《计算流体力学基础》第六次作业

姓名：杨阳
学号：150*******
专业：流体力学

题目：

在单位正方形内，求解不可压缩流动。其中只有上边界是水平运动的，其他边界都是固定的。当上边界以均匀速度 1 运动时，角点处存在奇性，使得此问题和网格是有关的。此时角点处涡量的大小依赖于角点处的网格尺度 h，如图1所示，角点涡量为 $-1/h$。为了避免这种速度场在角点附近的奇性，可以对上边界

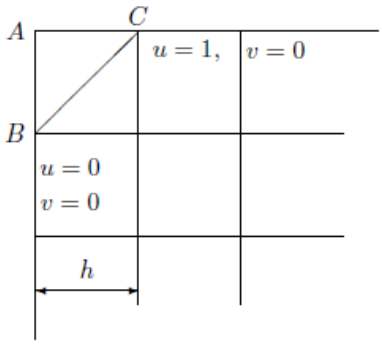


图 1: 驱动方腔内流动角点处涡量对网格尺度的依赖关系示意图，点 A 是方腔的左上角点，AB 和 AC 的长度都是网格尺度 h，在一个网格内，AB 上的速度为 $u=v=0$，AC 上的速度为 $u=1, v=0$，于是，如果在三角形 ABC 内认为速度是线性分布的，为满足连续性条件，A 点的速度应该取为上边界速度 1，AB 上的速度分布是从 1 降为 0，于是可以得到这个三角形内的速度分布为 $u=\frac{y-(1-h)}{h}$，$v=0$，涡量为 $\omega=\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}=-1/h$；如果 A 点的速度认为是 0，容易验证，不满足连续性条件。

给出一个速度分布

$$u(x)=16x^2(1-x)^2, \tag{1}$$

此分布在角点处函数值和导数值均为零，具有比较好的连续性。

因为所有的边界都是固壁边界，方腔流动中不需要人为引入出口处的边界条件，计算中的随意性比较少，做为经典算例，通常用来验证格式的计算精度。通常可以比较的量有中间线上的速度剖面，主涡涡心的位置和流函数值，角附近的二次涡的位置，三次涡的位置等。

求解过程：

1. 涡量和流函数法控制方程

（1）涡量输运方程

守恒形式的涡量输运方程：

$$\frac{\partial \omega}{\partial t}+\frac{\partial(u\omega)}{\partial x}+\frac{\partial(v\omega)}{\partial y}=\nu\nabla^2\omega$$

（2）流函数泊松方程

$$\nabla^2\psi=-\omega$$

（3）速度

$$u=\frac{\partial \psi}{\partial y},\quad v=-\frac{\partial \psi}{\partial x}$$

根据流函数的定义，速度可以表示成上面的形式。

（4）压力泊松方程

$$\nabla^2 p=2\left(\frac{\partial u}{\partial x}\frac{\partial v}{\partial y}-\frac{\partial u}{\partial y}\frac{\partial v}{\partial x}\right)$$

上面的控制方程可以通过对描述不可压缩粘性流体流动的纳维斯托克斯方程取旋度和散度得到。上面的方程组中，涡量的边界条件较为复杂，流函数的边界条件是边界为流线所以为常数，压力泊松方程的边界条件也较为复杂，但在计算过程中不需要使用压力，计算最后一步才会计算压力。

对于二维问题拉普拉斯算子为：

$$\nabla^2=\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}$$

2. 离散格式

（1） 涡量输运方程

$$\frac{\dot{\omega}_{i,j}^{n+1}-\omega_{i,j}^{n}}{\Delta t}+\frac{(u\omega)_{i+1,j}^{n}-(u\omega)_{i-1,j}^{n}}{2\Delta x}+\frac{(v\omega)_{i,j+1}^{n}-(v\omega)_{i,j-1}^{n}}{2\Delta y}$$

$$=\nu\left(\frac{\omega_{i+1,j}^{n}-2\omega_{i,j}^{n}+\omega_{i-1,j}^{n}}{\Delta x^2}+\frac{\omega_{i,j+1}^{n}-2\omega_{i,j}^{n}+\omega_{i,j-1}^{n}}{\Delta y^2}\right)$$

$$\frac{\hat{\omega}_{i,j}^{n+1}-\omega_{i,j}^{n}}{\Delta t}+\frac{\left(u\dot{\omega}^{n+1}\right)_{i+1,j}-\left(u\dot{\omega}^{n+1}\right)_{i-1,j}}{2\Delta x}+\frac{\left(v\dot{\omega}^{n+1}\right)_{i,j+1}-\left(v\dot{\omega}^{n+1}\right)_{i,j-1}}{2\Delta y}$$
$$=\nu\left(\frac{\dot{\omega}_{i+1,j}^{n+1}-2\dot{\omega}_{i,j}^{n+1}+\dot{\omega}_{i-1,j}^{n+1}}{\Delta x^{2}}+\frac{\dot{\omega}_{i,j+1}^{n+1}-2\dot{\omega}_{i,j}^{n+1}+\dot{\omega}_{i,j-1}{}^{n+1}}{\Delta y^{2}}\right)$$

涡量输运方程的计算在空间上采用中心差分格式，在时间上采用预测并校正的时间积分格式，这样在时空上都具有二阶的精度。

（2）流函数泊松方程

$$\psi_{i,j}^{n+1}=(1-\sigma)\psi_{i,j}^{n}+\frac{\sigma}{2(A+B)}\left[A\left(\psi_{i+1,j}^{n}+\psi_{i-1,j}\%^{n+1}\right)+B\left(\psi_{i,j+1}^{n}+\psi_{i,j-1}^{n+1}\right)-\omega_{i,j}^{n+1}\right]$$

其中

$$A=\frac{1}{\Delta x^{2}},\quad B=\frac{1}{\Delta y^{2}}$$

σ 为松弛因子，这计算的过程中取 1.5。
上面的公式为流函数泊松方程计算的 SOR 超松弛迭代公式。

（3）速度计算公式

$$u_{i,j}^{n+1}=\frac{\psi_{i,j+1}^{n+1}-\psi_{i,j-1}^{n+1}}{2\Delta y},\ v_{i,j}^{n+1}=-\frac{\psi_{i+1,j}^{n+1}-\psi_{i-1,j}^{n+1}}{2\Delta x}$$

在确定了涡量演化方程的离散格式和流函数泊松方程的离散格式以及速度的离散格式后就进行时间步的推进。

（4）压力泊松方程
压力泊松方程的离散格式与流函数泊松方程的离散格式类似：

$$p_{i,j}^{n+1}=(1-\sigma)\,p_{i,j}^{n}+\frac{\sigma}{2(A+B)}\left[A\left(p_{i+1,j}^{n}+p_{i-1,j}^{n+1}\right)+B\left(p_{i,j+1}^{n}+p_{i,j-1}^{n+1}\right)-\left(S_{p}\right)_{i,j}{}^{n+1}\right]$$

$$\left(S_{p}\right)_{i,j}^{n+1}=2\left(\frac{u_{i+1,j}^{n+1}-u_{i-1,j}^{n+1}}{2\Delta x}\frac{u_{i,j+1}^{n+1}-u_{i,j+1}^{n+1}}{2\Delta y}-\frac{u_{i,j+1}^{n+1}-u_{i,j+1}^{n+1}}{2\Delta y}\frac{u_{i+1,j}^{n+1}-u_{i-1,j}^{n+1}}{2\Delta x}\right)$$

3. 边界处理

（1）涡量边界条件

采用 Thom 壁面涡量公式

$$\omega\big|_w = -\frac{2}{\Delta n^2}\left[\psi\big|_{w+1} - \psi\big|_w + u_\tau \Delta n\right]$$

其中的顶部切向速度由题目中给出的速度分布确定，其余边界均为零。

（2） 流函数边界条件

因为边界为流线所以

$$\psi\big|_w = 0$$

在处理流函数和涡量的时候都应该谨慎处理角点。

（3）压力泊松方程边界条件

$$\left.\frac{\partial p}{\partial x}\right|_w = \mu \left.\frac{\partial^2 u}{\partial y^2}\right|_w = -\mu \left.\frac{\partial \omega}{\partial y}\right|_w$$

$$\frac{p_{i+1,1} - p_{i-1,1}}{2\Delta x} = -\mu \frac{-3\omega_{i,1} + 4\omega_{i,2} - \omega_{i,3}}{2\Delta y}$$

$$\frac{p_{i+1,1} - p_{i,1}}{\Delta x} = -\mu \frac{-3\omega_{i+1,1} + 4\omega_{i+1,2} - \omega_{i+1,3}}{2\Delta y}$$

壁面压强由壁面出动量守恒确定，但是在计算压强的时候必须至少给定一个点的压强值（参考压强）

4. 算法流程

（1）初始化速度场并计算与之关联的涡量场以及流函数。

（2）计算涡量边界条件并求解涡量输运方程。

（3）设置流函数边界条件并求解流函数泊松方程。

（4）根据流函数计算节点速度值。

（5）检查流场是否收敛并输出中间结果，如果收敛则进行下一步，否侧返回到第（2）步并继续时间推进。

（6）对收敛的流场设置压力泊松方程边界条件并求解压力泊松方程。

（7）输出计算日志文件

考虑到程序中除了泊松方程的求解过程外时间推进等步骤具有并行的可能性，因此对程序进行并行化以加快执行速度。

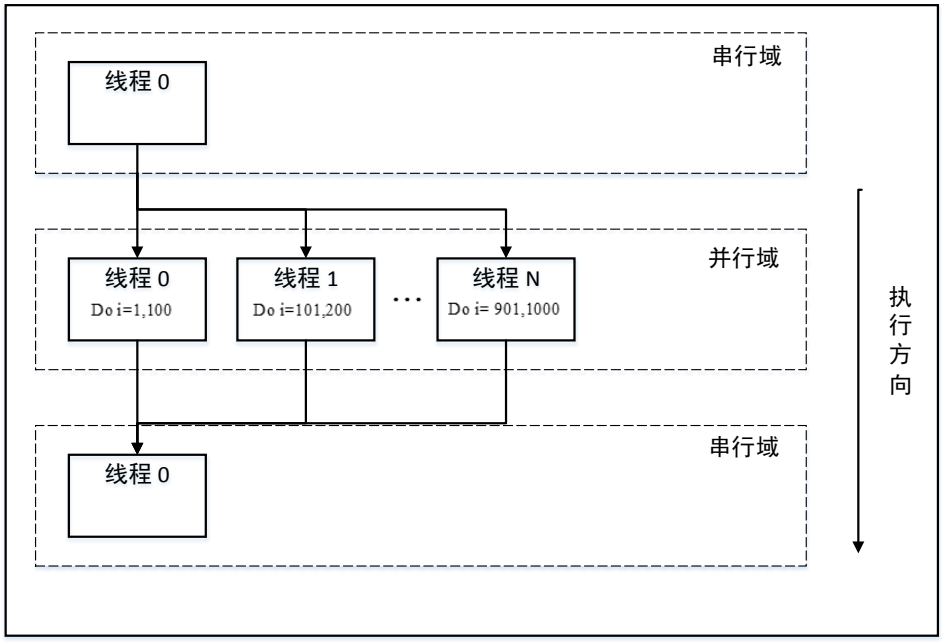
并行任务拆分过程如下所示

```
!$OMP PARALLEL DO
do i=1,1000
```

循环体

end do

!$OMP END PARALLEL DO

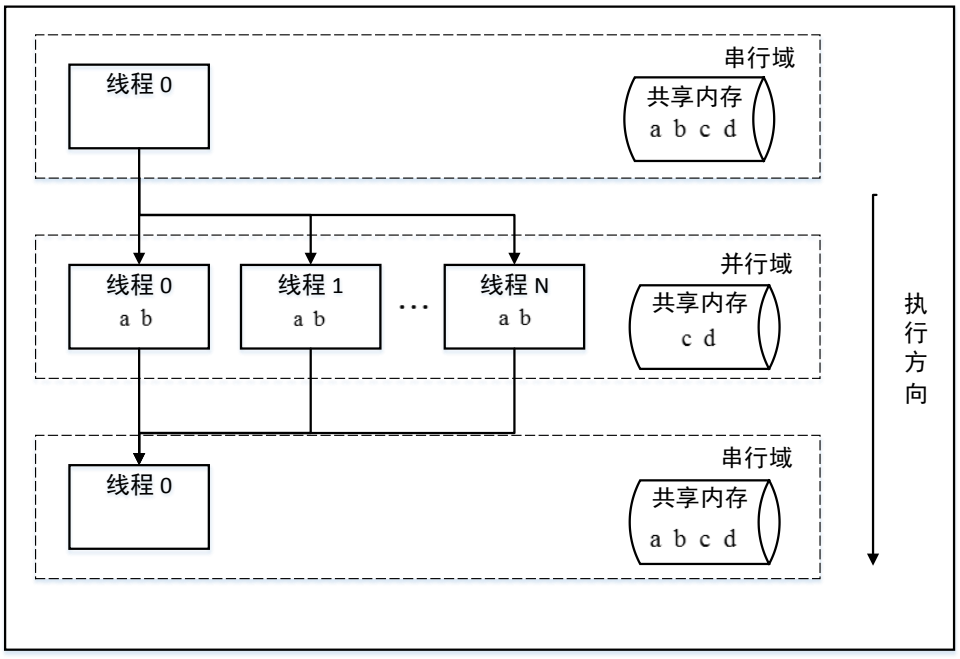


并行任务拆分过程中的数据环境配置如下所示

!$OMP PARALLEL PRIVATE(a,b) SHARED(c,d)

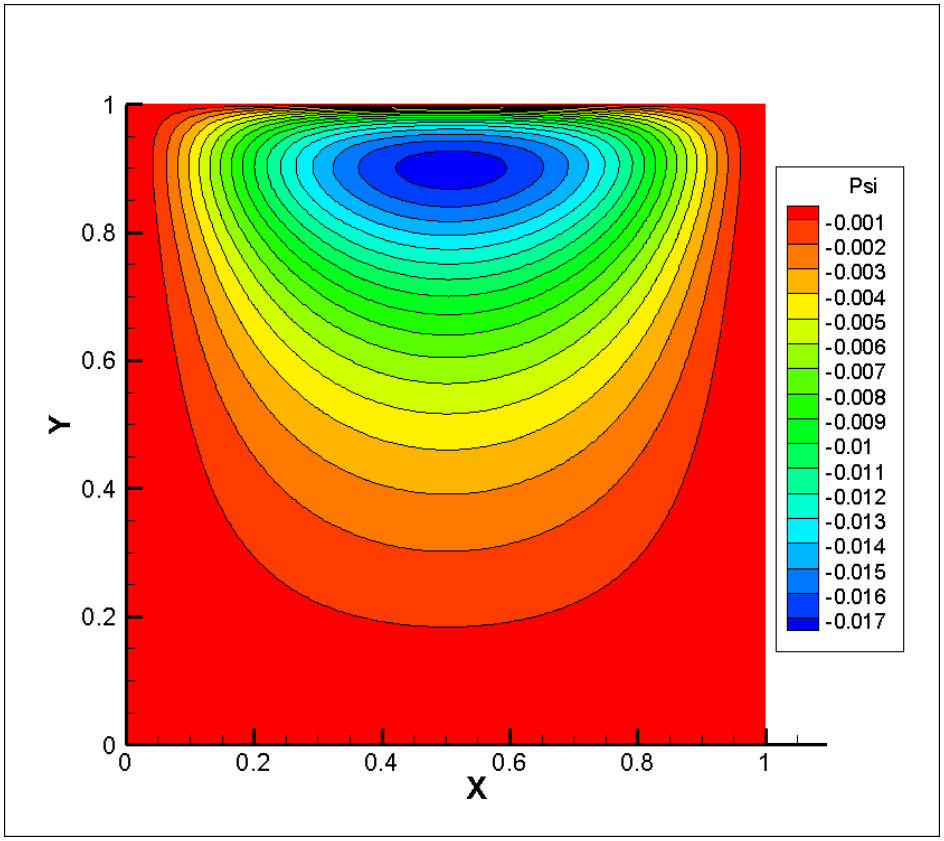
并行域

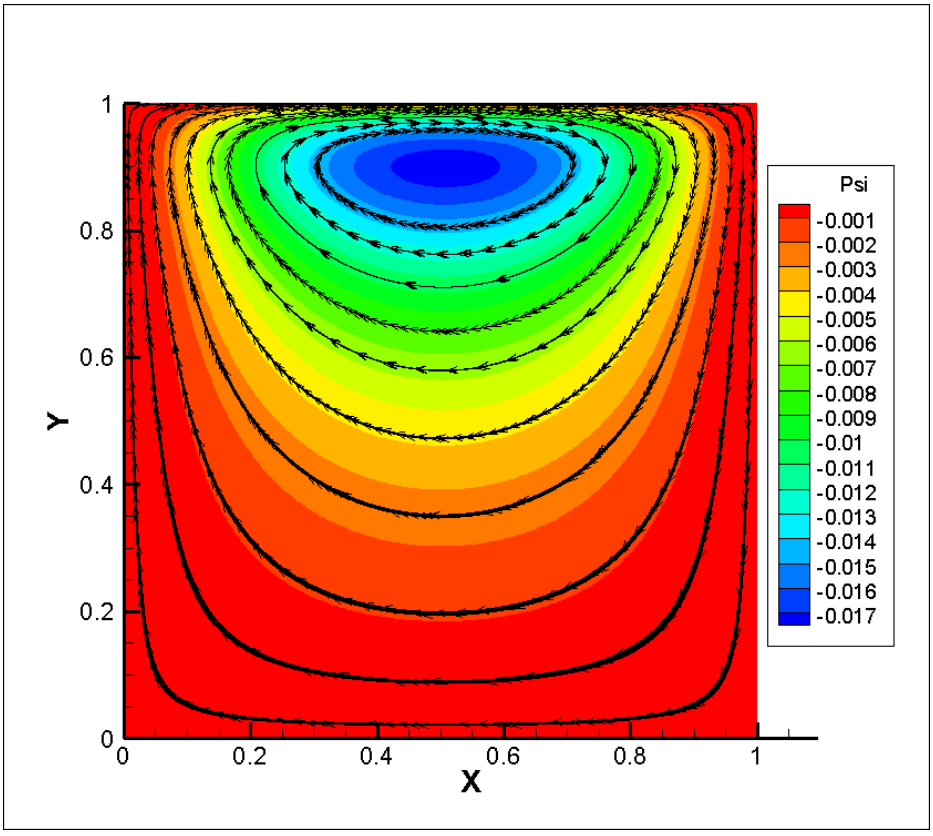
!$OMP END PARALLEL



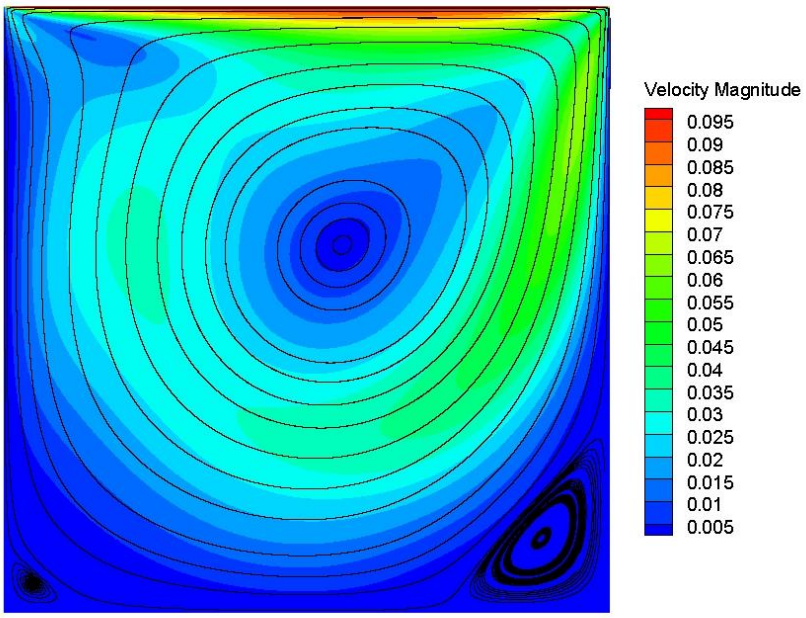
5. 计算结果

下面为一个雷诺数为 200 的无奇异性方腔流计算结果。





可以看到没有奇异性的方腔流对称性较好，下面是一个雷诺数 400 的有奇异性方腔流。



有奇异性的方腔流主漩涡的位置不在正中央。

程序附录

1. 涡量流函数法方腔流计算程序

LidDrivenFlow.f90

```
!-----------------------------------------------------------------------------------------------
---------------------------------------
MODULE ParametersBlock
implicit none
! Module name: ParametersBlock
! Modul purpose: Specify and compute basic parameters

! Parallel parameter
integer, parameter:: CPUS = 6

! Control parameters
integer, parameter:: Output_Interval = 1000, Report_Interval = 100
real*8, parameter:: eps = 5.d-4

! Discrete parameters
! Mesh parameters:
integer, parameter:: Nx = 256, Ny = 256          ! Mesh size
integer, parameter:: t_max = 1000000              ! Max number of time step

real*8, parameter:: delta_x = 1.d0                ! mesh step length in x
direction
real*8, parameter:: delta_y = delta_x             ! mesh step length in y
direction


! Physics parameters
! Flow parameters
real*8, parameter:: rho_0 = 1.d0                  ! Reference Density
real*8, parameter:: p_0 = 0.d0                    ! reference pressure
real*8, parameter:: Re = 200                      ! Renoyld number
(Viscosity)

! Computing parameters
! Fluid Flow
real*8, parameter:: Lx = Nx*delta_x               ! Characteristic length
real*8, parameter:: U_max = 1.d0                  ! Characteristic Velocity
real*8, parameter:: nu = U_max*Lx / Re            ! Viscosity of fluid
```

```
! Stable condition
real*8, parameter:: delta_t = 0.05d0 * 0.5d0 * 1/(1/(delta_x)**2 +
1/(delta_x)**2)    ! time step length, due to stable margin

END MODULE ParametersBlock
!-----------------------------------------------------------------------------------------------
---------------------------------------




!-----------------------------------------------------------------------------------------------
------------
!-----------------------------------------------------------------------------------------------
------------
!--Program purpose: This program solve lid-driven cavity flow by using
vorticity and stream function method
!----------Aurthor: Yang Yang       :-)
!-------------Date: 20/10/2015

!               Lid: u = u(x), v=0.
!               |---------------|
!               |                       |
!               |                       |
! No-slip |                       | No-slip
!       Wall |                       | Wall
!               |                       |
!               |                       |
!               |---------------|
!                    No-slip Wall

! main program
PROGRAM main
use ParametersBlock
implicit none

!%% global variables
real*8 u(1:Nx+1,1:Ny+1), v(1:Nx+1,1:Ny+1)
real*8 vor_new(1:Nx+1,1:Ny+1), vor_old(1:Nx+1,1:Ny+1)
real*8 psi_new(1:Nx+1,1:Ny+1), psi_old(1:Nx+1,1:Ny+1)
real*8 pressure(1:Nx+1,1:Ny+1)

!%% local variables
```

```
integer :: t = 0
real*8:: error = 1.d0

!%% Output Simulation parameters
call OutputParameters()
Write(*,*) 'Programing Paused, Continue?...'
pause

!%% set up initial flow field
call InitialField(vor_new,vor_old,psi_new,psi_old,u,v,pressure)


!%% Evaluation main loop
do t = 1,t_max
!!! solve vorticity equation
        call solvor(u,v,psi_new,vor_old,vor_new)

!!! solve Streamfunction equation
        call solpsi(vor_new,psi_old,psi_new)

!!! compute velocity components u and v
        call caluv(psi_new,u,v)

!!! Check and Output
        if(mod(t,Output_Interval).EQ.0) then
            call output(vor_new,psi_new,u,v,pressure,t)
        end if

        if(mod(t,Report_Interval).EQ.0 ) then
            call convergence(vor_old,vor_new,psi_old,psi_new,error)
            print 300, t,error
            if(error<eps .AND. t>5000) then                    ! we run at least 5000
time step
                Print*, 'error < epsilon',' Step = ',t
                Exit
            end if
        end if

        ! update field
        vor_old = vor_new
        psi_old = psi_new

end do
```

```
300 format(1x,'Current Time Step = ',I10,', Current Residuls = ',ES25.15,/)

!%% Solve pressure Poisson equation at steady flow state
call solppe(u,v,vor_new,pressure)

!%% output data file
call output(vor_new,psi_new,u,v,pressure,t)

!%% Report and output computing result in a log file
call SimulationReport(t,error)

!%% program end
END PROGRAM main
!-----------------------------------------------------------------------------------------------------
-------------------------------------



!-----------------------------------------------------------------------------------------------------
-----------------------------------------
!%% Output Simulation parameters
SUBROUTINE OutputParameters
use ParametersBlock
implicit none

! file name
character(len=20), parameter:: FileName1='SimulationParameters.txt'

! Print
write(*,100) CPUS
100 format(1x,'Parallel CPUS = ',I10,/)

write(*,101) Output_Interval, Report_Interval, eps
101 format(1x,'Output_Interval = ',I10,', Report_Interval = ',I10,', epsilon =
',ES15.5,/)

write(*,102) Nx,Ny,t_max
102 format(1x,'Nx = 'I10,', Ny = ',I10,', t_max = ',I10,/)

write(*,103) delta_x,delta_y,delta_t
103 format(1x,', delta_x = ',ES15.5,', delta_y = ',ES15.5,', delta_t = ',ES15.5,/)

write(*,104) Re,Rho_0,p_0
104 format(1x,'Re = ',F10.5,', Rho_0 = ',F10.5,', P_0 = 'F10.5,/)
```

```
write(*,105) Lx,U_max,nu
105 format(1x,'Lx = ',ES15.6,', U_max = ',ES15.6,', nu = ',ES15.6,/)


! Output to ReportFile
Open(unit=20,file=Trim(FileName1))

write(20,100) CPUS
write(20,101) Output_Interval, Report_Interval, eps
write(20,102) Nx,Ny,t_max
write(20,103) delta_x,delta_y,delta_t
write(20,104) Re,Rho_0,p_0
write(20,105) Lx,U_max,nu

Close(20)


! Subroutine end
END SUBROUTINE OutputParameters
!-------------------------------------------------------------------------------------------------
-----------------------------------------


!-------------------------------------------------------------------------------------------------
-----------------------------------
!!! set up initial flow field
SUBROUTINE InitialField(vor_new,vor_old,psi_new,psi_old,u,v,pressure)
use ParametersBlock
implicit none

! global variables
real*8, INTENT(INOUT):: u(1:Nx+1,1:Ny+1), v(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: vor_new(1:Nx+1,1:Ny+1), vor_old(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: psi_new(1:Nx+1,1:Ny+1), psi_old(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: pressure(1:Nx+1,1:Ny+1)

! local variables
integer i,j


! initial velocity field
!$OMP PARALLEL DO PRIVATE(i,j)
```

```
SHARED(vor_old,vor_new,psi_old,psi_new,u,v,pressure)
NUM_THREADS(CPUS)
! Macro Varibles
do i=1,Nx+1
    do j = 1,Ny+1

        u(i,j) = 0.d0
        v(i,j) = 0.d0

        vor_old(i,j) = 0.d0
        vor_new(i,j) = 0.d0

        psi_new(i,j) = 0.d0
        psi_old(i,j) = 0.d0

        pressure(i,j) = p_0

    end do
end do
!$OMP END PARALLEL DO



return
END SUBROUTINE InitialField
!-----------------------------------------------------------------------------------------------------
---------------------------------------



!-----------------------------------------------------------------------------------------------------
-----------------------------
!!! solve vorticity equation
SUBROUTINE solvor(u,v,psi_new,vor_old,vor_new)
use ParametersBlock
implicit none

! global variables
real*8, INTENT(IN):: u(1:Nx+1,1:Ny+1), v(1:Nx+1,1:Ny+1)
real*8, INTENT(IN):: psi_new(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: vor_new(1:Nx+1,1:Ny+1), vor_old(1:Nx+1,1:Ny+1)

! local variables
integer i,j
```

```
real*8 A,B,cx,cy,Fox,Foy,x
real*8 vor_meso(1:Nx+1,1:Ny+1)

!!!   Vorticity boundary treatment
!%%   Wall vorticity (Fromm Formula)
! up (Moving lid)
A = 2.d0 / (delta_y**2)
B = 2.d0 / (delta_y)
!$OMP PARALLEL DO PRIVATE(i,x) FIRSTPRIVATE(A,B)
SHARED(vor_old,vor_new,psi_new,vor_meso) NUM_THREADS(CPUS)
do i = 1,Nx+1
      vor_old(i,Ny+1) = A*( psi_new(i,Ny) - psi_new(i,Ny+1) ) + B*u(i,Ny+1)
      vor_new(i,Ny+1) = vor_old(i,Ny+1)
      vor_meso(i,Ny+1) = vor_old(i,Ny+1)
end do
!$OMP END PARALLEL DO

! bottom
!$OMP PARALLEL DO PRIVATE(i) FIRSTPRIVATE(A)
SHARED(vor_old,vor_new,psi_new,vor_meso) NUM_THREADS(CPUS)
do i = 1,Nx+1
      vor_old(i,1) = A*( psi_new(i,2) - psi_new(i,1) )
      vor_new(i,1) = vor_old(i,1)
      vor_meso(i,1) = vor_old(i,1)
end do
!$OMP END PARALLEL DO

! left
A = 2.d0 / (delta_x**2)
!$OMP PARALLEL DO PRIVATE(j) FIRSTPRIVATE(A)
SHARED(vor_old,vor_new,psi_new,vor_meso) NUM_THREADS(CPUS)
do j = 2,Ny
      vor_old(1,j) = A*( psi_new(2,j) - psi_new(1,j) )
      vor_new(1,j) = vor_old(1,j)
      vor_meso(1,j) = vor_old(1,j)
end do
!$OMP END PARALLEL DO

! right
!$OMP PARALLEL DO PRIVATE(j) FIRSTPRIVATE(A)
SHARED(vor_old,vor_new,psi_new,vor_meso) NUM_THREADS(CPUS)
do j = 2,Ny
      vor_old(Nx+1,j) = A*( psi_new(Ny,j) - psi_new(Ny+1,j) )
      vor_new(Nx+1,j) = vor_old(Nx+1,j)
```

```
      vor_meso(Nx+1,j) = vor_old(Nx+1,j)
end do
!$OMP END PARALLEL DO



!%% Vorticity evolution
cx = 0.5d0 * delta_t / delta_x
cy = 0.5d0 * delta_t / delta_x
Fox = nu*delta_t / (delta_x**2)
Foy = nu*delta_t / (delta_y**2)

!$OMP PARALLEL DO PRIVATE(i,j) FIRSTPRIVATE(cx,cy,Fox,Foy)
SHARED(vor_old,vor_meso,u,v) NUM_THREADS(CPUS)
do i = 2,Nx
    do j = 2,Ny
        vor_meso(i,j) = vor_old(i,j) - cx*( (u(i+1,j)*vor_old(i+1,j)) - (u(i-
1,j)*vor_old(i-1,j)) )&
&    - cy*( (u(i,j+1)*vor_old(i,j+1)) - (u(i,j-1)*vor_old(i,j-1)) )&
& + Fox*(vor_old(i+1,j)-2.d0*vor_old(i,j)+vor_old(i-1,j)) +
Foy*(vor_old(i,j+1)-2.d0*vor_old(i,j)+vor_old(i,j-1))
    end do
end do
!$OMP END PARALLEL DO

!$OMP PARALLEL DO PRIVATE(i,j) FIRSTPRIVATE(cx,cy,Fox,Foy)
SHARED(vor_old,vor_new,vor_meso,u,v) NUM_THREADS(CPUS)
do i = 2,Nx
    do j = 2,Ny
        vor_new(i,j) = 0.5d0*(vor_meso(i,j) + (vor_old(i,j) -
cx*( (u(i+1,j)*vor_meso(i+1,j)) - (u(i-1,j)*vor_meso(i-1,j)) )&
&    - cy*( (u(i,j+1)*vor_meso(i,j+1)) - (u(i,j-1)*vor_meso(i,j-1)) )&
& + Fox*(vor_meso(i+1,j)-2.d0*vor_meso(i,j)+vor_meso(i-1,j)) +
Foy*(vor_meso(i,j+1)-2.d0*vor_meso(i,j)+vor_meso(i,j-1))))
    end do
end do
!$OMP END PARALLEL DO



return
END SUBROUTINE solvor
!-----------------------------------------------------------------------------------------------
----------------------------
```

```
!-------------------------------------------------------------------------------------------------
-----------------------------
!!! solve Streamfunction equation
SUBROUTINE solpsi(vor_new,psi_old,psi_new)
use ParametersBlock
implicit none

! global variables
real*8, INTENT(IN):: vor_new(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: psi_old(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: psi_new(1:Nx+1,1:Ny+1)

! local variables
integer i,j,k
real*8 A,B,alpha,beta,eta,theta
real*8:: delta_psi = 0.d0,temp =0.d0

! control varibales
real*8,parameter:: sigma = 1.5
real*8,parameter:: eps_pe = 1.d-4
integer,parameter:: max_k = 5000000

! parameters
A = 1.d0 / (delta_x**2)
B = 1.d0 / (delta_y**2)

alpha = 1 - sigma
beta = sigma/(2.d0*(A+B)) * A
eta = sigma/(2.d0*(A+B)) * B
theta = sigma/(2.d0*(A+B)) * (-1.d0)


! stream function boundary dealing
! up (Moving lid)

!$OMP PARALLEL DO PRIVATE(i) SHARED(psi_new)
NUM_THREADS(CPUS)
do i = 1,Nx+1
      psi_new(i,Ny+1) = 0.d0
end do
!$OMP END PARALLEL DO
```

```
! bottom
!$OMP PARALLEL DO PRIVATE(i) SHARED(psi_new)
NUM_THREADS(CPUS)
do i = 1,Nx+1
      psi_new(i,1) = 0.d0
end do
!$OMP END PARALLEL DO

! left
!$OMP PARALLEL DO PRIVATE(j) SHARED(psi_new)
NUM_THREADS(CPUS)
do j = 2,Ny
      psi_new(1,j) = 0.d0
end do
!$OMP END PARALLEL DO

! right
!$OMP PARALLEL DO PRIVATE(j) SHARED(psi_new)
NUM_THREADS(CPUS)
do j = 2,Ny
      psi_new(Nx+1,j) = 0.d0
end do
!$OMP END PARALLEL DO



! SOR iterative for stream function Poisson equation
do k = 1,max_k
    delta_psi = 0.d0
    do j = 2,Ny
       do i = 2,Nx
            temp = alpha*psi_new(i,j) + beta*(psi_new(i+1,j)+psi_new(i-1,j)) +
eta*(psi_new(i,j+1)+psi_new(i,j-1)) + theta*vor_new(i,j)
            if(temp - psi_new(i,j) > delta_psi) then
                delta_psi = temp - psi_new(i,j)
            end if
            psi_new(i,j) = temp
       end do
    end do
!    print*, delta_psi
    if(delta_psi < eps_pe) then
!         print*, 'Stream function Possion equation convergence!'
```

```
          exit
      end if
end do


! Subroutine solpsi end
END SUBROUTINE solpsi
!-------------------------------------------------------------------------------------------------------
----------------------------


!-------------------------------------------------------------------------------------------------------
--------------------------
!!! compute velocity components u and v
SUBROUTINE caluv(psi_new,u,v)
use ParametersBlock
implicit none

! global variables
real*8, INTENT(IN):: psi_new(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: u(1:Nx+1,1:Ny+1), v(1:Nx+1,1:Ny+1)

!local variables
integer i,j
real*8 x

!physical boundary condition
do i=1,Nx+1
   x = (i-1.d0)/Nx
   u(i,1) = 0.0d0
   v(i,1) = 0.0d0
   u(i,Ny+1) = U_max*16.d0*(x**2)*(1.d0-x)**2
   v(i,Ny+1) = 0.0d0
end do

do j=1,Ny
     u(1,j) = 0.0d0
     u(Nx+1,j) = 0.0d0
     v(1,j) = 0.0d0
     v(Nx+1,j) = 0.0d0
end do
```

```
!$OMP PARALLEL DO PRIVATE(i,j) SHARED(u,v,psi_new)
NUM_THREADS(CPUS)
do i=2,Nx
    do j=2,Ny
        u(i,j) = 0.5d0*(psi_new(i,j+1)-psi_new(i,j-1))/delta_y
        v(i,j) = -0.5d0*(psi_new(i+1,j)-psi_new(i-1,j))/delta_x
    end do
end do
!$OMP END PARALLEL DO

return
! Subroutine end
END SUBROUTINE caluv
!-------------------------------------------------------------------------------------------------
----------------------------




!-------------------------------------------------------
SUBROUTINE convergence(vor_old,vor_new,psi_old,psi_new,error)
use ParametersBlock
use omp_lib
implicit none

! global varibales
real*8, INTENT(IN):: vor_old(1:Nx+1,1:Ny+1), vor_new(1:Nx+1,1:Ny+1)
real*8, INTENT(IN):: psi_old(1:Nx+1,1:Ny+1), psi_new(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: error

! local varibales
integer i,j
real*8:: num1 = 0.d0, den1 = 0.d0, num2 = 0.d0, den2 = 0.d0
real*8 temp1, temp2

! sum whole field
!$OMP PARALLEL DO PRIVATE(i,j)
SHARED(vor_old,vor_new,psi_old,psi_new)
REDUCTION(+:num1,den1,num2,den2) NUM_THREADS(CPUS)
do i = 2,Nx
    do j = 2,Ny
        num1 = num1 + (vor_new(i,j) - vor_old(i,j))**2
        den1 = den1 + vor_old(i,j)**2
```

```
          num2 = num2 + (psi_new(i,j) - psi_old(i,j))**2
          den2 = den2 + psi_new(i,j)**2
      end do
end do
!$OMP END PARALLEL DO

temp1 = sqrt(num1) / sqrt(den1)
temp2 = sqrt(num2) / sqrt(den2)

if(temp1 > temp2) then
       error = temp1
else
       error = temp2
end if

END SUBROUTINE convergence
!-------------------------------------------------------




!-------------------------------------------------------------------------------------------------------
------------------------------
SUBROUTINE solppe(u,v,vor_new,pressure)
use ParametersBlock
implicit none

! global variables
real*8, INTENT(IN):: u(1:Nx+1,1:Ny+1),v(1:Nx+1,1:Ny+1)
real*8, INTENT(IN):: vor_new(1:Nx+1,1:Ny+1)
real*8, INTENT(INOUT):: pressure(1:Nx+1,1:Ny+1)

! local variables
integer i,j,k
real*8 A,B,C,alpha,beta,eta,theta
real*8 delta_pressure,temp
real*8 Sp(1:Nx+1,1:Ny+1)

! control varibales
real*8,parameter:: sigma = 1.5
real*8,parameter:: eps_pe = 1.d-5
integer,parameter:: max_k = 10000000

! parameters
A = 1.d0 / (delta_x**2)
```

```
B = 1.d0 / (delta_y**2)
C = 0.25d0/(delta_x*delta_y)

alpha = 1 - sigma
beta = sigma/(2.d0*(A+B)) * A
eta = sigma/(2.d0*(A+B)) * B
theta = sigma/(2.d0*(A+B)) * (-1.d0)


!!! pressure boundary dealing

! bottom
pressure(1,1) = p_0
pressure(2,1) = pressure(1,1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(2,1) + 4.d0*vor_new(2,2) - vor_new(2,3) )

!$OMP PARALLEL DO PRIVATE(i) SHARED(pressure)
NUM_THREADS(CPUS)
do i = 2,Nx
      pressure(i+1,1) = pressure(i-1,1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(i,1) + 4.d0*vor_new(i,2) - vor_new(i,3) )
end do
!$OMP END PARALLEL DO

! right
pressure(Nx+1,2) = pressure(Nx+1,1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(Nx+1,2) + 4.d0*vor_new(Nx,2) - vor_new(Nx-1,2) )
!$OMP PARALLEL DO PRIVATE(j) SHARED(pressure)
NUM_THREADS(CPUS)
do j = 2,Ny
      pressure(Nx+1,j+1) = pressure(Nx+1,j-1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(Nx+1,j) + 4.d0*vor_new(Nx,j) - vor_new(Nx-1,j) )
end do
!$OMP END PARALLEL DO

! up (Moving lid)
pressure(Nx,Ny+1) = pressure(Nx+1,Ny+1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(Nx+1,Ny+1) + 4.d0*vor_new(Nx+1,Ny) - vor_new(Nx+1,Ny-1) )
!$OMP PARALLEL DO PRIVATE(i) SHARED(pressure)
NUM_THREADS(CPUS)
do i = Nx,-1,2
      pressure(i-1,Ny+1) = pressure(i+1,1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(i,Ny+1) + 4.d0*vor_new(i,Ny) - vor_new(i,Ny-1) )
end do
```

```
!$OMP END PARALLEL DO

! left
pressure(1,Ny) = pressure(1,Ny+1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(1,Ny) + 4.d0*vor_new(2,Ny) - vor_new(3,Ny) )
!$OMP PARALLEL DO PRIVATE(j) SHARED(pressure)
NUM_THREADS(CPUS)
do j = Ny,-1,2
      pressure(1,j-1) = pressure(1,j+1) - (delta_x)*nu/(2.d0*delta_y)*( -
3.d0*vor_new(1,j) + 4.d0*vor_new(2,j) - vor_new(3,j) )
end do
!$OMP END PARALLEL DO

! SOR iterative for pressure poisson equation
! sorce term
do i = 2,Nx
    do j = 2,Ny
        Sp(i,j) = C*( (u(i+1,j) - u(i-1,j))*(v(i,j+1) - v(i,j-1)) - (u(i,j+1) - u(i,j-
1))*(v(i+1,j) - v(i-1,j)) )
    end do
end do

! iterative
do k = 1,max_k
    delta_pressure = 0.d0
    do j = 2,Ny
       do i = 2,Nx
            temp = alpha*pressure(i,j) + beta*(pressure(i+1,j)+pressure(i-1,j)) +
eta*(pressure(i,j+1)+pressure(i,j-1)) + theta*Sp(i,j)
            if(temp - pressure(i,j) > delta_pressure) then
                delta_pressure = temp - pressure(i,j)
            end if
            pressure(i,j) = temp
       end do
    end do
    if(delta_pressure < eps_pe) then
        print*, 'Pressure Possion equation convergence!'
        exit
    end if
end do

! Subroutine end
END SUBROUTINE solppe
!-----------------------------------------------------------------------------------------
```

```
------------------------------



!-------------------------------------------------------------------------------------------------
----------------------------------------
!!! output data file
SUBROUTINE output(vor_new,psi_new,u,v,pressure,t)
use ParametersBlock
implicit none

integer, INTENT(IN):: t
real*8, INTENT(IN)::
vor_new(Nx+1,Ny+1),psi_new(Nx+1,Ny+1),u(Nx+1,Ny+1),v(Nx+1,Ny+1),press
ure(Nx+1,Ny+1)

real*8
u_out(Nx+1,Ny+1),v_out(Nx+1,Ny+1),vor_out(Nx+1,Ny+1),psi_out(Nx+1,Ny+
1)
character(len=100) Char_Temp,Nx_Temp,Ny_Temp
integer i,j

Write(Char_Temp,*) t
Write(Nx_Temp,*) Nx+1
Write(Ny_Temp,*) Ny+1
open(20,file='FlowDataAtStep'//Trim(Char_Temp)//'.dat')

Write(20,*) 'VARIABLES = X, Y, U, V, P, Omega, Psi'
Write(20,*) 'Zone T="", I ='//Trim(Nx_Temp)//', J ='//Trim(Ny_Temp)//', F =
Point'

! Dimensionless
do i = 1,Nx+1
    do j = 1,Ny+1
        u_out(i,j) = u(i,j) / U_max
        v_out(i,j) = v(i,j) / U_max
        vor_out(i,j) = vor_new(i,j) /U_max * Lx
        psi_out(i,j) = psi_new(i,j) / U_max / Lx
    end do
end do

! Output
do j = 1,Ny+1
    do i = 1,Nx+1
```

```
        Write(20,200) dble(i-1)/dble(Lx),dble(j-
1)/dble(Lx),u_out(i,j),v_out(i,j),pressure(i,j),vor_out(i,j),Psi_out(i,j)
    end do
end do
200 format(1x,7ES24.15)

close(20)

END SUBROUTINE output
!-----------------------------------------------------------------------------------------------
----------------------------------------


!-----------------------------------------------------------------------------------------------
-----------------------------------------
SUBROUTINE SimulationReport(t,error)
use ParametersBlock
implicit none

real*8, INTENT(IN):: t, error

! %% Compute Log File
open(20,file='Report.log')


write(20,*)
write(20,*)
'************************************************************'
write(20,*) 'This program sloves Lid Driven Cavity Flow problem'
write(20,*) 'using Vorticity-Stream function Methods'
write(20,*) 'nx =',nx,',            ny =',ny
write(20,*) 'Re=',Re
write(20,*) 'dt =', delta_t
write(20,*) 'eps =',eps
write(20,*) 'itc =',t
write(20,*) 'Developing time=',delta_t*t,'s'
write(20,*)
'************************************************************'
write(20,*)
write(20,200) t, error
200 format(1x,'Final Time Step:',I10,', Final Residual:',ES25.15)
 close(20)

END SUBROUTINE SimulationReport
```

```
!----------
```