2011 PLSI 병렬컴퓨팅 경진대회 문제

01. 대학원팀

- 경진 내용
  - ○ 주어진 순차코드를 병렬화하여 성능향상도(획득점수)를 측정
  - ○ 점수 = (순차코드 수행시간) / (병렬화코드 수행시간)

- 경진 환경
  - ○ 프로그래밍 언어: C, Fortran
  - ○ 순차코드는 50라인 내외로 작성하여 제공됨
  - ○ 참가자들은 컴파일러(intel, gnu, pgi) 선택가능
  - ○ 시간제한
    - ■ 6일 오후 1시~6시(5시간 2문제), 7일 오전 9시~12시(3시간 1문제)
    - ■ 제한 시간 내 병렬화코드를 제출
    - ■ 제출 파일: 병렬화코드, 병렬화코드 수행을 위한 Makefile
  - ○ 특정 라이브러리 및 컴파일러 최적화 옵션 사용 불가

- 출제 문제 유형
  - ○ 지난해의 체험대회를 바탕으로 일반적인 문제 출제를 원칙으로 하며, 문제해결을 통해 병렬화기법을 익힐 수 있는데 초점을 맞춤
  - ○ 예상 문제
    - ■ Monte Carlo Method를 이용한 과학문제 해결
    - ■ Molecular Dynamics(MD) 문제 - gnuplot 등을 통한 visualization
    - ■ Finite Difference Method(FDM) 문제

02. 학부팀

- 경진 내용 및 경진 환경은 대학원팀과 동일

- 출제 문제 유형
  - ○ 예상 문제
    - ■ 적분 문제 $\int_0^\infty \frac{\sin(x)}{x} dx = \frac{\pi}{2}$
    - ■ Taylor expansion : $\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$
    - ■ Matrix 곱하기

01. [학부팀 유형 1] - 수치적분

아래 적분식의 수치적분을 이용하여 PI의 근사값을 구하는 순차코드가 주어져 있다. 이 코드를 병렬화(MPI) 하여 최대 성능을 얻으시오. (CPU는 최대 16개 까지 사용)

$$\int_0^{\infty} \frac{2\sin(x)}{x} dx = \pi \quad => \quad \sum_1^n \frac{2\sin(x)}{x} \Delta x \approx \pi$$

○ 평가 방법
- 병렬 코드, 실행 파일, 컴파일 환경(컴파일러, 옵션, 사용 MPI 라이브러리) 제출.
- 순차 코드와 결과가 일치하는 병렬 코드에 대하여 성능 향상도를 점수화.

○ 예제 수행 시간
- 순차 코드 : 347.908 초
- 병렬 코드 : 42.206 초
- 성능 향상도 : 8.243

02. [학부팀 유형 2] - 행렬곱

주어진 (n,n) 행렬에 대하여 A·B를 구하고 모든 원소의 합을 구하는 순차 코드를 병렬화(MPI) 하여 최대 성능을 얻으시오. (CPU는 최대 16개 까지 사용)

○ 평가 방법
- 병렬 코드, 실행 파일, 컴파일 환경(컴파일러, 옵션, 사용 MPI 라이브러리) 제출.
- 순차 코드와 결과가 일치하는 병렬 코드에 대하여 성능 향상도를 점수화.

○ 예제 수행 시간
- 순차 코드 : 247.670 초
- 병렬 코드 : 39.488 초
- 성능 향상도 : 6.272

03. [학부팀 유형 3] - Taylor 급수를 이용한 sin(x) 구하기

아래의 sin(x)에 대한 Taylor 급수를 이용하여 근사값을 구하는 순차 코드를 병렬화(MPI) 하여 최대 성능을 얻으시오. (CPU는 최대 16개 까지 사용)

○ 평가 방법
- 병렬 코드, 실행 파일, 컴파일 환경(컴파일러, 옵션, 사용 MPI 라이브러리) 제출.
- 순차 코드와 결과가 일치하는 병렬 코드에 대하여 성능 향상도를 점수화.

○ 예제 수행 시간
- 순차 코드 : 108.717 초
- 병렬 코드 : 8.772 초
- 성능 향상도 : 12.394

04. [대학원팀 유형 1] - 2D Random Walk

1000000개 입자에 대한 1000번의 2D Random Walk를 수행하고 분포를 구하는 순차 코드를 병렬화(MPI) 하여 최대 성능을 얻으시오. (CPU는 최대 16개 까지 사용)

○ 평가 방법
- 병렬 코드, 실행 파일, 컴파일 환경(컴파일러, 옵션, 사용 MPI 라이브러리) 제출.
- 순차 코드와 결과가 일치하는 병렬 코드에 대하여 성능 향상도를 점수화.

○ 예제 수행 시간
- 순차 코드 : 28.370 초
- 병렬 코드 : 4.993 초
- 성능 향상도 : 5.682

05. [대학원팀 유형 2] - Molecular Dynamics

주어진 1000개 입자의 초기 위치에 대하여 각각의 입자는 1/R(|x1-x2|)의 힘을 받는다고 하면, 10000번의 step 후의 위치를 구할 수 있다. 순차 코드에 대하여 MPI 병렬화를 이용하여 최대 성능을 얻으시오. (CPU는 최대 16개 까지 사용)

○ 평가 방법
- 병렬 코드, 실행 파일, 컴파일 환경(컴파일러, 옵션, 사용 MPI 라이브러리) 제출.
- 순차 코드와 결과가 일치하는 병렬 코드에 대하여 성능 향상도를 점수화.

○ 예제 수행 시간
- 순차 코드 : 29.153 초
- 병렬 코드 : 9.761 초
- 성능 향상도 : 2.987

06. [대학원팀 유형 3] - 2D FDM

주어진 2D 경계조건에서 FDM을 이용하면 유일해를 구할 수 있다. 300X300 grid를 이용하여 FDM의 해를 구하는 순차 코드에 대하여 MPI 병렬화를 이용하여 최대 성능을 얻으시오. (CPU는 최대 16개 까지 사용)

○ 평가 방법
- 병렬 코드, 실행 파일, 컴파일 환경(컴파일러, 옵션, 사용 MPI 라이브러리) 제출.
- 순차 코드와 결과가 일치하는 병렬 코드에 대하여 성능 향상도를 점수화.

○ 예제 수행 시간

- 순차 코드 : 44.202 초
- 병렬 코드 : 7.554 초
- 성능 향상도 : 5.851

01. 학부팀 유형 1 코드

순차 코드 (Fortran)

```
program integration
implicit none

real(8) :: pi, x, x_max
integer(8) :: num_step, i

pi=0.0d0
x=0.0d0
x_max=100000.0d0
num_step=5000000000

do i = 1, num_step
 x = x_max*real(i,8)/real(num_step,8)
 pi = pi + 2.0d0*sin(x)/x*x_max/real(num_step,8)
end do
print *,'PI = ',pi
end program integration
```

병렬 코드 (Fortran)

```
program integration

implicit none
include 'mpif.h'

real(8) :: pi, x, x_max, sum
integer(8) :: num_step, i, is, ie
integer(4) :: myrank, nprocs, ierr

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)

sum=0.0d0
x=0.0d0
x_max=100000.0d0
num_step=5000000000

is = 1 ; ie = num_step
call para_range(is,ie,nprocs,myrank,is,ie)

do i = is, ie
 x = x_max*real(i,8)/real(num_step,8)
 sum = sum + 2.0d0*sin(x)/x*x_max/real(num_step,8)
end do
call MPI_REDUCE(sum,pi,1,MPI_DOUBLE_PRECISION,MPI_SUM,0,MPI_COMM_WORLD,ierr)
if(myrank.eq.0) then
 print *, 'PI = ',pi
endif

call MPI_FINALIZE(ierr)
stop

contains
  subroutine para_range(n1,n2,nprocs,irank,ista,iend)
  implicit none
  integer              :: nprocs,irank
  integer(8)           :: np,ir
  integer(8)           :: n1,n2,ista,iend
  integer(8)           :: iwork1,iwork2
  np=int(nprocs,8)
  ir=int(irank,8)
  iwork1 = (n2 - n1 + 1) / np
  iwork2 = mod(n2 - n1 + 1, np)
  ista = ir * iwork1 + n1 + min(ir,iwork2)
```

```
    iend = ista + iwork1 -1
    if(iwork2 .gt. ir) iend = iend + 1
    return
    end subroutine

end program integration
```

02. 학부팀 유형 2 코드

순차 코드 (Fortran)

```
program matrix
implicit none

integer :: i,j,k,n
real,dimension(:,:),allocatable :: a,b,c
real :: sum
n=5000
sum=0.0
allocate (a(n,n))
allocate (b(n,n))
allocate (c(n,n))
c=0.0

do j=1,n
 do i=1,n
  a(i,j)=real(i)/real(j)
  b(i,j)=real(j)/real(i)
 end do
end do

do k=1,n
 do j=1,n
  do i=1,n
   c(i,j)=c(i,j)+a(i,k)*b(k,j)
  end do
 end do
end do

do j=1,n
 do i=1,n
  sum=sum+c(i,j)
 enddo
enddo
print *, 'SUM = ',sum
deallocate(a,b,c)

end program matrix
```

병렬 코드 (Fortran)

```
program matrix
implicit none
include 'mpif.h'

integer :: i,j,k,n,is,ie
integer(4) :: myrank, nprocs, ierr
real,dimension(:,:),allocatable :: a,b,c
real :: sum,psum

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)

n=5000
sum=0.0
psum=0.0
allocate (a(n,n))
allocate (b(n,n))
allocate (c(n,n))
c=0.0

is=1;ie=n
call para_range(is,ie,nprocs,myrank,is,ie)

do j=1,n
 do i=1,n
  a(i,j)=real(i)/real(j)
  b(i,j)=real(j)/real(i)
 end do
end do

do k=1,n
 do j=is,ie
  do i=1,n
   c(i,j)=c(i,j)+a(i,k)*b(k,j)
  end do
 end do
end do

do j=is,ie
 do i=1,n
  psum=psum+c(i,j)
 enddo
enddo
call MPI_REDUCE(psum,sum,1,MPI_REAL,MPI_SUM,0,MPI_COMM_WORLD,ierr)
```

```
if(myrank.eq.0)  then
  print *,  'SUM  =  ',sum
endif

deallocate(a,b,c)

call  MPI_FINALIZE(ierr)
stop

contains
   subroutine  para_range(n1,n2,nprocs,irank,ista,iend)
   implicit  none
   integer                 ::  nprocs,irank
   integer             ::  np,ir
   integer             ::  n1,n2,ista,iend
   integer             ::  iwork1,iwork2
   np=nprocs
   ir=irank
   iwork1  =  (n2  -  n1  +  1)  /  np
   iwork2  =  mod(n2  -  n1  +  1,  np)
   ista  =  ir  *  iwork1  +  n1  +  min(ir,iwork2)
   iend  =  ista  +  iwork1  -1
   if(iwork2  .gt.  ir)  iend  =  iend  +  1
   return
   end  subroutine

end  program  matrix
```

03. 학부팀 유형 3 코드

순차 코드 (Fortran)

```
program taylor
implicit none
real(8) :: x, mysin, tmp, x_max, step
integer(4) :: i, j, k, n, num_step

x_max=4.0d0
x=0.0d0
num_step=10000
n=1000
step=x_max/real(num_step,8)

do i=0,num_step
 mysin=0.0d0
 do j=1,n
  tmp=1.0d0
  do k=1,2*j-1
   tmp=tmp*x/real(k,8)
  enddo
  if(mod(j,2)) then
   mysin=mysin+tmp
  else
   mysin=mysin-tmp
  endif
 enddo
 print *, x, mysin, sin(x)
 x=x+step
enddo

end program taylor
```

병렬 코드 (Fortran)

```
program taylor
implicit none
include 'mpif.h'

real(8) :: x, mysin, tmp, x_max, step
integer(4) :: i, j, k, n, num_step
integer(4) :: myrank, nprocs, ierr, is, ie

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)

x_max=4.0d0
num_step=10000
n=1000
step=x_max/real(num_step,8)

is=0;ie=num_step
call para_range(is,ie,nprocs,myrank,is,ie)

do i=is,ie
 x=real(i,8)*step
 mysin=0.0d0
 do j=1,n
  tmp=1.0d0
  do k=1,2*j-1
   tmp=tmp*x/real(k,8)
  enddo
  if(mod(j,2)) then
   mysin=mysin+tmp
  else
   mysin=mysin-tmp
  endif
 enddo
 print *, x, mysin, sin(x)
enddo
call MPI_FINALIZE(ierr)

contains
  subroutine para_range(n1,n2,nprocs,irank,ista,iend)
  implicit none
  integer              :: nprocs,irank
  integer              :: np,ir
  integer              :: n1,n2,ista,iend
  integer              :: iwork1,iwork2
```

```
    np=nprocs
    ir=irank
    iwork1 = (n2 - n1 + 1) / np
    iwork2 = mod(n2 - n1 + 1, np)
    ista = ir * iwork1 + n1 + min(ir,iwork2)
    iend = ista + iwork1 -1
    if(iwork2 .gt. ir) iend = iend + 1
    return
    end subroutine

end program taylor
```

04. 대학원팀 유형 1 코드

순차 코드 (Fortran)

```
program randomwalk

implicit none
integer  :: i,j,rtc
integer, parameter :: N_P=1000000, N_W=1000
integer, dimension(N_P) :: part_x, part_y
integer, dimension(:,:),allocatable :: pos
real :: seed, rand,temp

part_x=0
part_y=0
call srand(0.5)

do i=1, N_P
    do j=1,N_W
        temp=rand(0)
        if(temp <= 0.25) then
            part_x(i)=part_x(i)+1
        elseif(temp <= 0.5) then
            part_y(i)=part_y(i)+1
        elseif(temp <=0.75) then
            part_x(i)=part_x(i)-1
        else
            part_y(i)=part_y(i)-1
        endif
     enddo
enddo

allocate(pos(-N_W:N_W,-N_W:N_W))

pos=0
do i=1,N_P
    pos(part_x(i),part_y(i))=pos(part_x(i),part_y(i))+1
enddo

do i=-N_W,N_W,2
 do j=-N_W,N_W,2
  if(pos(i,j) .ne. 0) print*, i,j, pos(i,j)
 enddo
enddo

deallocate(pos)
```

```
end  program  randomwalk
```

병렬 코드 (Fortran)

```
program randomwalk
implicit none
include 'mpif.h'
integer  :: mysize, myrank, ierr, pos_count, ista, iend
integer  :: i,j,d,min_p,max_p,rtc
integer, parameter :: N_P=1000000, N_W=1000
integer, dimension(N_P) :: part_x, part_y
integer, dimension(:,:),allocatable :: pos,pos_all
real :: seed, rand,temp
part_x=0
part_y=0
call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,mysize, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank, ierr)

call srand(0.5+myrank*0.1)
call para_range(1,N_P,mysize,myrank,ista,iend)
do i=ista,iend
    do j=1,N_W
        temp=rand(0)
        if(temp <= 0.25) then
            part_x(i)=part_x(i)+1
        elseif(temp <= 0.5) then
            part_y(i)=part_y(i)+1
        elseif(temp <=0.75) then
            part_x(i)=part_x(i)-1
        else
            part_y(i)=part_y(i)-1
        endif
     enddo
enddo
allocate(pos(-N_W:N_W,-N_W:N_W))
allocate(pos_all(-N_W:N_W,-N_W:N_W))
pos=0
do i=1,N_P
    pos(part_x(i),part_y(i))=pos(part_x(i),part_y(i))+1
enddo
pos_count=size(pos)
call MPI_REDUCE(pos,pos_all,pos_count,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)
if(myrank==0) then
 pos=pos_all
 do i=-N_W,N_W,2
  do j=-N_W,N_W,2
```

```
      if(pos(i,j) .ne. 0) print*, i,j, pos(i,j)
    enddo
  enddo
endif
deallocate(pos)
deallocate(pos_all)

call MPI_FINALIZE(ierr)

CONTAINS
   subroutine para_range(n1,n2,nprocs,irank,ista,iend)
   implicit none
   integer                :: n1,n2,nprocs,irank,ista,iend
   integer                :: iwork1,iwork2
   iwork1 = (n2 - n1 + 1) / nprocs
   iwork2 = mod(n2 - n1 + 1, nprocs)
   ista = irank * iwork1 + n1 + min(irank,iwork2)
   iend = ista + iwork1 -1
   if(iwork2 .gt. irank) iend = iend + 1
   return
   end subroutine

end program randomwalk
```

05. 대학원팀 유형 2 코드

순차 코드 (Fortran)

```
program md

implicit none
integer  :: i,j,k
integer, parameter :: N_P=1000, step=10000
real, dimension(N_P) :: x, force
real :: f_jk

do i=1,N_P
 read *, x(i)
 x(i)=x(i)*10000000.0
enddo

do i=1,step
 force=0.0
 do j=1,N_P-1
  do k= j+1,N_P
   f_jk = 1.0/(x(k)-x(j))
   force(j)=force(j)+f_jk
   force(k)=force(k)-f_jk
  enddo
 enddo
 do j=1,N_P
  x(j)=x(j)+force(j)
 enddo
enddo

do i=1,N_P
 print *, i,x(i)
enddo

end program md
```

병렬 코드 (Fortran)

```
program md

implicit none
include 'mpif.h'
integer :: mysize, myrank, ierr, ista, iend
integer :: i,j,k
integer, parameter :: N_P=1000, step=10000
real, dimension(N_P) :: x, force, force_all
real :: f_jk

call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,mysize, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank, ierr)

if(myrank==0) then
 do i=1,N_P
  read *, x(i)
  x(i)=x(i)*10000000.0
 enddo
 print *,'start'
endif

call MPI_BCAST(x,N_P,MPI_REAL,0,MPI_COMM_WORLD,ierr)

do i=1,step
 force=0.0
 call para_range(1,N_P-1,mysize,myrank,ista,iend)
 do j=ista,iend
  do k= j+1,N_P
   f_jk = 1.0/(x(k)-x(j))
   force(j)=force(j)+f_jk
   force(k)=force(k)-f_jk
  enddo
 enddo
 call MPI_REDUCE(force,force_all,N_P,MPI_REAL,MPI_SUM,0,MPI_COMM_WORLD,ierr)
 if(myrank==0) then
  do j=1,N_P
   x(j)=x(j)+force_all(j)
  enddo
 endif
enddo

if(myrank==0) then
 do i=1,N_P
  print *, i,x(i)
```

```
  enddo
endif

call  MPI_FINALIZE(ierr)

CONTAINS
   subroutine  para_range(n1,n2,nprocs,irank,ista,iend)
   implicit  none
   integer                 :: n1,n2,nprocs,irank,ista,iend
   integer                 :: iwork1,iwork2
   iwork1 = (n2 - n1 + 1) / nprocs
   iwork2 = mod(n2 - n1 + 1, nprocs)
   ista = irank * iwork1 + n1 + min(irank,iwork2)
   iend = ista + iwork1 -1
   if(iwork2 .gt. irank) iend = iend + 1
   return
   end  subroutine

end  program  md
```

06. 대학원팀 유형 3 코드

순차 코드 (Fortran)

```
      program  FDM2D
      integer  im,jm,im1,jm1
      parameter(im=300,jm=300)
      integer  is,ie,js,je
      integer  iter,itermax,nprt
      real     tolerance
      real     bc(4)  !left,right,bottom,top
      real     u(0:im+1,0:jm+1)
      real     error

! read  input  data
      itermax=100000
      nprt=500
      tolerance  =  1.0e-7
      bc(1)  =  10.0
      bc(2)  =  10.0
      bc(3)  =  10.0
      bc(4)  =  20.0

! initialize
      im1=im+1
      jm1=jm+1
      do  j=0,jm1
      do  i=0,im1
        u(i,j)  =  0.0
      end  do
      end  do

! boundary  conditions
      do  j=0,jm1
        u(0   ,j)   =  bc(1)  !left
        u(im1,j)   =  bc(2)  !right
      end  do

      do  i=0,im1
        u(i,0   )   =  bc(3)  !bottom
        u(i,jm1)   =  bc(4)  !top
      end  do

! set  computation  range
      is  =  1
      ie  =  im
```

```
      js = 1
      je = jm

! main routine
      iter = 0
      error = 1000.
! ***********************************************************************
      do while(iter.le.itermax.and.error.gt.tolerance)
        call jacobi(u,im,jm,is,ie,js,je,error)
        iter = iter + 1
        if(mod(iter,nprt).eq.0) write(*,100) iter,error
      end do
! ***********************************************************************

      print*,'Error=',error
      print*,'Converged after ',iter,'iteration'
100   format('Iteration=',i6,' Error=',e9.4)

!       OPEN(UNIT=10, FILE='outFDM2Seq.dat', STATUS='REPLACE', ACTION='WRITE')
      DO j=1,jm
!         WRITE (*, '128(F7.4,1x)'), (u(i,j), i=1,im)
!         WRITE (10, '128(F7.4,1x)'), (u(i,j), i=1,im)
      ENDDO
!       CLOSE(10)
      stop
      end


      subroutine jacobi(u,im,jm,is,ie,js,je,error)
      integer im,jm,is,ie,js,je
      integer i,j
      real error
      real u(0:im+1,0:jm+1), uo(0:im+1,0:jm+1)

! store old data
      do j=0,jm+1
      do i=0,im+1
        uo(i,j) = u(i,j)
      end do
      end do

! jacobi
      do j=js,je
      do i=is,ie
        u(i,j) = (uo(i-1,j)+uo(i+1,j)+uo(i,j-1)+uo(i,j+1))/4.0
      end do
      end do
```

```
! error
        error = 0.0
        do j=js,je
        do i=is,ie
          error = error + (u(i,j) - uo(i,j))**2
        end do
        end do
        return
        end
```

병렬 코드 (Fortran)

```
      PROGRAM FDM2D
      IMPLICIT NONE
      INCLUDE 'mpif.h'
      INTEGER :: nprocs, myrank, viewrank, ista, iend, jsta, jend, ierr, status(MPI_STATUS_SIZE), inext,
iprev, isend1, isend2, irecv1, irecv2
      INTEGER :: i, j, k
      INTEGER :: im,jm,im1,jm1, njm
      PARAMETER(im=300,jm=300)
      INTEGER :: is,ie,js,je
      INTEGER :: iter,itermax,nprt
      REAL    ::  tolerance
      REAL    :: bc(4) !left,right,bottom,top
      REAL    :: u(0:im+1,0:jm+1), newu(0:im+1, 0:jm+1), works1(jm), works2(jm), workr1(jm),
workr2(jm)
      REAL    :: error, errorsum
      CHARACTER*2 :: number
      CHARACTER*30 :: filename(0:10)
      INTEGER :: irecv(0:3), iscnt, ircnt(0:3), idisp(0:3)=(/1,33,65,97/)
      viewrank = 3
! mpi ready
      CALL MPI_INIT(ierr)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
      CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
      CALL para_range(1,im, nprocs, myrank, ista, iend)

! read input data
      itermax=100000
      nprt=500
      tolerance = 1.0e-7
      bc(1) = 10.0
      bc(2) = 10.0
      bc(3) = 10.0
      bc(4) = 20.0

! initialize
      im1=im+1
      jm1=jm+1
      DO j=0,jm1
      DO i=0,im1
        u(i,j) = 0.0
      END DO
      END DO

! boundary conditions
```

```
      IF(myrank == 0) THEN
        DO j=0, jm+1
          u(0   ,j)   = bc(1) !left
        ENDDO
      ENDIF

      IF(myrank == nprocs-1) THEN
        DO j=0, jm+1
          u(im1,j)   = bc(2) !right
        END DO
      ENDIF

      DO i=ista, iend
        u(i,0   )   = bc(3) !bottom
        u(i,jm1)   = bc(4) !top
      END DO
! set computation range
      is = ista
      ie = iend
      js = 1
      je = jm

      inext = myrank + 1
      iprev = myrank - 1
      IF(myrank == 0) iprev = MPI_PROC_NULL
      IF(myrank == nprocs -1 ) inext = MPI_PROC_NULL

! main routine
      iter = 0
      error = 1000.
      errorsum = 1000.
! ***********************************************************************
      DO WHILE(iter.LE.itermax.AND.errorsum.GT.tolerance)
!       do while(iter.le.itermax)
        IF(myrank /= nprocs-1) THEN
          DO j=1, jm
            works1(j) = u(iend, j)
          ENDDO
        ENDIF
        IF(myrank /= 0) THEN
          DO j=1, jm
            works2(j) = u(ista, j)
          ENDDO
        ENDIF
        CALL MPI_ISEND(works1, jm, MPI_REAL, inext, 1, MPI_COMM_WORLD, isend1, ierr)
        CALL MPI_ISEND(works2, jm, MPI_REAL, iprev, 1, MPI_COMM_WORLD, isend2, ierr)
```

```
      CALL MPI_IRECV(workr1, jm, MPI_REAL, iprev, 1, MPI_COMM_WORLD, irecv1, ierr)
      CALL MPI_IRECV(workr2, jm, MPI_REAL, inext, 1, MPI_COMM_WORLD, irecv2, ierr)
      CALL MPI_WAIT(isend1, status, ierr)
      CALL MPI_WAIT(isend2, status, ierr)
      CALL MPI_WAIT(irecv1, status, ierr)
      CALL MPI_WAIT(irecv2, status, ierr)
      IF(myrank /= 0)THEN
        DO j=1, jm
          u(ista-1, j)=workr1(j)
        ENDDO
      ENDIF
      IF(myrank /= nprocs-1) THEN
        DO j=1, jm
          u(iend+1, j)= workr2(j)
        ENDDO
      ENDIF
      call jacobi(u,im,jm,is,ie,js,je,error)
      iter = iter + 1
      CALL MPI_ALLREDUCE(error, errorsum, 1, MPI_REAL, MPI_SUM, MPI_COMM_WORLD, ierr)
      if((mod(iter,nprt).eq.0) .and. myrank == viewrank) write(*,100) iter,errorsum
    end do
! ********************************************************************

100    format('Iteration=',i6,' Error=',e9.4)
    IF(myrank == 0) THEN
      print*,'Error=',errorsum
      print*,'Converged after ',iter,'iteration'
200    format('Iteration=',i6,' Error=',e9.4)
    ENDIF

! delete communication data
    IF(myrank /= 0)THEN
      DO j=1, jm
        u(ista-1, j)=0.0
      ENDDO
    ENDIF
    IF(myrank /= nprocs-1) THEN
      DO j=1, jm
        u(iend+1, j)=0.0
      ENDDO
    ENDIF

    CALL MPI_REDUCE(u, newu, (im+2)*(jm+2), MPI_REAL, MPI_SUM, 0, MPI_COMM_WORLD, ierr)

    CALL MPI_FINALIZE(ierr)
    STOP
```

```
CONTAINS
  subroutine para_range(n1,n2,nprocs,irank,ista,iend)
  implicit none
  integer                :: n1,n2,nprocs,irank,ista,iend
  integer                :: iwork1,iwork2
  iwork1 = (n2 - n1 + 1) / nprocs
  iwork2 = mod(n2 - n1 + 1, nprocs)
  ista = irank * iwork1 + n1 + min(irank,iwork2)
  iend = ista + iwork1 -1
  if(iwork2 .gt. irank) iend = iend + 1
  return
  end subroutine

! jacobi subroutine
      subroutine jacobi(u,im,jm,is,ie,js,je,error)
      integer im,jm,is,ie,js,je,njm
      integer i,j
      real error
      real u(0:im+1,0:jm+1), uo(0:im+1,0:jm+1)

      do j=js-1,je+1
      do i=is-1,ie+1
        uo(i,j) = u(i,j)
      end do
      end do

! jacobi
      do j=js,je
      do i=is,ie
        u(i,j) = (uo(i-1,j)+uo(i+1,j)+uo(i,j-1)+uo(i,j+1))/4.0
      end do
      end do

! error
      error = 0.0
      do j=js,je
      do i=is,ie
        error = error + (u(i,j) - uo(i,j))**2
      end do
      end do
      return
      end subroutine

end program FDM2D
```