Parallel Successive Over Relaxation Red-black Scheme

A parallel implementation of the Successive Over Relaxation Red-black Scheme (based on a Parallel Jacobi Implementation) as applied to the Laplace equation is included below. Note that:

- ***F90 is used.***
- ***System size, m, is determined at run time.***
- ***MPI cartesian topology is used.***
- ***Boundary conditions are handled by subroutine bc.***
- ***The parallel Jacobi code uses subroutine neighbors to provide adjoining process numbers. In this implementation, since cartesian topology is used, MPI_Cart_shift is used to provide the same information.***
- ***A red-black scheme is used to speed in-processor convergence.***
- ***Subroutine update_bc_2 updates the blue cells of current and adjoining processes simultaneously by MPI routine that pairs send and receive, MPI_Sendrecv, for subsequent iteration.***
- ***Subroutine update_bc_1 may be used in place of update_bc_2 as an alternative message passing method***
- ***Subroutine printmesh may be used to print local solution for tiny cases (like 4x4)***
- ***Pointer arrays c, n, e, w, and s point to various parts of the solution space, u. They are used to avoid unnecessary memory usage as well as to improve readability.***
- ***MPI_Allreduce is used to collect global error from all participating processes to determine whether further iteration is required. This is somewhat costly to do in every iteration. Using the fact that the number of iterations is proportional to the grid size, m, and the assumption that we focus on m in the hundredths, MPI_Allreduce is called once every 100 iterations. (In the interests of clarity, we opt to use this simplified criteria). As we said earlier in the parallel Jacobi implementation, there is a small price to pay by calling MPI_Allreduce infrequently. If the solution error threshold is reached inside the 100 iterations, the solution marches on unabated until the 100 count is reached and hence unnecessary computation is performed. However, with slight modifications to the testing criteia, wasteful computing cycles may be minimized. At least for this problem, the savings in MPI_Allreduce calls far outweigh the penalty.***
- ***The effect of MPI_Allreduce call is significantly less noticeable on the SGI Origin2000 shared-memory multiprocessor than on, say, a Linux Pentium cluster due to better communications on the Origin.***
- ***This scheme is considerably more rapid in convergent rate than the Jacobi Scheme.***

```
PROGRAM sor_cart
  USE types_module
  USE sor_module
  TYPE (redblack) :: c, n, e, w, s
  INTEGER, PARAMETER :: period=0, ndim=1
  INTEGER :: grid_comm, me, iv, coord, dims
  LOGICAL, PARAMETER :: reorder = .true.

  CALL MPI_Init(ierr)                              ! starts MPI
  CALL MPI_Comm_rank(MPI_COMM_WORLD, k, ierr)      ! get current process id
  CALL MPI_Comm_size(MPI_COMM_WORLD, p, ierr)      ! get # procs from env or
                                                   ! command line
  if(k == 0) then
    write(*,*)'Enter matrix size, m :'
    read(*,*)m
  endif
  CALL MPI_Bcast(m, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)

  rhoj = 1.0d0 - pi*pi*0.5/(m+2)**2
  rhojsq = rhoj*rhoj
  mp = m/p
  ALLOCATE (vnew(m/2,mp/2), v(0:m+1,0:mp+1))

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

  ! create 1D cartesian topology for matrix
  dims = p
  CALL MPI_Cart_create(MPI_COMM_WORLD, ndim, dims,  &
            period, reorder, grid_comm, ierr)
  CALL MPI_Comm_rank(grid_comm, me, ierr)
  CALL MPI_Cart_coords(grid_comm, me, ndim, coord, ierr)
  iv = coord
  CALL bc(v, m, mp, iv, p)     ! set up boundary conditions

  CALL MPI_Cart_shift(grid_comm, 0, 1, below, above, ierr)

  c = news(v, m, mp, 0, 0)          ! i+0,j+0  center
  n = news(v, m, mp, 0, 1)          ! i+0,j+1  north
  e = news(v, m, mp, 1, 0)          ! i+1,j+0  east
  w = news(v, m, mp,-1, 0)          ! i-1,j+0  west
  s = news(v, m, mp, 0,-1)          ! i+0,j-1  south

  omega = 1.0d0
  CALL update_u(c%red, n%red, e%red, w%red, s%red, &
                vnew, m, mp, omega, delr)   ! update red
  CALL update_bc_2( v, m, mp, iv, below, above)
  omega = 1.0d0/(1.0d0 - 0.50d0*rhojsq)
  CALL update_u(c%black, n%black, e%black, w%black, s%black, &
                vnew, m, mp, omega, delb)   ! update black
  CALL update_bc_2( v, m, mp, iv, below, above)
  DO WHILE (gdel > tol)
    iter = iter + 1    ! increment iteration counter
    omega = 1.0d0/(1.0d0 - 0.25d0*rhojsq*omega)
    CALL update_u(c%red, n%red, e%red, w%red, s%red, &
                vnew, m, mp, omega, delr)   ! update red
    CALL update_bc_2( v, m, mp, iv, below, above)
    omega = 1.0d0/(1.0d0 - 0.25d0*rhojsq*omega)
    CALL update_u(c%black, n%black, e%black, w%black, s%black, &
                vnew, m, mp, omega, delb)   ! update black
    del = (delr + delb)*4.d0
    IF(MOD(iter,100)==0) THEN
      del = (delr + delb)*4.d0
      CALL MPI_Allreduce( del, gdel, 1, MPI_DOUBLE_PRECISION, MPI_MAX,   &
             MPI_COMM_WORLD, ierr )  ! find global max error
```

```
      IF(k == 0) WRITE(*,'(i5,3d13.5)')iter,del,gdel,omega
    ENDIF
  ENDDO

  CALL cpu_time(end_time)          ! stop timer

  IF(k == 0) THEN
    PRINT *,'########################################'
    PRINT *,'Total cpu time =',end_time - start_time,' x',p
    PRINT *,'Stopped at iteration =',iter
    PRINT *,'The maximum error =',del
    write(40,"(3i5)")m,mp,p
  ENDIF
  WRITE(41+k,"(6d13.4)")v
  DEALLOCATE (vnew, v)

  CALL MPI_Finalize(ierr)

END PROGRAM sor_cart
```