computing. Python provides a very fluid interface within IPython with libraries including Numpy and SciPy that have surpassed the former industry leader Matlab. The computational science tools Matlab, Maple, Mathematica are very useful tools that provide user friendly environments and a plethora of easy to use built-in functions. However, such high level functionality is inversely proportional to performance. Here I will show how Python can be used as a glue language so one can get the best of both worlds, the availability of high level built in functions with the performance of lower level languages like C or even better.

In order to showcase Pythons abilities I am going to use a problem that requires some computational power. I am going to show how to solve the wave partial differential equation over a 100 x 100 grid with many time steps. The numerical method used is a standard central differencing scheme for this equation:

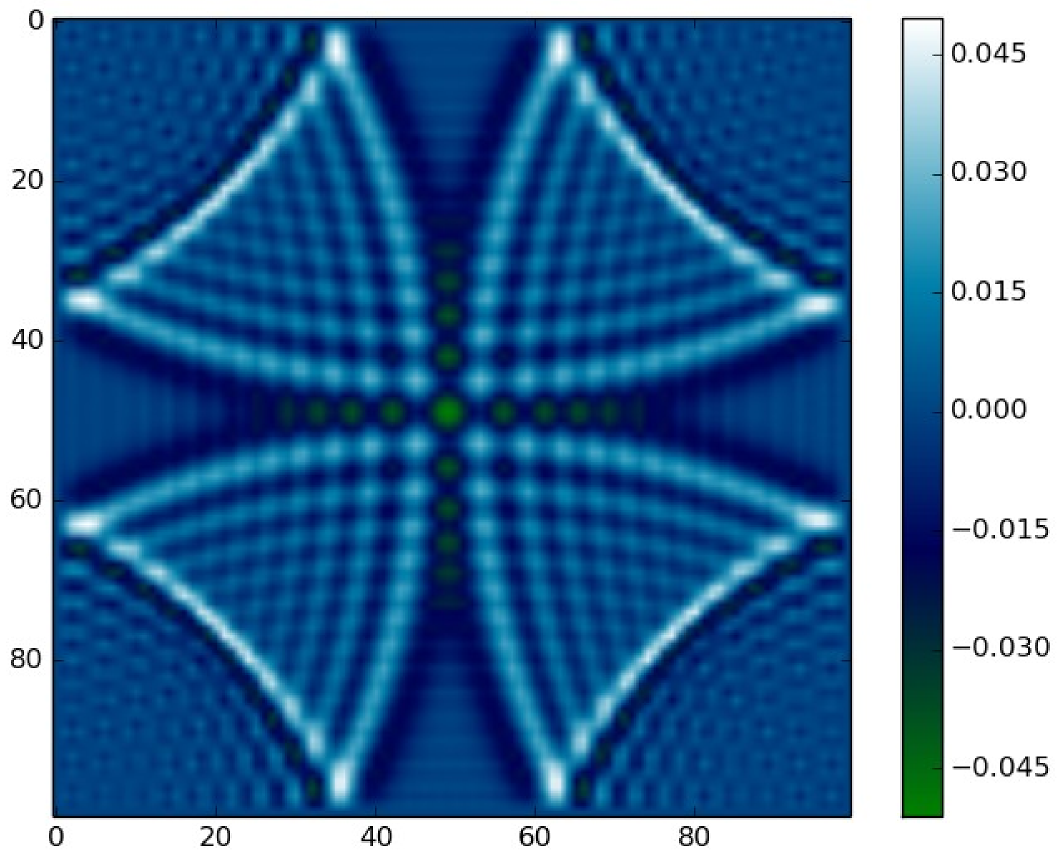
$$\frac{\partial^2}{\partial t^2} u(x,t) + \frac{\partial}{\partial t} \beta u(x,t) = \frac{\partial}{\partial x}\left([c(x,t)]^2 \frac{\partial}{\partial x} u(x,t)\right)$$

Here is the standard python code :

```
'''George Lees Jr.
2D Wave pde '''

from numpy import *
import numpy as np
import matplotlib.pyplot as plt
from pylab import *
np.set_printoptions(threshold=np.nan)

'''Declare Variables:
I am creating three 2D arrays due to mulitple levels of time stepping. u_prev is
for previous time step due to d/dt.
x and y are for indexing into the equations. '''

Lx=Ly=100
dx=dy = 1.0
x=y = np.array(xrange(Lx))
u_prev=np.ndarray(shape=(Lx,Ly), dtype=np.double)
u=np.ndarray(shape=(Lx,Ly), dtype=np.double)
u_next=np.ndarray(shape=(Lx,Ly), dtype=np.double)
c = 1 #-----------------------------------------------------constant velocity
dt = (1/float(c))*(1/sqrt(1/dx**2 + 1/dy**2))#-------------numerical stability is
exactly this number
t_old=0.0;t=0.0;t_end=100.0

'''Set Initial Conditions and Boundary Points
I(x) is initial shape of the wave '''

def I(x,y): return exp(-(x-Lx/2.0)**2/2.0 -(y-Ly/2.0)**2/2.0)
```

```
#---------------------------------------set up initial wave shape
for i in xrange(Lx):
        for j in xrange(Ly):
            u[i,j] = I(x[i],y[j])

#-----------------------------------------copy initial wave shape for printing later

u1=u.copy()

#-----------------------------------------set up previous time step array

for i in xrange(1,Lx-1):
        for j in xrange(1,Ly-1):
                u_prev[i,j] = u[i,j] + 0.5*((c*dt/dx)**2)*(u[i-1,j] - 2*u[i,j] +
u[i+1,j]) + \
                        0.5*((c*dt/dy)**2)*(u[i,j-1] - 2*u[i,j] + u[i,j+1])

#------------------------------------------set boundary conditions to 0

for j in xrange(Ly): u_prev[0,j] = 0
for i in xrange(Lx): u_prev[i,0] = 0
for j in xrange(Ly): u_prev[Lx-1,j] = 0
for i in xrange(Lx): u_prev[i,Ly-1] = 0

#------------------------------------------the wave steps through time
while t<t_end:
        t_old=t; t +=dt

        for i in xrange(1,Lx-1):
                for j in xrange(1,Ly-1):
                        u_next[i,j] = - u_prev[i,j] + 2*u[i,j] + \
                                ((c*dt/dx)**2)*(u[i-1,j] - 2*u[i,j] + u[i+1,j]) + \
                                ((c*dt/dx)**2)*(u[i,j-1] - 2*u[i,j] + u[i,j+1])

#------------------------------------------set boundary conditions to 0

        for j in xrange(Ly): u_next[0,j] = 0
        for i in xrange(Lx): u_next[i,0] = 0
        for j in xrange(Ly): u_next[Lx-1,j] = 0
        for i in xrange(Lx): u_next[i,Ly-1] = 0

#------------------------------------------set prev time step equal to current one
        u_prev = u.copy(); u = u_next.copy();

#plot the matrix at t=100
fig = plt.figure()
plt.imshow(u,cmap=plt.cm.ocean)
plt.colorbar()
plt.show()
```

The program runs for only 20 seconds due to the fact the I just numpy arrays which provide more optimization due to vectorization. And here is the plot:



So overall not bad ,but it is apparent that most of the compute time of the program is in the one while loop where the time stepping is being done. So, we can easily export the most expensive part of the program to a C function. The easiest way I found to do this is by using a program called Cython. Essentially Cython is its own language but it is very easy to pick up if you know C and Python already. To use a Cython function you just write the function in a separate file saved with a .pyx extension then compile it with the Cython compiler. Then you can go ahead and call this function from anywhere within the same folder after importing the folder like a library.

Here is the Cython function:

```
from numpy import *
cimport numpy as np

#The hardest part of doing this was making sure your data types match up

def cwave_prop( np.ndarray[double,ndim=2] u_prev, np.ndarray[double,ndim=2] u,
np.ndarray[double,ndim=2] u_next, int Lx , int Ly , double t_end):

        cdef double t = 0
        cdef double t_old = 0
        cdef int i,j
        cdef double c = 1
        cdef double dx = 1
        cdef double dy = 1
        cdef double dt = (1/(c))*(1/(sqrt(1/dx**2 + 1/dy**2)))

        while t<t_end:
                t_old=t; t +=dt

                #wave steps through time and space

                for i in xrange(1,Lx-1):
                        for j in xrange(1,Ly-1):
                                u_next[i,j] = - u_prev[i,j] + 2*u[i,j] + \
                                ((c*dt/dx)**2)*(u[i-1,j] - 2*u[i,j] + u[i+1,j]) + \
                                ((c*dt/dx)**2)*(u[i,j-1] - 2*u[i,j] + u[i,j+1])

                #set boundary conditions of grid to 0

                for j in xrange(Ly): u_next[0,j] = 0
                for i in xrange(Lx): u_next[i,0] = 0
                for j in xrange(Ly): u_next[Lx-1,j] = 0
                for i in xrange(Lx): u_next[i,Ly-1] = 0

                #set prev time step equal to current one
                for i in xrange(Lx):
                        for j in xrange(Ly):
                                u_prev[i,j] = u[i,j];
                                u[i,j] = u_next[i,j];
```

When I call this function from with my original wave.py program it runs in .48 seconds using the same values and I get the same plot ! I tried to accomplish the same thing using an interface language called SWIRL and after three days I switched to Cython and it worked magically. Cython was able to pass the 2D arrays to the C function without a problem. Also a quick note about the stability constant dt; if it is not that number exactly you get completely different results !!

Next, I wanted to learn how to do domain decomposition. So, I proceeded to write the mpi code for the 2D wave pde. What I thought would be a straight-forward program turned into a 708 line monstrosity. But I am pleased to announce; It works! I spent a very long time writing and debugging this code. I get the same graphs for u_prev and u after initialization and also the same plot for u_next after the main computation. Here it is highly commented:

```
'''  George Lees Jr.

     2D Wave pde mpi cartesian cords
This code only works for 8 processors. You have to have mpi4py installed to run it:
mpiexec -n 8 python mpi_wave2.py  '''

from __future__ import division
import matplotlib.pyplot as plt
from numpy import *
import numpy as np
from mpi4py import MPI
from time import time
if __name__ == "__main__":
        #-----------------------------------------------------------------
prints whole matrix
```

```
        np.set_printoptions(threshold=np.nan)
        np.set_printoptions(precision=2)

        def I(x,y): return exp(-(x-Lx/2.0)**2/2.0 -(y-Ly/2.0)**2/2.0)
#Initial Condition / shape of wave
        #----------------------------------------------------------------------setup
mpi
        ndims = 2
        comm = MPI.COMM_WORLD
        p = comm.Get_size()
        rank = comm.Get_rank()
        #-----------------------------------------------------------------------
        NUM_OF_PROCS = 8
        NUM_OF_PROC_DIMS = NUM_OF_PROCS/2
        #----------------------------------------------------------------------
declare variables
        Lx = Ly = (100)
#Length of matrix
        dx=dy = 1
        #----------------------------------------------------------------------
create cartesian topology
        dims = MPI.Compute_dims( p, ndims )                                # So
8 processors over a 2D grid
        ccomm = comm.Create_cart( dims )
        NORTH = 0
        SOUTH = 1
        EAST = 2
        WEST = 3
        #----------------------------------------------------------------------get
myranks local coordinates
        my_mpi_rows, my_mpi_cols = ccomm.Get_coords( ccomm.rank )            #
variables to help with message passing
        num_blocks_x = dims[1]
        num_blocks_y = dims[0]
        neigh = [0,0,0,0]
        neigh[NORTH], neigh[SOUTH] = ccomm.Shift(0, 1)
        neigh[WEST],  neigh[EAST]  = ccomm.Shift(1, 1)
        up = int(neigh[NORTH])
        down = int(neigh[SOUTH])
        right =int(neigh[EAST])
        left = int(neigh[WEST])
        num_rows_block = int(Lx/num_blocks_y)
        num_cols_block = int(Ly/num_blocks_x)
        #---------------------------------------------------------------------
starting x and y values for current procs
        my_x= my_mpi_cols * num_cols_block
        my_y= my_mpi_rows * num_rows_block

        print("My rank:%d mympirow:%d mympicol:%d upper neighbor:%d  #blocksx:%d
#blocksy%d my_x:%d my_y:%d num_rows_block:%d num_cols_block:%d" % ( rank,
my_mpi_rows, my_mpi_cols, up, dims[1], dims[0], my_x, my_y, num_rows_block,
num_cols_block ))
        comm.barrier()
        #------------------------_________________________-So basically the grid
looks like this-numbers corresponding to ranks
        #                       |           |           |-except all side are equal
length
        #                       |    0      |     1     |-each block is 50 indices
wide and 25 indices tall
```

```
        #                       |           |           |-hence 100 x 100
        #                       |___________|___________|
        #                       |           |           |
        #                       |     2     |     3     |
        #                       |           |           |
        #                       |___________|___________|
        #                       |           |           |
        #                       |     5     |     6     |
        #                       |           |           |
        #                       |___________|___________|
        #                       |           |           |
        #                       |      7    |     8     |
        #                       |           |           |
        #-----------------------|___________|___________|----------------------need
3 arrays u_prev is for previous time step due to d/dt
        u_prev = np.empty(( 100 , 100 ),np.float64)
        u_prev1 = np.empty(( num_rows_block , num_cols_block ),np.float64)#----and
i needed 8. 1 for each process and also a final one so
        u_prev2 = np.empty(( num_rows_block , num_cols_block ),np.float64)#----
after the computations i could send all of them to master
        u_prev3 = np.empty(( num_rows_block , num_cols_block ),np.float64)#----then
stack up to get the final matrix back
        u_prev4 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_prev5 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_prev6 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_prev7 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_prev8 = np.empty(( num_rows_block , num_cols_block ),np.float64)

        u1 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u2 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u3 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u4 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u5 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u6 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u7 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u8 = np.empty(( num_rows_block , num_cols_block ),np.float64)

        u1temp = np.empty(( num_rows_block+1 , num_cols_block+1 ),np.float64)
        u2temp = np.empty(( num_rows_block+1 , num_cols_block+1 ),np.float64)
        u3temp = np.empty(( num_rows_block+2 , num_cols_block+1 ),np.float64)
        u4temp = np.empty(( num_rows_block+2 , num_cols_block+1 ),np.float64)
        u5temp = np.empty(( num_rows_block+2 , num_cols_block+1 ),np.float64)
        u6temp = np.empty(( num_rows_block+2 , num_cols_block+1 ),np.float64)
        u7temp = np.empty(( num_rows_block+1 , num_cols_block+1 ),np.float64)
        u8temp = np.empty(( num_rows_block+1 , num_cols_block+1 ),np.float64)
        u9temp = np.empty(( num_rows_block+1 , num_cols_block+1 ),np.float64)
        uF1 = np.empty(( 100 , 100 ),np.float64)
        u = np.empty(( 100 , 100 ),np.float64)

        u_Final = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next1 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next2 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next3 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next4 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next5 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next6 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next7 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next8 = np.empty(( num_rows_block , num_cols_block ),np.float64)
        u_next= np.empty((100 , 100 ),np.float64)
        send_horiz = np.empty((1, num_cols_block), np.float64 ) #----ok i need all
of these because arrays bc i have to
        send_horiz1 = np.empty((1, num_cols_block), np.float64 )#----pack up and
sent the boundaries to each other
        send_horiz2 = np.empty((1, num_cols_block), np.float64 )
        send_horiz3 = np.empty((1, num_cols_block), np.float64 )
        send_horiz4 = np.empty((1, num_cols_block), np.float64 )
        send_horiz5 = np.empty((1, num_cols_block), np.float64 )
        send_horiz6 = np.empty((1, num_cols_block), np.float64 )
        send_horiz7 = np.empty((1, num_cols_block), np.float64 )
        send_horiz8 = np.empty((1, num_cols_block), np.float64 )
        send_horiz9 = np.empty((1, num_cols_block), np.float64 )
        send_horiz10 = np.empty((1, num_cols_block), np.float64 )
        send_horiz11 = np.empty((1, num_cols_block), np.float64 )
        send_horiz12 = np.empty((1, num_cols_block), np.float64 )

        send_vert = np.empty( (num_rows_block,1), np.float64 )
        send_vert1 = np.empty( (num_rows_block,1), np.float64 )
        send_vert2 = np.empty( (num_rows_block,1), np.float64 )
        send_vert3 = np.empty( (num_rows_block,1), np.float64 )
        send_vert4 = np.empty( (num_rows_block,1), np.float64 )
        send_vert5 = np.empty( (num_rows_block,1), np.float64 )
        send_vert6 = np.empty( (num_rows_block,1), np.float64 )
        send_vert7 = np.empty( (num_rows_block,1), np.float64 )
        send_vert8 = np.empty( (num_rows_block,1), np.float64 )
        #---------------------------------------------------------------------
        x = y = np.array( xrange( Lx ))
#setting up a linspace
        c = 1
#constant velocity
        dt = (1/float(c))*(1/sqrt(1/dx**2 + 1/dy**2))                         #dt
stability equation
        print dt
        t_old=0 ; t=0 ; t_end=100
        #---------------------------------------------------------------------set up
initial wave shape
        if rank == 0:#-------------------------------------------------------ok so
here im setting the initial condition wave
                for i in xrange(num_rows_block):#-----------------------------each
process has its own indexing into x and y
                        for j in xrange(num_cols_block):
                                u1[i,j] = I(x[my_y+i],y[my_x+j])

        if rank == 1:
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u2[i,j] = I(x[my_y+i],y[my_x+j])

                comm.Send(u2, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u2, source=1, tag=44)

        if rank == 2 :
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u3[i,j] = I(x[my_y+i],y[my_x+j])

                comm.Send(u3, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u3, source=2, tag=44)

        if rank == 3 :
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u4[i,j] = I(x[my_y+i],y[my_x+j])


                comm.Send(u4, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u4, source=3, tag=44)

        if rank == 4 :
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u5[i,j] = I(x[my_y+i],y[my_x+j])

                comm.Send(u5, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u5, source=4, tag=44)

        if rank == 5 :
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u6[i,j] = I(x[my_y+i],y[my_x+j])

                comm.Send(u6, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u6, source=5, tag=44)

        if rank == 6:
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u7[i,j] = I(x[my_y+i],y[my_x+j])

                comm.Send(u7, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u7, source=6, tag=44)

        if rank == 7:
                for i in xrange(num_rows_block):
                        for j in xrange(num_cols_block):
                                u8[i,j] = I(x[my_y+i],y[my_x+j])

                comm.Send(u8, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u8, source=7, tag=44)

#Ok now in order to calculate u_prev i have to send all neighbors the corresponding
buffer arrays and receive them. This way i can
#index without going out of bounds

        req = [None, None, None, None] #-------------------------------------
required to hold error messages for mpi

        if rank == 0:
                for i in xrange(num_cols_block):
                        send_horiz1[0,i] = u1[num_rows_block-1,i]
                for i in xrange(num_rows_block):
                        send_vert1[i,] = u1[i,num_cols_block-1]

                req[SOUTH] = ccomm.Isend(send_horiz1, neigh[SOUTH])
                req[EAST] = ccomm.Isend(send_vert1, neigh[EAST])

        if rank == 1:
                req[WEST] = ccomm.Irecv(send_vert1, neigh[WEST])

        if rank == 2:
                req[NORTH] = ccomm.Irecv(send_horiz1, neigh[NORTH])

        if rank == 1:
                for i in xrange(num_cols_block):
                        send_horiz2[0,i] = u2[num_rows_block-1,i]
                for i in xrange(num_rows_block):
                        send_vert2[i,] = u2[i,0]
                req[SOUTH] = ccomm.Isend(send_horiz2, neigh[SOUTH])
                req[WEST] = ccomm.Isend(send_vert2, neigh[WEST])

        if rank == 0:
                req[EAST] = ccomm.Irecv(send_vert2, neigh[EAST])

        if rank == 2:
                req[NORTH] = ccomm.Irecv(send_horiz2, neigh[NORTH])

        if rank == 2:
                for i in xrange(num_cols_block):
                        send_horiz3[0,i] = u3[0,i]
                        send_horiz4[0,i] = u3[num_rows_block-1,i]
                for i in xrange(num_rows_block):
                        send_vert3[i,] = u3[i,num_cols_block-1]

                req[NORTH] = ccomm.Isend(send_horiz3, neigh[NORTH])
                req[SOUTH] = ccomm.Isend(send_horiz4, neigh[SOUTH])
                req[EAST] = ccomm.Isend(send_vert3, neigh[EAST])
        if rank == 0:
                req[SOUTH] = ccomm.Irecv(send_horiz3, neigh[SOUTH])
        if rank == 3:
                req[WEST] = ccomm.Irecv(send_vert3, neigh[WEST])
        if rank == 4:
                req[NORTH] = ccomm.Irecv(send_horiz4, neigh[NORTH])

        if rank == 3:
                for i in xrange(num_cols_block):
                        send_horiz5[0,i] = u4[0,i]
                        send_horiz6[0,i] = u4[num_rows_block-1,i]
                for i in xrange(num_rows_block):
                        send_vert4[i,] = u4[i,0]

                req[NORTH] = ccomm.Isend(send_horiz5, neigh[NORTH])
                req[SOUTH] = ccomm.Isend(send_horiz6, neigh[SOUTH])
                req[WEST] = ccomm.Isend(send_vert4, neigh[WEST])
        if rank == 1:
                req[SOUTH] = ccomm.Irecv(send_horiz5, neigh[SOUTH])
        if rank == 5:
                req[NORTH] = ccomm.Irecv(send_horiz6, neigh[NORTH])
        if rank == 2:
                req[EAST] = ccomm.Irecv(send_vert4, neigh[EAST])
```

```
if rank == 4:
        for i in xrange(num_cols_block):
                send_horiz7[0,i] = u5[0,i]
                send_horiz8[0,i] = u5[num_rows_block-1,i]
        for i in xrange(num_rows_block):
                send_vert5[i,] = u5[i,num_cols_block-1]

        req[NORTH] = ccomm.Isend(send_horiz7, neigh[NORTH])
        req[SOUTH] = ccomm.Isend(send_horiz8, neigh[SOUTH])
        req[EAST] = ccomm.Isend(send_vert5, neigh[EAST])
if rank == 2:
        req[SOUTH] = ccomm.Irecv(send_horiz7, neigh[SOUTH])
if rank == 6:
        req[NORTH] = ccomm.Irecv(send_horiz8, neigh[NORTH])
if rank == 5:
        req[WEST] = ccomm.Irecv(send_vert5, neigh[WEST])

if rank == 5:
        for i in xrange(num_cols_block):
                send_horiz9[0,i] = u6[0,i]
                send_horiz10[0,i] = u6[num_rows_block-1,i]
        for i in xrange(num_rows_block):
                send_vert6[i,] = u6[i,0]

        req[NORTH] = ccomm.Isend(send_horiz9, neigh[NORTH])
        req[SOUTH] = ccomm.Isend(send_horiz10, neigh[SOUTH])
        req[WEST] = ccomm.Isend(send_vert6, neigh[WEST])
if rank == 3:
        req[SOUTH] = ccomm.Irecv(send_horiz9, neigh[SOUTH])
if rank == 7:
        req[NORTH] = ccomm.Irecv(send_horiz10, neigh[NORTH])
if rank == 4:
        req[EAST] = ccomm.Irecv(send_vert6, neigh[EAST])

if rank == 6:
        for i in xrange(num_cols_block):
                send_horiz11[0,i] = u7[0,i]
        for i in xrange(num_rows_block):
                send_vert7[i,] = u7[i,num_cols_block-1]

        req[NORTH] = ccomm.Isend(send_horiz11, neigh[NORTH])
        req[EAST] = ccomm.Isend(send_vert7, neigh[EAST])
if rank == 4:
        req[SOUTH] = ccomm.Irecv(send_horiz11, neigh[SOUTH])
if rank == 7:
        req[WEST] = ccomm.Irecv(send_vert7, neigh[WEST])

if rank == 7:
        for i in xrange(num_cols_block):
                send_horiz12[0,i] = u8[0,i]
        for i in xrange(num_rows_block):
                send_vert8[i,] = u8[i,0]

        req[NORTH] = ccomm.Isend(send_horiz12, neigh[NORTH])
        req[WEST] = ccomm.Isend(send_vert8, neigh[WEST])
if rank == 5:
        req[SOUTH] = ccomm.Irecv(send_horiz12, neigh[SOUTH])
if rank == 6:
```

```
                req[EAST] = ccomm.Irecv(send_vert8, neigh[EAST])

        comm.barrier()
#---------------------------------------------------------------------------Now
stack buffers onto the 2D arrays for index padding
        if rank ==0:
                u1 = np.concatenate((u1,send_horiz3),axis=0)
                a=np.array([[0]])
                send_vert2 = np.append(a,send_vert2,axis=0)
                u1 = np.append(u1,send_vert2,axis=1)

        if rank ==1:
                u2 = np.concatenate((u2,send_horiz5),axis=0)
                a=np.array([[0]])
                send_vert1 = np.append(a,send_vert1,axis=0)
                u2 = np.append(send_vert1,u2,axis=1)

                comm.Send(u2, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u2temp, source=1, tag=44)

        if rank ==2:
                u3 = np.concatenate((send_horiz1,u3),axis=0)
                u3 = np.concatenate((u3,send_horiz7),axis=0)
                a=np.array([[0]])
                send_vert4 = np.append(send_vert4,a,axis=0)
                send_vert4 = np.append(a,send_vert4,axis=0)
                u3 = np.append(u3,send_vert4,axis=1)

                comm.Send(u3, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u3temp, source=2, tag=44)

        if rank ==3:
                u4 = np.concatenate((send_horiz2,u4),axis=0)
                u4 = np.concatenate((u4,send_horiz9),axis=0)
                a=np.array([[0]])
                send_vert3 = np.append(send_vert3,a,axis=0)
                send_vert3 = np.append(a,send_vert3,axis=0)
                u4 = np.append(send_vert3,u4,axis=1)

                comm.Send(u4, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u4temp, source=3, tag=44)

        if rank ==4:
                u5 = np.concatenate((send_horiz4,u5),axis=0)
                u5 = np.concatenate((u5,send_horiz11),axis=0)
                a=np.array([[0]])
                send_vert6 = np.append(send_vert6,a,axis=0)
                send_vert6 = np.append(a,send_vert6,axis=0)
                u5 = np.append(u5,send_vert6,axis=1)

                comm.Send(u5, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u5temp, source=4, tag=44)

        if rank ==5:
                u6 = np.concatenate((send_horiz6,u6),axis=0)
                u6 = np.concatenate((u6,send_horiz12),axis=0)
                a=np.array([[0]])
                send_vert5 = np.append(send_vert5,a,axis=0)
                send_vert5 = np.append(a,send_vert5,axis=0)
                u6 = np.append(send_vert5,u6,axis=1)

                comm.Send(u6, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u6temp, source=5, tag=44)


        if rank ==6:
                u7 = np.concatenate((send_horiz8,u7),axis=0)
                a=np.array([[0]])
                send_vert8 = np.append(send_vert8,a,axis=0)
                u7 = np.append(u7,send_vert8,axis=1)

                comm.Send(u7, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u7temp, source=6, tag=44)


        if rank ==7:
                u8 = np.concatenate((send_horiz10,u8),axis=0)
                a=np.array([[0]])
                send_vert7 = np.append(send_vert7,a,axis=0)
                u8 = np.append(send_vert7,u8,axis=1)

                comm.Send(u8, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u8temp, source=7, tag=44)

#-------Now rank zero has them all and can stack them and show it for comparison
with the serial code

        if rank ==0:
                u1= u1[0:25,0:50]
                u2= u2temp[0:25,1:51]
                u3= u3temp[1:26,0:50]
                u4= u4temp[1:26,1:51]
                u5= u5temp[1:26,0:50]
                u6= u6temp[1:26,1:51]
                u7= u7temp[1:26,0:50]
                u8= u8temp[1:26,1:51]
                u_init1 = np.append(u1,u2,axis=1)
                u_init2 = np.append(u3,u4,axis=1)
                u_init3 = np.append(u5,u6,axis=1)
                u_init4 = np.append(u7,u8,axis=1)
                u_init5 = np.concatenate((u_init3,u_init4),axis=0)
                u_init6 = np.concatenate((u_init2,u_init5),axis=0)
                u = np.concatenate((u_init1,u_init6),axis=0)
                uF1= u.copy()
                #print u.shape
                #fig = plt.figure()
                #plt.imshow(u,cmap=plt.cm.ocean)
                #plt.colorbar()
                #plt.show()
                comm.Send(uF1, dest=1, tag=44)
        if rank == 1:
                comm.Recv(uF1, source=0, tag=44)


        comm.Bcast([u, MPI.DOUBLE], root = 0)#---------------------------------
Broadcast final result
        comm.barrier()
#-------------------------------------------------------------------------------
Then do the same thing for u_prev
        if rank == 0:
                for i in xrange(1,u_prev1.shape[0]):
                        for j in xrange(1,u_prev1.shape[1]):
                                u_prev1[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for i in xrange(u_prev1.shape[1]): u_prev1[0,i] = 0
                for j in xrange(u_prev1.shape[0]): u_prev1[j,0] = 0

        if rank == 1:
                for i in xrange(1,u_prev2.shape[0]):
                        for j in xrange(0,u_prev2.shape[1]-1):
                                u_prev2[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for i in xrange(u_prev2.shape[1]): u_prev2[0,i] = 0
                for j in xrange(u_prev2.shape[0]): u_prev2[j,u_prev2.shape[1]-1] =
0

                comm.Send(u_prev2, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u_prev2, source=1, tag=44)

        if rank == 2 :
                for i in xrange(0,u_prev3.shape[0]):
                        for j in xrange(1,u_prev3.shape[1]):
                                u_prev3[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for j in xrange(u_prev3.shape[0]): u_prev3[j,0] = 0


                comm.Send(u_prev3, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u_prev3, source=2, tag=44)

        if rank == 3 :
                for i in xrange(0,u_prev4.shape[0]):
                        for j in xrange(0,u_prev4.shape[1]-1):
                                u_prev4[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for j in xrange(u_prev4.shape[0]): u_prev4[j,u_prev4.shape[1]-1] =
0

                comm.Send(u_prev4, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u_prev4, source=3, tag=44)

        if rank == 4 :
                for i in xrange(0,u_prev5.shape[0]):
                        for j in xrange(1,u_prev5.shape[1]):
                                u_prev5[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for j in xrange(u_prev5.shape[0]): u_prev5[j,0] = 0

                comm.Send(u_prev5, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u_prev5, source=4, tag=44)

        if rank == 5 :
                for i in xrange(0,u_prev6.shape[0]):
                        for j in xrange(0,u_prev6.shape[1]-1):
                                u_prev6[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for j in xrange(u_prev6.shape[0]): u_prev6[j,u_prev6.shape[1]-1] =
0


                comm.Send(u_prev6, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u_prev6, source=5, tag=44)

        if rank == 6:
                for i in xrange(0,u_prev7.shape[0]-1):
                        for j in xrange(1,u_prev7.shape[1]):
                                u_prev7[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for i in xrange(u_prev7.shape[1]): u_prev7[u_prev7.shape[0]-1,i] =
0
                for j in xrange(u_prev7.shape[0]): u_prev7[j,0] = 0


                comm.Send(u_prev7, dest=0, tag=44)
        if rank == 0:
                comm.Recv(u_prev7, source=6, tag=44)

        if rank == 7:
                for i in xrange(0,u_prev8.shape[0]-1):
                        for j in xrange(0,u_prev8.shape[1]-1):
                                u_prev8[i,j] = u[my_y+i,my_x+j] + \
                                0.5*((c*dt/dx)**2)*(u[my_y+i-1,my_x+j] -
2*u[my_y+i,my_x+j] + u[my_y+i+1,my_x+j]) + \
                                0.5*((c*dt/dy)**2)*(u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                for i in xrange(u_prev8.shape[1]): u_prev8[u_prev8.shape[0]-1,i] =
0
                for j in xrange(u_prev8.shape[0]): u_prev8[j,u_prev8.shape[1]-1] =
0

                comm.Send(u_prev8, dest=0, tag=34)
        if rank == 0:
                comm.Recv(u_prev8, source=7, tag=34)

                u_prev_F1 = np.append(u_prev1,u_prev2,axis=1)
                u_prev_F2 = np.append(u_prev3,u_prev4,axis=1)
                u_prev_F3 = np.append(u_prev5,u_prev6,axis=1)
                u_prev_F4 = np.append(u_prev7,u_prev8,axis=1)
                u_prev_F5 = np.concatenate((u_prev_F3,u_prev_F4),axis=0)
                u_prev_F6 = np.concatenate((u_prev_F2,u_prev_F5),axis=0)
                u_prev = np.concatenate((u_prev_F1,u_prev_F6),axis=0)
                #fig = plt.figure()
                #plt.imshow(u_prev,cmap=plt.cm.ocean)
                #plt.colorbar()
                #plt.show()
        comm.Bcast([u_prev, MPI.DOUBLE], root = 0)
        comm.barrier()

#Main Calculation u_next! I switched to just indexing from within each sub block
into the whole matrix
#because i couldn't get the other way to work. Kept the boundaries 0.

        while t<t_end:
                t_old=t; t +=dt

                if rank == 0:
                        for i in xrange(1,num_rows_block):
                                for j in xrange(1,num_cols_block):
                                        u_next1[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for i in xrange(u_next1.shape[1]): u_next1[0,i] = 0
                        for j in xrange(u_next1.shape[0]): u_next1[j,0] = 0

                if rank == 1:
                        for i in xrange(1,num_rows_block):
                                for j in xrange(0,num_cols_block-1):
                                        u_next2[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for i in xrange(u_next2.shape[1]): u_next2[0,i] = 0
                        for j in xrange(u_next2.shape[0]):
```

```
u_next2[j,u_next2.shape[1]-1] = 0

                        comm.Send(u_next2, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next2, source=1, tag=44)

                if rank == 2 :
                        for i in xrange(0,num_rows_block):
                                for j in xrange(1,num_cols_block):
                                        u_next3[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for j in xrange(u_next3.shape[0]): u_next3[j,0] = 0
                        comm.Send(u_next3, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next3, source=2, tag=44)

                if rank == 3 :
                        for i in xrange(0,num_rows_block):
                                for j in xrange(0,num_cols_block-1):
                                        u_next4[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for j in xrange(u_next4.shape[0]):
u_next4[j,u_next4.shape[1]-1] = 0
                        comm.Send(u_next4, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next4, source=3, tag=44)

                if rank == 4 :
                        for i in xrange(0,num_rows_block):
                                for j in xrange(1,num_cols_block):
                                        u_next5[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for j in xrange(u_next5.shape[0]): u_next5[j,0] = 0
                        comm.Send(u_next5, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next5, source=4, tag=44)

                if rank == 5 :
                        for i in xrange(0,num_rows_block):
                                for j in xrange(0,num_cols_block-1):
                                        u_next6[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for j in xrange(u_next6.shape[0]):
u_next6[j,u_next6.shape[1]-1] = 0
                        comm.Send(u_next6, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next6, source=5, tag=44)

                if rank == 6:
                        for i in xrange(0,num_rows_block-1):
                                for j in xrange(1,num_cols_block):
                                        u_next7[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for i in xrange(u_next7.shape[1]):
u_next7[u_next7.shape[0]-1,i] = 0
                        for j in xrange(u_next7.shape[0]): u_next7[j,0] = 0
                        comm.Send(u_next7, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next7, source=6, tag=44)

                if rank == 7:
                        for i in xrange(0,num_rows_block-1):
                                for j in xrange(0,num_cols_block-1):
                                        u_next8[i,j] = - u_prev[my_y+i,my_x+j] +
2*u[my_y+i,my_x+j] + \
                                                ((c*dt/dx)**2)*(u[my_y+i-1,my_x+j]
- 2*u[my_y+i,my_x+j] + \
                                                u[my_y+i+1,my_x+j]) +
((c*dt/dx)**2)* \
                                                (u[my_y+i,my_x+j-1] -
2*u[my_y+i,my_x+j] + u[my_y+i,my_x+j+1])
                        for i in xrange(u_next8.shape[1]):
u_next8[u_next8.shape[0]-1,i] = 0
                        for j in xrange(u_next8.shape[0]):
u_next8[j,u_next8.shape[1]-1] = 0


                        comm.Send(u_next8, dest=0, tag=44)
                if rank == 0:
                        comm.Recv(u_next8, source=7, tag=44)
                        u_Final1 = np.append(u_next1,u_next2,axis=1)
                        u_Final2 = np.append(u_next3,u_next4,axis=1)
                        u_Final3 = np.append(u_next5,u_next6,axis=1)
                        u_Final4 = np.append(u_next7,u_next8,axis=1)
                        u_Final5 = np.concatenate((u_Final3,u_Final4),axis=0)
                        u_Final6 = np.concatenate((u_Final2,u_Final5),axis=0)
                        u_next = np.concatenate((u_Final1,u_Final6),axis=0)
                        u_prev = u.copy();
                        u = u_next.copy();
                comm.Bcast([u, MPI.DOUBLE], root = 0)
                comm.Bcast([u_prev, MPI.DOUBLE], root = 0)


#-----------------------------------------------------------------------------------
Build Final matrix result

        #if rank ==0:
                #fig = plt.figure()
                #plt.imshow(u_next,cmap=plt.cm.ocean)
                #plt.colorbar()
                #plt.show()
```

Here I commented out the graphs to get a time of 6.11 seconds ! I was real scared there for a while because the code is 708 lines so I thought it would take forever to run. But, I saw a very nice speed up. It is a shame that the mpi_wave program is not dynamic enough to scale up the project. However, I do believe that the mpi version would eventually beat the C extension as we went to bigger spaces. Unfortunately, in this case I had to do a lot of work to see a speed up. But, the mpi4py style is exactly like mpi for C and is very easy to work with. Overall, I do see a future for python as far a program for performance ! I think once one gets comfortable with these tools rapid prototyping would outweigh the time it would take to do things in the lower languages for some projects. Also, all computers in the future will have upwards of two cores. This means that computer programmers are going to have to learn how to write programs that can take advantage of these cores, and I think that python has the versatility to make this paradigm shift happen.