MPI and Stommel Model
(Continued)

Timothy H. Kaiser, PH.D.
tkaiser@mines.edu

More topics

- Alternate IO routines with Gatherv
- Different types of communication
  - Isend and Irecv
  - Sendrecv
  - Derived types, contiguous and vector
  - Persistent
- Advice
- Topics not covered

Alternate output strategies

- We have been doing a primitive form of parallel output
- May want everything in the same file
  - Same output as serial version
  - Easier to work with
- Our new primitive strategy for serial output is:
  - Bring data back to node 0 a line at a time
  - Print the data from node 0

Simulate serial output - Example

```
subroutine write_grid(psi,i1,i2,j1,j2)
! input is the grid and the indices for the interior cells
    use numz
    use mympi
    use face ,only : unique
    use input
    implicit none
    real(b8),dimension(i1-1:i2+1,j1-1:j2+1):: psi
    integer,intent(in):: i1,i2,j1,j2
    integer i,j,k,i0,j0,i3,j3
    integer istart,iend,jstart,jend,local_bounds(4)
    integer, allocatable :: bounds(:,:),counts(:),offsets(:)
    integer dj,mystart,myend,icol
    real(b8),allocatable :: arow(:)

! the master prints the whole grid a line at a time.
! for a given line, each processor checks to see if
! it holds part of that line and then sends the
! number of cells held using the MPI_GATHER.  the
! MPI_GATHERV is then used to send the data
```

Simulate serial output - Example

```
istart=i1
iend=i2
jstart=j1
jend=j2
if(istart .eq. 1)istart=0
if(jstart .eq. 1)jstart=0
if(iend .eq. nx)iend=nx+1
if(jend .eq. ny)jend=ny+1
i0=0
j0=0
i3=nx+1
j3=ny+1
if(myid .eq. mpi_master)then
    open(18,file=unique("out3d_"),recl=max(80,15*((ny)+3)+2))
    write(18,'(2i6)')i3-i0+1,j3-j0+1
    allocate(arow(j0:j3))
    allocate(counts(0:numnodes-1))
    allocate(offsets(0:numnodes-1))
    offsets(0)=0
endif
```

Simulate serial output - Example

```
do i=i0,i3
   if(i .ge. istart .and. i .le. iend)then
       dj=jend-jstart+1;    mystart=jstart;    myend=jend; icol=i
   else
       dj=0;     mystart=jstart;    myend=jstart;    icol=istart
   endif
   call MPI_GATHER(dj,    1,MPI_INTEGER, counts,1,MPI_INTEGER, &
                   mpi_master,MPI_COMM_WORLD,mpi_err)
   if(myid .eq. mpi_master)then
   do k=1,numnodes-1
          offsets(k)=counts(k-1)+offsets(k-1)
      enddo
   endif
   call MPI_GATHERV(psi(icol,mystart:myend),dj,MPI_DOUBLE_PRECISION,
&
                    arow(0),counts,offsets,MPI_DOUBLE_PRECISION, &
                    mpi_master,MPI_COMM_WORLD,mpi_err)
   if(myid .eq. mpi_master)then
      do j=j0,j3
         write(18,'(g14.7)',advance="no")arow(j)
         if(j .ne. j3)write(18,'(" ")',advance="no")
      enddo
      write(18,*)
   endif
enddo
```

Asynchronous - Nonblocking

- Recall we might want to use nonblocking communications to overlap work and communications
- We don't do that here
- MPI_Isend, MPI_Irecv

Isend and Irecv (Fortran) - Example

- Allocate vectors sv1, sv2, sv3, sv4, rv1, rv2, rv3, rv4
- We copy the sending data to sv1-sv4
- Call Isend and Irecv collecting requests in a vector
- Call MPI_Waitall
- Copy data from rv1-rv4 to matrix

st_04

Isend and Irecv (Fortran) - Example

- Why the copy?
  - We were using F90 array syntax to pass noncontiguous blocks of data
  - F90 does NOT pass by reference in this case
  - Does a copy-in copy-out and it copy-outs the wrong data.
  - Challenge: Which of the copies are not required?

C Isend and Irecv - Example st_04

- We already did a copy for the noncontiguous data so a new one is not required
- We copy the sending data to sv1 and sv2
- Call Isend and Irecv collecting requests in a vector
- Call MPI_Waitall
- Copy data from rv1 and rv2 to matrix

MPI_Sendrecv - Example st_05

```
! send to left
  call MPI_SEND(psi(:,j1),  num_x,MPI_DOUBLE_PRECISION,myleft,  &
                            100,ROW_COMM,mpi_err)
! rec from left
  call MPI_RECV(psi(:,j1-1),num_x,MPI_DOUBLE_PRECISION,myleft,  &
                            100,ROW_COMM,status,mpi_err)
! rec from right
  call MPI_RECV(psi(:,j2+1),num_x,MPI_DOUBLE_PRECISION,myright, &
                            100,ROW_COMM,status,mpi_err)
! send to right
  call MPI_SEND(psi(:,j2),  num_x,MPI_DOUBLE_PRECISION,myright, &
                            100,ROW_COMM,mpi_err)
```

Becomes…

```
call MPI_Sendrecv(psi(:,j1),  num_x,MPI_DOUBLE_PRECISION, myleft,100,&
                  psi(:,j1-1),num_x,MPI_DOUBLE_PRECISION, myleft,100,&
                                    ROW_COMM,status, mpi_err)
call MPI_Sendrecv(psi(:,j2),  num_x,MPI_DOUBLE_PRECISION, myright,100,&
                  psi(:,j2+1),num_x,MPI_DOUBLE_PRECISION, myright,100,&
                                    ROW_COMM,status, mpi_err)
```

With similar changes for the other communications

st_05

Using derived types

- A row or column of ghost cells are lumped into a derived data type
- We do a send/receive with count =1 in our MPI calls sending a single element of our type
- Eliminates all explicit copies
- Works with Isend and Irecv
- Definition of the data types are reversed for Fortran and C

Contiguous and Vector data types - Example st_06

Fortran-derived data type definition

```
subroutine do_types(psi,i1,i2,j1,j2)
  use numz
  use mympi
  use input
  implicit none
  real(b8),dimension(i1-1:i2+1,j1-1:j2+1):: psi
  integer,intent(in):: i1,i2,j1,j2
  integer num_x,num_y,stride
  num_x=i2-i1+3
  call MPI_TYPE_CONTIGUOUS(num_x,MPI_DOUBLE_PRECISION, &
                MPI_LEFT_RITE,mpi_err)
  call MPI_TYPE_COMMIT(MPI_LEFT_RITE,mpi_err)
  num_y=j2-j1+3
  stride=num_x
  call MPI_TYPE_VECTOR(num_y,1,stride,MPI_DOUBLE_PRECISION, &
                MPI_TOP_BOT,mpi_err)
  call MPI_TYPE_COMMIT(MPI_TOP_BOT,mpi_err)
end subroutine do_types
```

Fortran-derived data type communication routines

```
num_x=1
num_y=1
ireq=0
if(myleft .ne. MPI_PROC_NULL)then
! send to left
  ireq=ireq+1
  call MPI_ISEND(psi(i1-1,j1),num_x,MPI_LEFT_RITE,myleft, &
                 100,ROW_COMM,req(ireq),mpi_err)
…
…
…
 if(mybot .ne. MPI_PROC_NULL)then
! rec from bot
  ireq=ireq+1
  call MPI_IRECV(psi(i2+1,j1-1),num_y,MPI_TOP_BOT,mybot, &
                 10,COL_COMM,req(ireq),mpi_err)
```

Note that we pass starting element of our arrays

st_06

C-derived data type definition

```
void do_types(INT i1,INT i2,INT j1,INT j2){
  INT num_x,num_y,stride;

  num_y=j2-j1+3;
  mpi_err=MPI_Type_contiguous(num_y,MPI_DOUBLE,
                &MPI_TOP_BOT);
  mpi_err=MPI_Type_commit(&MPI_TOP_BOT);


  num_x=i2-i1+3;
  stride=num_y;
  mpi_err=MPI_Type_vector(num_x,1,stride,MPI_DOUBLE,
                &MPI_LEFT_RITE);
  mpi_err=MPI_Type_commit(&MPI_LEFT_RITE);
}
```

C-derived data type communication routines

```
num_x=1;
num_y=1;
ireq=0;
if((myid_col % 2) == 0){
/* send to left */
  mpi_err=MPI_Isend(&psi[i1-1][j1],num_x,MPI_LEFT_RITE,
                    myleft,100,ROW_COMM,&req[ireq]);ireq++;
...
...
...
/* send to top */
  mpi_err=MPI_Irecv(&psi[i2+1][j1-1],num_y,MPI_TOP_BOT,
                      mybot,10,COL_COMM,&req[ireq]);ireq++;
```

Note that we pass address of starting element of our arrays

Persistent communication - Example

- Persistent communication is used when you have an often-repeated pattern of communication
- We add a routine do_persistent
  - Called once before we start the main loop
  - Sets up our pattern of communication
  - Calls the routines:

    MPI_Recv_init

    MPI_Send_init

  - Saves communication requests in an array req

st_07

Persistent communication - Example

- The communication routine is simple

  call MPI_STARTALL

  call MPI_WAITALL

  and copies

- I chose to do copies of data to/from vectors
  - Some of the copies are required, some not
  - Challenge: Which are required and why?

From the routine do_persistent...

```
num_x=i2-i1+3
num_y=j2-j1+3
ireq=0
! send to left
if(myleft .ne. MPI_PROC_NULL)then
  ireq=ireq+1
  allocate(sv3(i1-1:i2+1))
! sv3=psi(:,j1)
  call MPI_SEND_INIT(sv3,num_x,MPI_DOUBLE_PRECISION,myleft,&
                     100,ROW_COMM,req(ireq),mpi_err)
! rec from left
  ireq=ireq+1
  allocate(rv3(i1-1:i2+1))
! rv3=psi(:,j1-1)
  call MPI_RECV_INIT(rv3,num_x,MPI_DOUBLE_PRECISION,myleft,&
                     100,ROW_COMM,req(ireq),mpi_err)
endif
```

Similar calls for the other 3 directions

From the routine do_persistent...

```
void do_presistent(FLT **psi,INT i1,INT i2,INT j1,INT j2) {
 INT num_x,num_y;
 num_x=i2-i1+3; num_y=j2-j1+3;
 ireq=0;
 sv1=NULL; sv2=NULL; sv3=NULL; sv4=NULL;
 rv1=NULL; rv2=NULL; rv3=NULL; rv4=NULL;
/* send to left */
 if(myleft != MPI_PROC_NULL){
/* sv3=psi(:,j1)  */
  sv3=vector(i1-1,i2+1);
  mpi_err=MPI_Send_init(&sv3[i1-1],num_x,MPI_DOUBLE,
                 myleft,100,ROW_COMM,&req[ireq]);ireq++;
/* rec from left */
/* rv3=psi(:,j1-1) */
  rv3=vector(i1-1,i2+1);
  mpi_err=MPI_Recv_init(&rv3[i1-1],num_x,MPI_DOUBLE,
                 myleft,100,ROW_COMM,&req[ireq]);ireq++;
    }
```

Similar calls for the other 3 directions

Persistent communication - Example

```
subroutine do_transfer(psi,i1,i2,j1,j2)
 use numz
 use mympi
 use input
 implicit none
 real(b8),dimension(i1-1:i2+1,j1-1:j2+1):: psi
 integer,intent(in):: i1,i2,j1,j2
 if(allocated(sv1))sv1=psi(i1,:)
 if(allocated(sv2))sv2=psi(i2,:)
 if(allocated(sv3))sv3=psi(:,j1)
 if(allocated(sv4))sv4=psi(:,j2)
    call MPI_STARTALL(ireq,req,mpi_err)
    call MPI_WAITALL(ireq,req,stat_ray,mpi_err)
 if(allocated(rv1))psi(i1-1,:)=rv1
 if(allocated(rv2))psi(i2+1,:)=rv2
 if(allocated(rv3))psi(:,j1-1)=rv3
 if(allocated(rv4))psi(:,j2+1)=rv4
end subroutine do_transfer
```

C version is the same except for explicit for loops

Thoughts from...

NPACI Parallel Computing Institute
June 21-25, 1999
San Diego Supercomputing Center

MPI Tips

- One of the processors having less load is better than one of the processors having more load (5,5,5,3 vs 4,4,4,6)
- Overlapping computation and communication
- Minimize the number of messages by packing data and/or using derived data types

MPI Tips

- Use asynchronous unbuffered send-receives if possible
- Avoid global barriers
- Minimize the global communications by creating sub-processor sets
- Use send_init, etc., to make repeated send-receives efficient

MPI Tips

- For performing complex communications, use MPI-provided calls such as MPI_Alltoallv rather than building them from send-receives
- Use MPI-2 features such as one- sided communication and parallel I/O

MPI Tips

- Obviously parallel algorithm should be designed to be independent of the number of processors
- It's good practice to block the parallel code and wrap them in #ifdef directives

Topics not covered

- Derived data types with absolute addresses
- Packing
- Scan
- User-defined operations with reduce and scan
- Intercommunications
- Process topologies functions
- Buffered communication