Programowanie współbieżne

Laboratorium - 7c: Fortran 95

Andrzej Baran
baran@kft.umcs.lublin.pl

Pochodna 1, MPI I

Fortran 95

- Zaproponuj wersję MPI programu obliczania pierwszej pochodnej z funkcji jednej zmiennej x.

```
!
! Derivative 1, MPI
!
subroutine FirstDeriv1dp(npts, dx, u, u_x, mynode, totalnodes)
  implicit none
  integer, intent(in) :: npts, mynode, totalnodes
  real*8, intent(in) :: dx
  real*8, dimension(:), intent(in) :: u
  real*8, dimension(:), intent(out) :: u_x
  real*8 two_invdx
  MPI_STATUS status
  two_invdx = 1d0/(2d0*dx)

  if(mynode == 0) then
     ! forward...
     u_x(1) = (-3.*u(1)+4.*u(2)-u(3))*two_invdx
  end if

  if(mynode == totalnodes) then
     ! backward
     u_x(npts) = (3.*u(npts)-4.*u(npts-1)+u(npts-2))*two_invdx
  end if
```

Pochodna 1, MPI II

Fortran 95

```
  do i=2,npts-1
     ! central...
     u_x(i) = (u(i+1)-u(i-1))*two_invdx
  end do

  if (mynode == 0) then
     mpitemp = u(npts)
     call MPI_SEND(mpitemp, 1, MPI_DOUBLE, 1, 1, MPI_COM_WORLD)
     call MPI_RECV(mpitemp, 1, MPI_DOUBLE, 1, 1, MPI_COM_WORLD, status)
     u_x(npts) = (mpitemp-u(npts-1))*two_invdx
  else if (mynode == totalnodes) then
     call MPI_RECV(mpitemp, 1, MPI_DOUBLE, mynode, 1, MPI_COM_WORLD, status)
     u_x(1) = (u(2)-mpitemp)*two_invdx
     mpitemp = u(1)
     call MPI_SEND(mpitemp, 1, MPI_DOUBLE, mynode, 1, MPI_COM_WORLD)
  else
     call MPI_RECV(mpitemp, 1, MPI_DOUBLE, mynode, 1, MPI_COM_WORLD, status)
     u_x(1) = (u(2)-mpitemp)*two_invdx
     mpitemp = u(1)
     call MPI_SEND(mpitemp, 1, MPI_DOUBLE, mynode-1, 1, MPI_COM_WORLD)
     mpitemp = u(npts)
     call MPI_SEND(mpitemp, 1, MPI_DOUBLE, mynode+1, 1, MPI_COM_WORLD)
     call MPI_RECV(mpitemp, 1, MPI_DOUBLE, mynode+1, 1, MPI_COM_WORLD, status)
     u_x(npts) = (mpitemp-u(npts-1))*two_invdx
  end if
end subroutine FirstDeriv1dp
```

Pochodna 1, MPI, test I

Fortran 95

- Napisz program testowy obliczania pochodnej z funkcji jednej zmiennej i wykonaj go z maksymalną dostępną liczbą procesów.
  Porównaj wyniki obliczeń równoległych z wynikami otrzymanymi w obliczeniach sekwencyjnych.

```
! przykład użycia procedur SENDRECV i SEND-NDRECV_REPLACE
!
integer mynode, totalnodes, mpierr
integer datasize ! liczba jednostek danych send/recv
integer process1, process2 ! rangi procesów wymieniających dane
integer tag1, tag2 ! znacznik komunikatu
real*8, dimension(:), allocatable :: buff, sendbuff, recvbuff
MPI_status status ! zmienna przechowujaca informacje o stanie

call MPI_INIT( ierr )
call MPI_COMM_SIZE(MPI_COM_WORLD, totalnodes, mpierr)
call MPI_COMM_RANK(MPI_COM_WORLD, mynode, mpierr)

! rezerwacja buforow danych, process1, process2
allocate(sendbuff(datasize))
allocate(recvbuff(datasize))
allocate(buff(datasize))

if (mynode == process1) then
   ! dane w sendbuff są wysylane do process2, odbierane od process2
```

Pochodna 1, MPI, test II

Fortran 95

```
   ! dane trafiają do bufora recvbuff
   call MPI_SENDRECV(sendbuff, datasize, MPI_DOUBLE, process2, &
        tag1, recvbuff, datasize, MPI_DOUBLE, process2, &
        tag2, MPI_COMM_WORLD, status)
   ! wywolanie powoduje wymiane (swap) zawartosci bufora buff
   ! z process2
   call MPI_SENDRECV_REPLACE(buff, datasize, MPI_DOUBLE, process2, &
        tag1, process2, tag2, MPI_COMM_WORLD, status)
end if

if (mynode == process2) then
   ! zawartosc sendbuff jest przekazana do procesu process1, a dane
   ! trzymane od procesu process1 sa umieszczane w buforze recvbuff
   call MPI_SENDRECV(sendbuff, datasize, MPI_DOUBLE, process1, &
        tag2, recvbuff, datasize, MPI_DOUBLE, process1, &
        tag1, MPI_COMM_WORLD, status)
   ! wymiana danych buff z procesem process1
   !
   call MPI_SENDRECV_REPLACE(buff, datasize, MPI_DOUBLE, process1, &
        tag2, process1, tag1, MPI_COMM_WORLD, status)
end if

! process1 ma w swoim buforze recvbuff zawartosc bufora sendbuff
! procesu process1; process2 w swoim buforze recvbuff ma zawartosc
! buforu sendbuff procesu process1; zawartosc buforow buff obu
! procesów zostala zamieniona
```