Serial Jacobi Iterative Scheme

A single-process implementation of the Jacobi Scheme as applied to the Laplace equation is included below. Note that:

- ***F90 is used.***
- ***System size, m, is determined at run time.***
- ***Boundary conditions are handled by subroutine bc.***
- ***Pointer arrays c, n, e, w, and s point to parts of the solution space, u. They are used to avoid unnecessary memory usage as well as to improve readability.***
- ***This scheme is very slow to converge and is not used in practice.***
- ***This code is shown primarily to provide a starting point for subsequent introduction of parallel concepts. It also serves as a starting point for convergence rate improvement ideas.***

```
! Solve Laplace equation using Jacobi iteration method
! Kadin Tseng, Boston University, November 1999

MODULE jacobi_module
  IMPLICIT NONE
  INTEGER, PARAMETER :: real4 = selected_real_kind(6,37)
  INTEGER, PARAMETER :: real8 = selected_real_kind(15,307)
  REAL(real8), DIMENSION(:,:), ALLOCATABLE         :: unew
  REAL(real8), DIMENSION(:,:), ALLOCATABLE, TARGET :: u  ! solution array
  REAL(real8) :: tol=1.d-4, gdel=1.0d0
  REAL(real4) :: start_time, end_time
  INTEGER :: m, iter = 0
  PUBLIC

CONTAINS
  SUBROUTINE bc(u, m)
! PDE: Laplacian u = 0;      0<=x<=1;  0<=y<=1
! B.C.: u(x,0)=sin(pi*x); u(x,1)=sin(pi*x)*exp(-pi); u(0,y)=u(1,y)=0
! SOLUTION: u(x,y)=sin(pi*x)*exp(-pi*y)
    IMPLICIT NONE
    INTEGER m, j
    REAL(real8), DIMENSION(0:m+1,0:m+1) :: u
    REAL(real8), DIMENSION(:,:), POINTER :: c
    REAL(real8), DIMENSION(0:m+1) :: y0

    y0 = sin(3.141593*(/(j,j=0,m+1)/)/(m+1))

    u = 0.0d0     ! at x=0,1; all y plus initialize interior
    u(:,  0) = y0                ! at y = 0; all x
    u(:,m+1) = y0*exp(-3.141593) ! at y = 1; all x
    RETURN
  END SUBROUTINE bc

END MODULE jacobi_module

PROGRAM Jacobi
USE jacobi_module
REAL(real8), DIMENSION(:,:), POINTER :: c, n, e, w, s
```

```
write(*,*)'Enter matrix size, m:'
read(*,*)m

CALL cpu_time(start_time)    ! start timer, measured in seconds

ALLOCATE ( unew(m,m), u(0:m+1,0:m+1) )  ! mem for unew, u

c => u(1:m  ,1:m  )          ! i  ,j    Current/Central for 1<=i<=m; 1<=j<=m
n => u(1:m  ,2:m+1)          ! i  ,j+1  North (of Current)
e => u(2:m+1,1:m  )          ! i+1,j    East  (of Current)
w => u(0:m-1,1:m  )          ! i-1,j    West  (of Current)
s => u(1:m  ,0:m-1)          ! i  ,j-1  South (of Current)

CALL bc(u, m)                ! set up boundary values

DO WHILE (gdel > tol)        ! iterate until error below threshold
  iter = iter + 1            ! increment iteration counter
  IF(iter > 5000) THEN
    WRITE(*,*)'Iteration terminated (exceeds 5000)'
    STOP                     ! nonconvergent solution
  ENDIF
  unew = ( n + e + w + s )*0.25 ! new solution, Eq. 3
  gdel = MAXVAL(DABS(unew-c))    ! find local max error
  IF(MOD(iter,10)==0) WRITE(*,"('iter,gdel:',i6,e12.4)")iter,gdel
  c = unew                   ! update interior u
ENDDO

CALL CPU_TIME(end_time)      ! stop timer
PRINT *,'Total cpu time =',end_time - start_time,' x 1'
PRINT *,'Stopped at iteration =',iter
PRINT *,'The maximum error =',gdel

DEALLOCATE (unew, u)

END PROGRAM Jacobi
```