Natural Convection of Air in a Square Cavity Using Message Passing Interface

Priyesh Kakka (ME17S069)
High Performance Computing (AM5080)
Instructor – Dr.Sarith P Sathian

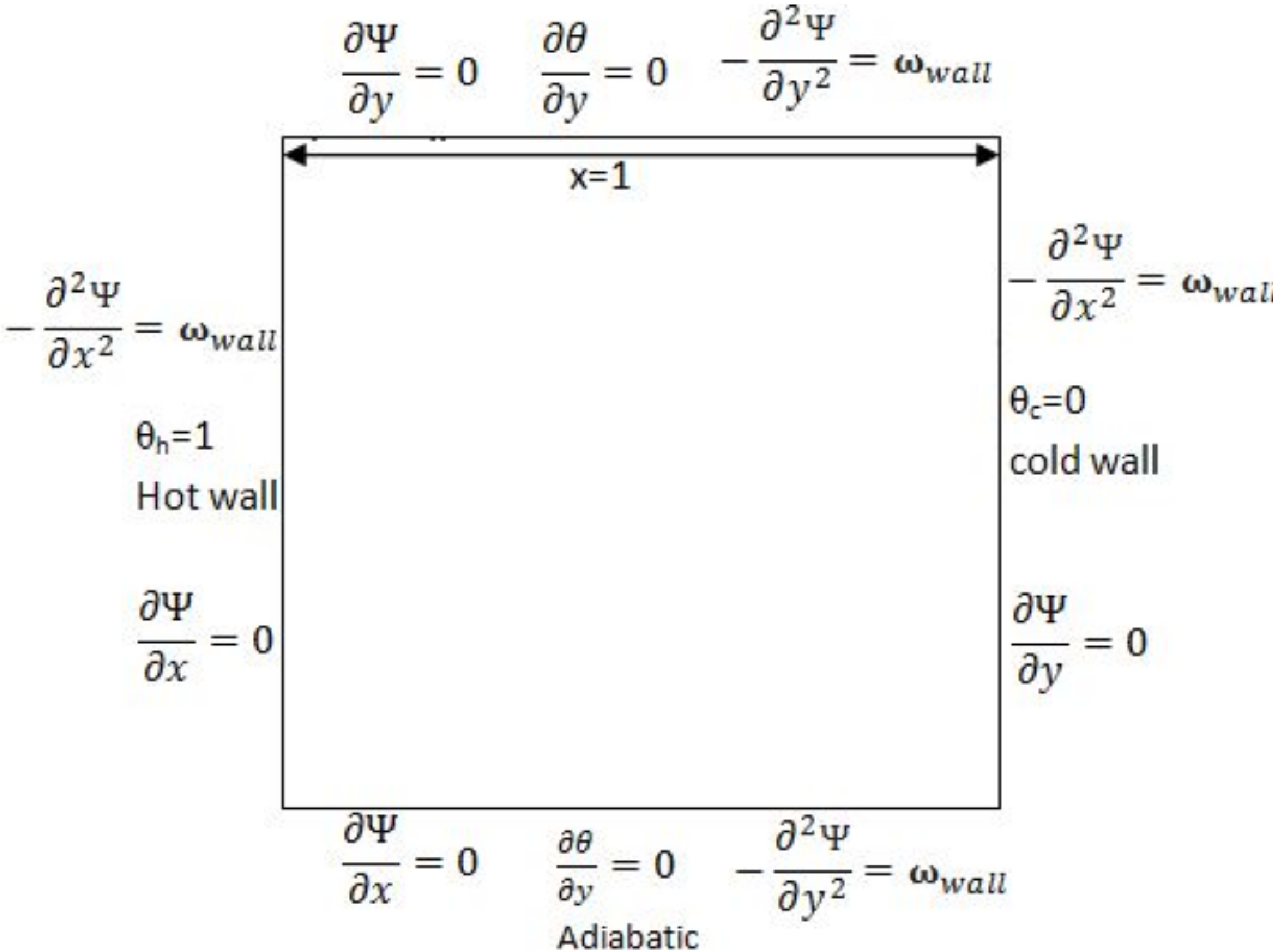
Stream-Vorticity Formulations

$$\frac{\partial \psi}{\partial y^2} + \frac{\partial \psi}{\partial x^2} = -\omega \qquad (1)$$

$$\frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = pr\left(\frac{\partial^2 \omega}{\partial y^2} + \frac{\partial^2 \omega}{\partial x^2}\right) - Rapr\frac{\partial \theta}{\partial y} \qquad (2)$$

$$\frac{\partial \psi}{\partial y}\frac{\partial \theta}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \theta}{\partial y} = \left(\frac{\partial^2 \theta}{\partial y^2} + \frac{\partial^2 \theta}{\partial x^2}\right) \qquad (3)$$

Boundary Conditions



Matrix initialization and distribution

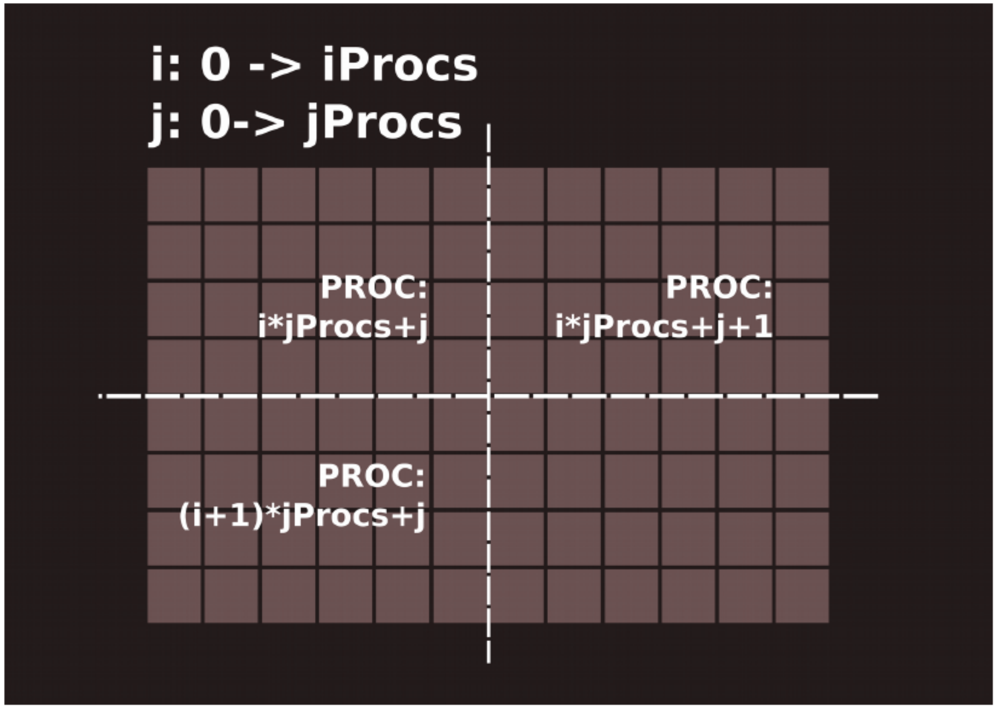
```
while(k1!=5):
                                                        #flags
 if (rank==0):                                       #   #writing seprate module for rank 0 for
robustness and ease of codeing as there wont be lag in time as all prc run seprately
   if (ite==1):

    print "No of cells in X is ",nx,"No of cells in Y is ",ny
    print("\n")
    print "No of Processors used ",nProcs
    print("\n")
    print "iProcs ",iProcs,"jProcs",jProcs
    print "\n"

    for i in range (0,iProcs):      #i is for nys
     for j in range (0,jProcs):
                                                                                  #loop is for
sending the remaining decomposed matrix into different cores
      data4=T[D[i]:D[i+1],E[j]:E[j+1]]
      data5=vort[D[i]:D[i+1],E[j]:E[j+1]]
      data6=phi[D[i]:D[i+1],E[j]:E[j+1]]
      data4 = np.pad(data4, pad_width=1, mode='constant', constant_values=0)
#defining ghost cells at each decomposed domain
      data5 = np.pad(data5, pad_width=1, mode='constant', constant_values=0)
      data6 = np.pad(data6, pad_width=1, mode='constant', constant_values=0)

      comm.send(data4,dest=i*jProcs+j,tag=15)  #sending data to differnet processors and
mapped
      comm.send(data5,dest=i*jProcs+j,tag=16)
      comm.send(data6,dest=i*jProcs+j,tag=17)


 if(ite==1):      #deleteing and giving boundaries to domains for the first iteration
only
    T1=comm.recv(source=0,tag=15);                    #reciving data distributed by
processor zero
    sf=comm.recv(source=0,tag=16);
    Phi=comm.recv(source=0,tag=17);
```

Assigns ghost cells to each divided domain

Sending and receiving 3 different matrix with different tags

Find Grid Location (from all Ranks)

- Rank = i*jProcs+j
- i = int(Rank/jProcs)
- j = Rank % jProcs



Code Explanation #1

```
from mpi4py import MPI
import numpy as np
comm=MPI.COMM_WORLD
rank=comm.Get_rank()
nProcs=comm.Get_size()
import matplotlib.pyplot as plt

ny=50                             #cells in  coulmns
nx=40                             #cells in rows
Y=10;                             #length of yaxis domain
h=0.025
Ra=10000                       #Raleigh number
Pr=0.71                        #Prandtl number
r=0.8  #relaxation factor
vort = np.zeros((ny, nx))         #defining Matrix
T = np.zeros((ny, nx))
phi = np.zeros((ny, nx))
C=[]
T1=[]
sf=[]

OR=float(nx)/ny;   #Aspect ratio of mesh
diff=1;
diff2=23;

#mapping nprocs to domain

if (nProcs==1):
  def fact(nProcs,OR):
    return 1,1
else:
 def fact(nProcs,OR):

        diff_old = 23
        for i in range(1, (nProcs/2)+1):
            if nProcs % i == 0:
                aspectRatio = i**2/float(nProcs)
                diff = abs( OR - aspectRatio)

                if diff < diff_old:
                   diff_old = diff
                   #print "i",i
        if i*int(nProcs/i)==nProcs:
             return i, int(nProcs/i)
        else:
             return nProcs,1                                # if cannot factor dividing nys in nprocs

iProcs, jProcs = fact(nProcs, float(nx/ny))
D=np.linspace(0,ny,(iProcs+1),dtype=int)
E=np.linspace(0,nx,(jProcs+1),dtype=int)
#print("D",D,"E",E)

Bi=int(rank/jProcs)   #finding i according to slide,position of mesh
Bj=int(rank%jProcs)
```

Divides the domain to allocate cores

Equation Discretisation and Residual calculation

```
    if(Bi==(iProcs-1)):
     T1 = np.delete(T1, -1,axis=0)
     sf = np.delete(sf, -1,axis=0)
     Phi = np.delete(Phi, -1,axis=0)
     T1[-1]=T1[-2]
     sf[-1]=-2.0*Phi[-2]/(h*h);                #bottom
```

This searches the boundary and gives B.C as well as deletes the ghost cells at boundary

```
    #applying Jacobi formula
 r=len(T1)
 c=len(T1[0])

 v=np.zeros((r,c))
 w=np.zeros((r,c))
 Rc=np.zeros((r,c))                         #initiating dummy variables for calculating
residue

 w=Phi
 for j in range (1,c-1,1):
      for i in range (1,r-1,1):                      # As numpy is row major

        v[i,j]=(((-0.25*((Phi[i,j+1]-Phi[i,j-1])*(sf[i+1,j]-sf[i-1,j]))+0.25*(((Phi[i
+1,j])-(Phi[i-1,j]))*(sf[i,j+1]-sf[i,j-1]))))/(h*h*Pr))+(((sf[i+1,j]+sf[i-1,j]+sf[i,j+1]
+sf[i,j-1]))/(h*h))-(Ra*(T1[i,j+1]-T1[i,j-1])/(2*h)))*(0.25*(h*h));
 Res=abs(np.subtract(sf[1:r-1,1:c-1],v[1:r-1,1:c-1]))               #Stream Function
equation
 Ressq=np.square(Res)
 Sum=np.sum(Ressq)
 #print Sum,"rank",rank
 sf[1:r-1,1:c-1]=v[1:r-1,1:c-1];
```

Stream Function Equation

Residual (L2 norm)

Discrete Equations and Boundary conditions

```
        #applying Jacobi formula

         w[i,j]=0.25*(Phi[i+1,j]+Phi[i-1,j]+Phi[i,j+1]+Phi[i,j-1]+h*h*sf[i,j]);
#vorticity equation

 Phi[1:r-1,1:c-1]=w[1:r-1,1:c-1];

 sf[0]=-2.0*Phi[1]/(h*h);
 sf[:,-1]=-2.0*Phi[:,-2]/(h*h);
 sf[:,0]=-2.0*Phi[:,1]/(h*h);
 sf[-1]=-2.0*Phi[-2]/(h*h);
 Rc=T1

 for j in range (1,c-1,1):
      for i in range (1,r-1,1):

        #applying Jacobi formula
        Rc[i,j]=(((-0.25*((Phi[i,j+1]-Phi[i,j-1])*(T1[i+1,j]-T1[i-1,j]))+0.25*(((Phi[i
+1,j])-(Phi[i-1,j]))*(T1[i,j+1]-T1[i,j-1])))/(h*h))+((T1[i+1,j]+T1[i-1,j]+T1[i,j+1]+T1
[i,j-1])/(h*h)))*(0.25*(h*h));                                          #Energy equation

 T1[1:r-1,1:c-1]=Rc[1:r-1,1:c-1];

 T1[0]=T1[1];
 T1[:,-1]=0
 T1[:,0]=1
 T1[-1]=T1[-2]
```

Vorticity and temperature Equation

Boundary Conditions

Communication

```
#communnication

if (Bj<(jProcs-1)):

    comm.send(T1[1:-1,-2],dest=(Bi*jProcs+Bj)+1,tag=12)
    T1r=comm.recv(source=(Bi*jProcs+Bj)+1,tag=12)

    T1[1:-1,-1]=T1r

    comm.send(sf[1:-1,-2],dest=(Bi*jProcs+Bj)+1,tag=13)
    sfr=comm.recv(source=(Bi*jProcs+Bj)+1,tag=13)
    sf[1:-1,-1]=sfr
    comm.send(Phi[1:-1,-2],dest=(Bi*jProcs+Bj)+1,tag=14)
    Phir=comm.recv(source=(Bi*jProcs+Bj)+1,tag=14)
    Phi[1:-1,-1]=Phir

if(Bj>0):

    comm.send(T1[1:-1,1],dest=(Bi*jProcs+Bj)-1,tag=12)
    T1r=comm.recv(source=(Bi*jProcs+Bj)-1,tag=12)
    T1[1:-1,0]=T1r
    comm.send(sf[1:-1,1],dest=(Bi*jProcs+Bj)-1,tag=13)
    sfr=comm.recv(source=(Bi*jProcs+Bj)-1,tag=13)
    sf[1:-1,0]=sfr
    comm.send(Phi[1:-1,1],dest=(Bi*jProcs+Bj)-1,tag=14)
    Phir=comm.recv(source=(Bi*jProcs+Bj)-1,tag=14)
    Phi[1:-1,0]=Phir
```

Sending and receiving values from ghost nodes with different tags

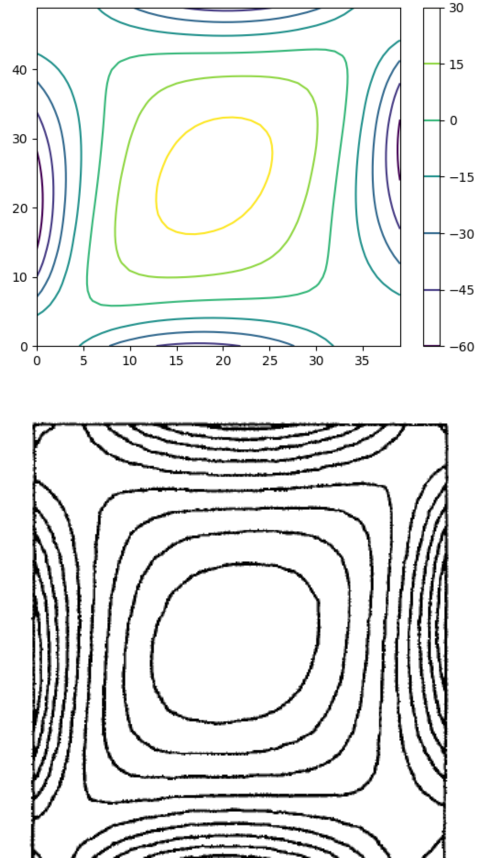
Cutting ghost cells and gathering

```
if (F==1):                                        #Flag if last iteration
    if (Bi!=0):
     T1 = np.delete(T1, 0,axis=0)
     sf = np.delete(sf, 0,axis=0)
     Phi = np.delete(Phi, 0,axis=0)

    if(Bj!=0):
     T1 = np.delete(T1, 0,axis=1)
     sf = np.delete(sf, 0,axis=1)
     Phi = np.delete(Phi, 0,axis=1)

    if(Bi!=(iProcs-1)):
     T1 = np.delete(T1, -1,axis=0)
     sf = np.delete(sf, -1,axis=0)
     Phi = np.delete(Phi, -1,axis=0)


    if(Bj!=(jProcs-1)):
     T1 = np.delete(T1, -1,axis=1)
     sf = np.delete(sf, -1,axis=1)
     Phi = np.delete(Phi, -1,axis=1)

    k1=5                                                #Flag to stop while loop

ite=ite+1
res=comm.allgather(Sum)
RESF=np.sum(res)
#print(ite,RESF)
if (RESF<0.05 and ite>200):                     #checking Residue
   F=1;
                                              #flag to start final iteration
L=comm.gather(Phi,root=0)                           #gathering T1 from all proc

if rank == 0:

       C = np.empty([ny, nx])

       t = 0
       for m in range(0,iProcs):
           for n in range(0,jProcs):
               C[D[m]:D[m+1], E[n]:E[n+1]] = L[t]
               t += 1

       print("\n Matrix C is \n")
       #print C
       print "\n","\n","Final Residue is",RESF
```

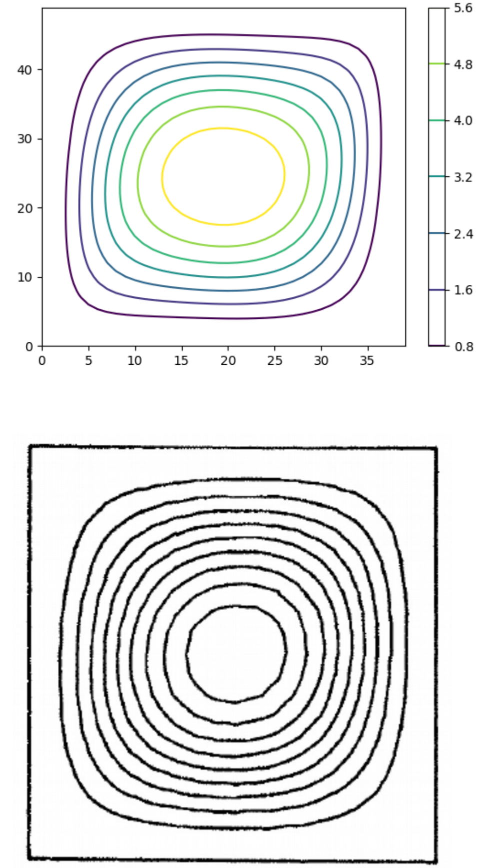
Cutting all the ghost cells to maintain matrix shape and gathering

Checking Residue and assigning a Flags

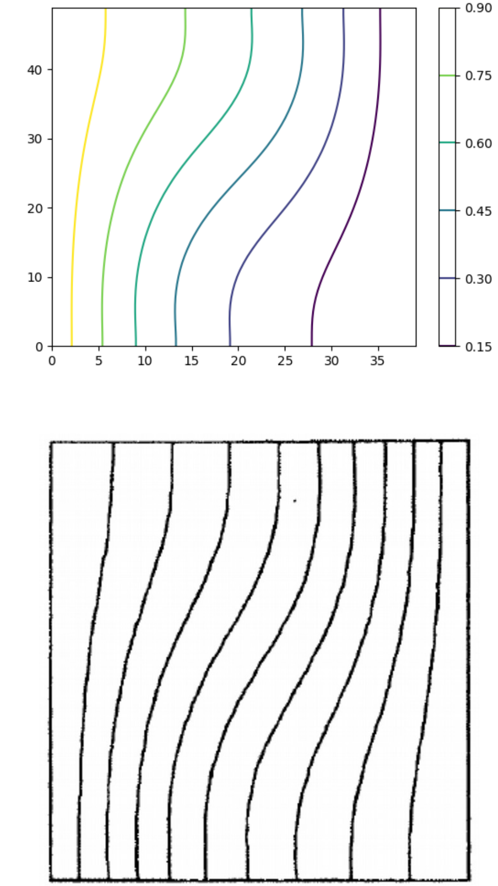
Output Profiles at Rayleigh number 1000
Code vs De Vahls (bottom)



Vorticity

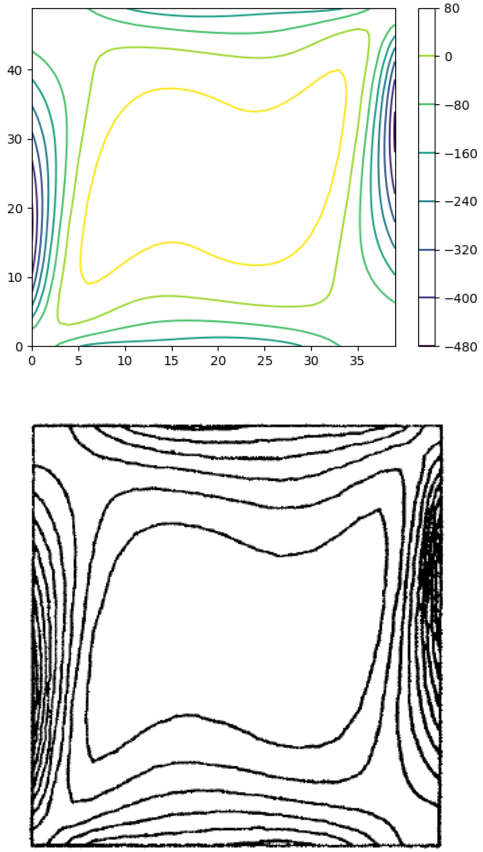


Stream Function

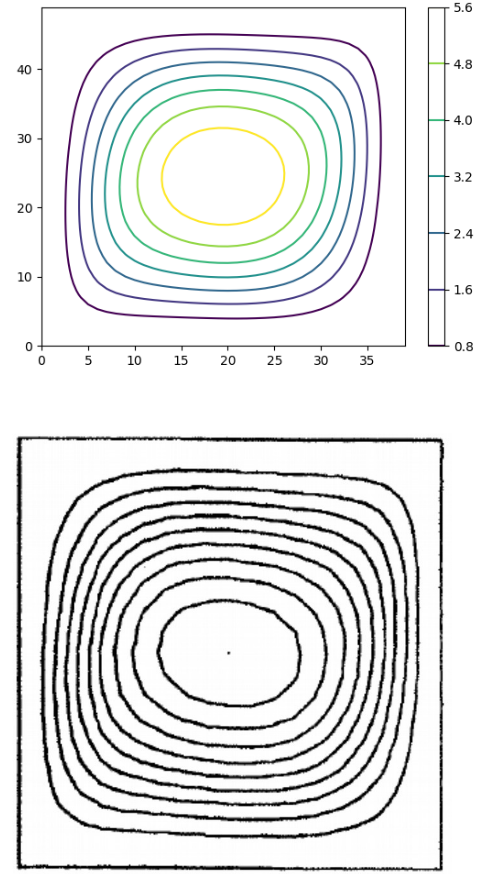


Temperature

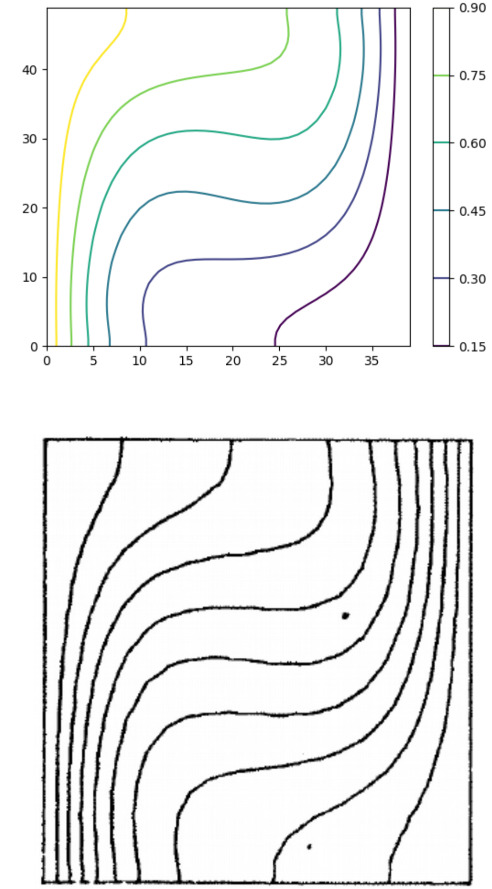
Output Profiles at Rayleigh number 10000 Code vs De Vahl's (bottom)



Vorticity

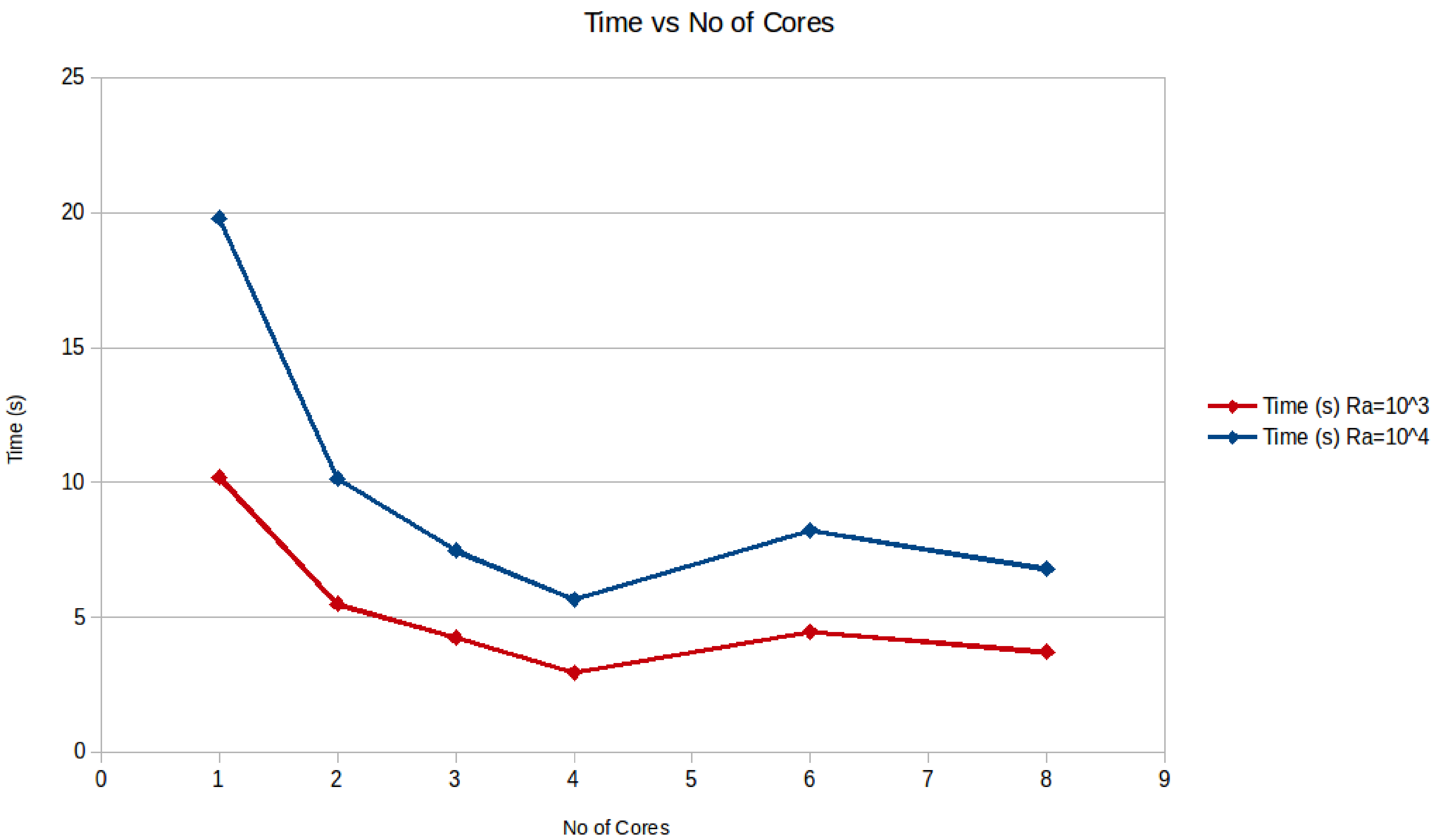


Stream Function



Temperature

Speed Up Due to MPI



Thank You