LAB
Hybrid Computing

What you will see in this code

- Using MPI and OpenMP Together
  - Threads on Nodes and MPI Communication between nodes
  - MPI calls from serial region
  - MPI calls from master thread in a parallel region
  - MPI calls from all threads in a parallel region
- Examples of Matrix
  - On node Matrix-Matrix multiply, Fox method submatrix-submatrix multiplies (using non-smp dgemm).
  - On node Matrix-Matrix multiply with SMP dgemm.

Hybrid Computing -- Ranger

- Untar the file numahybrid.tar
  - cd (change to HOME dir.)
  - tar xvf ~train00/numahybrid.tar (extract files)
  - cd numahybrid (change directory to numahybrid)
  - cd threadedmpi (change directory to hybrid)

- The makefile is setup to use the Intel compiler
  - module del mvapich2
  - module swap pgi intel
  - module load mvapich2
  - “make” to build an executable

MPI/OpenMP – Ranger (job.tmpi)

```
#!/bin/tcsh
#                          # use bash shell
#$ -V                      # inherit submission environment
#$ -cwd                    # use submission directory
#$ -N threadedmpi          # jobname (threadedmpi)
#$ -j y                    # stdout/err combined
#$ -o $JOB_NAME.o$JOB_ID   # output name jobname.ojobid
#$ -pe 1way 32             # 1 task/node, 32 cores total
#$ -q development          # queue name, can use development
#$ -l h_rt=00:10:00        # request 10 minutes
#$ -A A-tr3
## -M your_email_address   # Mail address !! your own mail
## -m be                   # send email at begin/end of job}
```

If # of task does not equal multiple of 16, set value here.

```
setenv MY_NSLOTS 2
setenv OMP_NUM_THREADS 16
ibrun ./tmpi < input
```

This will give you 10 minutes of 2 nodes (=32/16) 1 task per node (1-way) and a total of 2 tasks, in the development queue.

MPI/OpenMP – Ranger

- Can use the “build.sh” script to build versions of the hybrid code with different MPI stacks
- Can then use “job.stacks” to run two different MPI flavors in the same job script

Don’t worry about this warning:/opt/apps/intel/10.1/fc//lib/libimf.so: warning: warning: feupdateenv is not implemented and will always fail tmpi ready

MPI/OpenMP – Ranger

```
include "mpif.h"
…
call MPI_Init(ierr)
call MPI_Comm_size(MPI_COMM_WORLD,nranks, ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,irank,ierr)

if(irank == 0) then
   call mpi_send(as,n,MPI_REAL8, 1,9,MPI_COMM_WORLD, ierr)
   call mpi_recv(as,n,MPI_REAL8, 1,1,MPI_COMM_WORLD, istatus,ierr)
else if (irank == 1) then
   call mpi_recv(as,n,MPI_REAL8, 0,9,MPI_COMM_WORLD, istatus,ierr)
   call mpi_send(as,n,MPI_REAL8, 0,1,MPI_COMM_WORLD, ierr)
endif



if(irank .eq. 0) read(*,'(i5)') iread1
call MPI_Bcast(iread1,1,MPI_INTEGER, 0,iwcomm, ierr)
```

"Serial Code"

(and error argument in f90 codes)

MPI/OpenMP – Ranger

```
!$OMP PARALLEL private(i,ithread,nthreads, icp1, icp2, icpd)


ithread =OMP_GET_THREAD_NUM()

if(ithread == 0) then
   if(irank .eq. 0) read(*,'(i5)') iread2
   call MPI_Bcast(iread2,1,MPI_INTEGER, 0,iwcomm, ierr)
end if
```

Parallel Region

⋮

(and error argument in f90 codes)

MPI/OpenMP – Ranger

```
!$OMP DO ordered
  do i = 1,nthreads
!$OMP ordered
  if(irank == 0) then
     call mpi_send(as,ns,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, ierr)
     call mpi_recv(as,ns,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, istatus,ierr)
  else if (irank == 1) then
     call mpi_recv(as,ns,MPI_REAL8, 0,ithread,MPI_COMM_WORLD, istatus,ierr)
     call mpi_send(as,ns,MPI_REAL8, 0,ithread,MPI_COMM_WORLD,ierr)
   endif
!$OMP end ordered
  end do
```

Parallel Region

Not needed

```
if(irank == 0 .and. ithread == 0) then
     call mpi_send(as,n,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, ierr)
     call mpi_recv(ar,n,MPI_REAL8, 1,ithread,MPI_COMM_WORLD, istatus,ier
else if (irank == 1 .and. ithread == 0) then
     call mpi_recv(ar,n,MPI_REAL8, 0,ithread,MPI_COMM_WORLD, istatus,ier
     call mpi_send(as,n,MPI_REAL8, 0,ithread,MPI_COMM_WORLD, ierr)
endif
!$OMP barrier

!$OMP END PARALLEL
call mpi_finalize(ierr)
```

⋮

End of Parallel

End of MPI

(and error argument in f90 codes)

MPI/OpenMP – Ranger

RESULTS (see next page):

```
1.) Serial   size:secs   400000 total:     0.00720
2.) rank  0              has 111 from Broadcast
    rank  1              has 111 from Broadcast
3.) rank  0 of thread  0 has 222 from Broadcast
      rank  1 of thread  0 has 222 from  Broadcast
4.) One call size:secs   400000 total:     0.00758
5.) 16 calls size:secs    25000 total:     0.00775

    values  0.00104  0.00045  0.00044  0.00044  0.00044
            0.00044  0.00044  0.00044  0.00044  0.00044
            0.00044  0.00044  0.00044  0.00047  0.00048
            0.00044
```

1.) **Time to send and receive back 400,000 DP words. (First MPI Send/Rec may be expensive!!!)**
2.) **(Broadcast in Serial region, value check)**
3.) **(Broadcast in parallel region, value check)**
4.) **Time to send and receive back 400,000 DP words in parallel region.**
5.) **Time to send and receive back 25,000 DB words from 16 threads, in serialized "ordered" loop. (Total time, and individual times shown.)**

MPI/OpenMP – Ranger

Experiments:

1.) **Make a copy of the first send/receive pair (without the timer), and place it before the the timed pair, so that you are not recording the cost of the first send/receive in the timed pair. (Does the time change significantly, and is it comparable to the parallel region send/receive now?)**

2.) **Remove the Ordered Do from the parallel region, and allow all the threads to communicated at once. (This is allowed for MVAPICH2– only.) Does the cost change?**

Remove:

```
!$OMP DO ordered
do i = 1,nthreads
!$OMP ordered
```

and

```
!$OMP end ordered
end do
```

Also, put a single timer outside of the barriers (but also take into account the cost of the barrier).

3.) **Try using other MPI "flavors": mvapich1 and mvapich-devel, etc. Does the time change?**

a.) remove the tmpi.o and tmpi files. { make clean }

b.) exchange MPI

c.) remake

d.) submit job

```
make clean
module swap mvapich2 mvapich1
make
qsub job.tmpi
```

→

```
make clean
module swap mvapich1 mvapich-devel
make
qsub job.tmpi
```

MPI/OpenMP – Ranger

Notes:

1.) **The f90cp function (when inlined) has an accuracy of 1 Clock Period (CP) ~1/(2.0E+9)**
2.) **The accuracy is assured only if the differences between access of the counter is 80 CPs.**