Introduction to MPI

Masao Fujinaga
Academic Information and Communication Technology
University of Alberta

Connect to the linux cluster

- ssh -Y **ccid**@cluster.srv.ualberta.ca
- "-Y" : enables trusted X11 forwarding
- cd /scratch/**ccid**/
- cp -r /scratch/fujinaga/nov2010-mpi .
- cd nov2010-mpi
- ggv nov2010-mpi.pdf &

Message Passing

- Parallel computation occurs through a number of processes, each with its own local data
- Sharing of data is achieved by message passing. i.e. by explicitly sending and receiving data between processes

What is MPI?

- MPI
  – Specified by a committee of experts from research and industry
  – Standard message-passing specification for all the Massively Parallel Processor (MPP) vendors involved

A simple MPI program

- Fortran

```
INCLUDE 'mpif.h'
INTEGER error, rank,len
character*255 hostname
CALL MPI_Init(error)
CALL MPI_Comm_rank(MPI_COMM_WORLD, rank, error)
CALL MPI_Get_processor_name(hostname,len,error)
PRINT *, "Hello world from ",rank,hostname
CALL MPI_Finalize(error)
STOP
END
```

- C

```
#include <stdio.h>
#include <mpi.h>
void main (int argc, char *argv[]) {
  int rank,len;
  char name[255];
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Get_processor_name(name,&len);
  printf("Hello world from %d %s\n", rank,name);
  MPI_Finalize();
}
```

Different implementations of MPI

- Default
  - MPICH2 over gigabit ethernet
  - Portland compilers
- OpenMPI
  - Infiniband
  - gnu compilers
- MVAPICH2
  - Infiniband
  - gnu compilers
- Intel MPI
  - Infiniband
  - gnu compiler

OpenMPI

- module load mpi/openmpi-1.2.5
- which mpif77

```
/usr/local/openmpi-1.2.5/bin/mpif77
```

- mpif77 -o hello hello.f -show

```
gfortran -I/usr/local/openmpi-1.2.5/include -pthread -o hello hello.f -L/usr/local/
    openmpi-1.2.5/lib -lmpi_f77 -lmpi -lopen-rte -lopen-pal -ldl -Wl,--export-
    dynamic -lnsl -lutil -lm -ldl
```

- Change compiler

  C: OMPI_CC
  C++: OMPI_CXX
  Fortran 77: OMPI_F77
  Fortran 90: OMPI_FC

- setenv OMPI_F77 pgf77
- mpif77 -o hello hello.f -show

```
pgf77 -I/usr/local/openmpi-1.2.5/include -pthread -o hello hello.f -L/usr/local/
    openmpi-1.2.5/lib -lmpi_f77 -lmpi -lopen-rte -lopen-pal -ldl -Wl,--export-
    dynamic -lnsl -lutil -lm -ldl
```

Running an MPI program

- module load mpi/openmpi-1.2.5
- mpif77 -o hello hello.f
  - mpicc -o hello hello.c
- mpiexec -np 4 ./hello

```
Hello world from  0 cluster-login.nic.ualberta.ca
Hello world from  1 cluster-login.nic.ualberta.ca
Hello world from  2 cluster-login.nic.ualberta.ca
Hello world from  3 cluster-login.nic.ualberta.ca
```

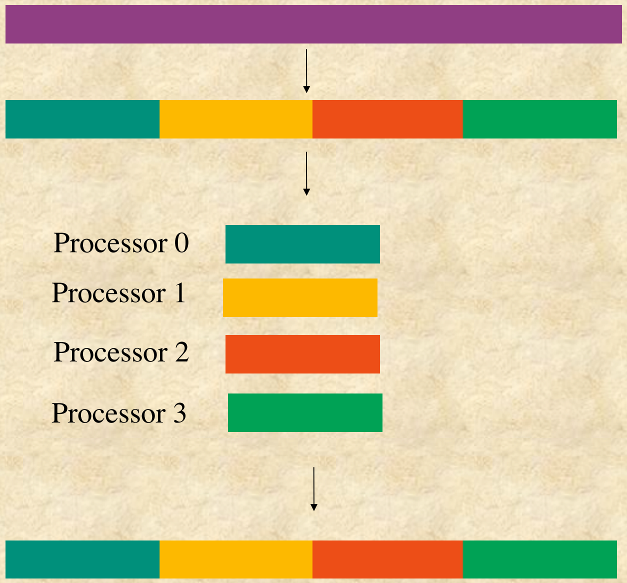
Running MPI in batch

```
#!/bin/bash  -l
#PBS  -S  /bin/bash
#PBS  -l  nodes=2:ppn=2
#PBS  -l  walltime=01:00:00
module  load  mpi/openmpi-1.2.5
cd  $PBS_O_WORKDIR
mpiexec  ./hello > out
```

```
qsub  script.pbs
qstat  -u  ccid
cat  out
```

Serial program

```
do i = 1, n
    y(i) = x(i)**2.3
enddo
```



Master-slave program

```
call MPI_Comm_rank(MPI_COMM_WORLD, rank, error)
call MPI_Comm_size(MPI_COMM_WORLD, size, error)

if(rank .eq. 0)then
   - master code
        send data to slaves
        calculate its share of results
        receive results from slaves
else
   - slave code
        receive data from master
        calculate results
        send results to master
endif
```

MPI_Send/MPI_Recv

- MPI_Send

MPI_Send(buf, count, type, dest, tag, comm, ierr)

- MPI_Recv

MPI_Recv(buf, count, type, source, tag, comm, **status**, ierr)

Wildcards

MPI_ANY_SOURCE, MPI_ANY_TAG

Fortran

status(MPI_SOURCE)

status(MPI_TAG)

status(MPI_ERROR)

C

status.MPI_SOURCE

status.MPI_TAG

status.MPI_ERROR

MPI Data Type	Fortran Data Type
MPI_INTEGER	integer
MPI_REAL	real
MPI_DOUBLE_PRECISION	double precision
MPI_COMPLEX	complex
MPI_CHARACTER	character(1)
MPI_LOGICAL	logical
MPI_BYTE	(none)
MPI_PACKED	(none)

MPI Data Type	C Data Type
MPI_INT	int
MPI_LONG	long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long

```
integer status(MPI_STATUS_SIZE)
call MPI_Comm_rank(MPI_COMM_WORLD, rank, error)
call MPI_Comm_size(MPI_COMM_WORLD, size, error)
npart = nmax/size
if(rank .eq. 0)then
   do iproc = 1, size-1
       index = iproc*npart+1
       call MPI_Send(x(index), npart, MPI_REAL, iproc, 1, MPI_COMM_WORLD,
   error)
   enddo
   do 210 i = 1, npart
       y(i) = x(i)**2.3
   enddo
    do iproc = 1, size-1
       index = iproc*npart+1
       call MPI_Recv(y(index), npart, MPI_REAL, iproc, 2, MPI_COMM_WORLD,
   status,error)
   enddo
else
   call MPI_Recv(x, npart, MPI_REAL, 0, 1, MPI_COMM_WORLD, status, error)
   do i = 1, npart
       y(i) = x(i)**2.3
   enddo
   call MPI_Send(y, npart, MPI_REAL, 0, 2,  MPI_COMM_WORLD, error)
endif
```

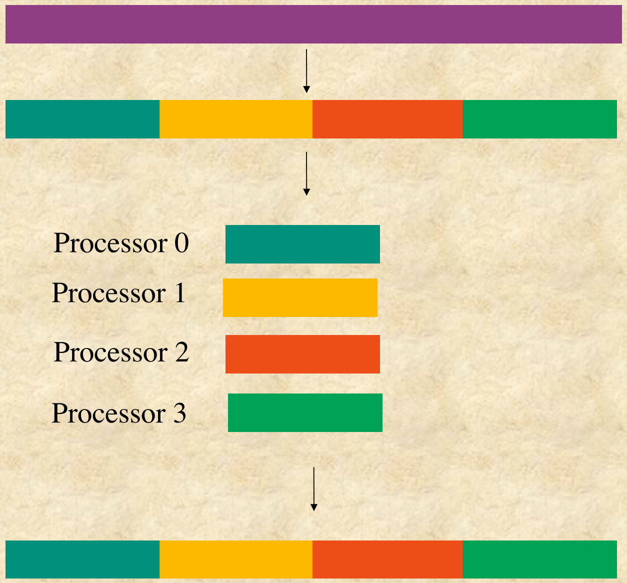
Basic commands

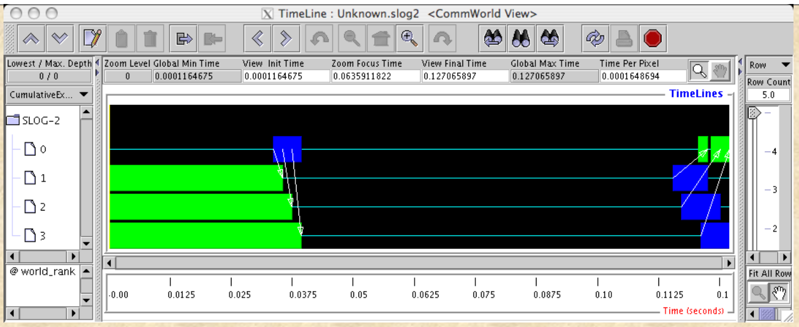
- Include file
- MPI_Init
- MPI_Comm_rank
- MPI_Comm_size
- MPI_Send
- MPI_Recv
- MPI_Finalize

- To see list of routines, do
  man -k MPI
- To see details,
  man MPI_Send

A simpler way

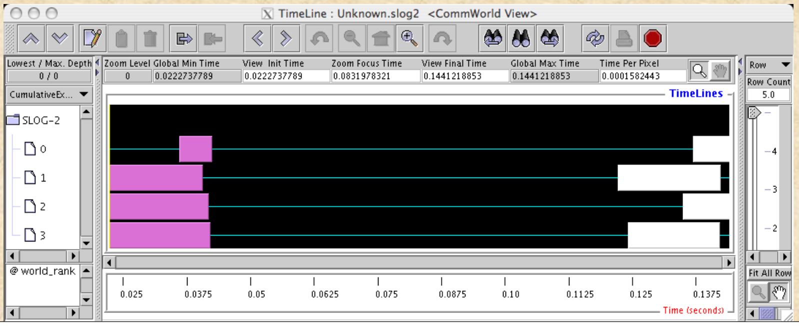
```
call MPI_Scatter(x, npart, MPI_REAL, x, npart, MPI_REAL,
    0, MPI_COMM_WORLD, error)
do i = 1, npart
    y(i) = x(i)**2.3
enddo
call MPI_Gather(y, npart, MPI_REAL, y, npart, MPI_REAL,
    0, MPI_COMM_WORLD, error)
```




TimeLine : Unknown.slog2 <CommWorld View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel
0 / 0	0	0.0001164675	0.0001164675	0.0635911822	0.127065897	0.127065897	0.0001648694

Row Count 5.0

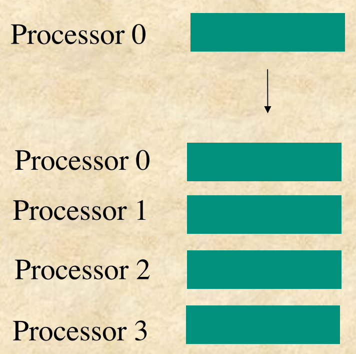

TimeLine : Unknown.slog2 <CommWorld View>

Lowest / Max. Depth	Zoom Level	Global Min Time	View Init Time	Zoom Focus Time	View Final Time	Global Max Time	Time Per Pixel
0 / 0	0	0.0222737789	0.0222737789	0.0831978321	0.1441218853	0.1441218853	0.0001582443

Row Count 5.0

Broadcast

MPI_Bcast(buffer, count, type, rank, comm, ierr)



Exercise 1

- Modify exercise1.f or exercise1.c by adding MPI_Scatter, MPI_Bcast and MPI_Gather
- The program will read from standard input, a1,a2,a3.
- For an array, x, it will calculate

  y=a1*x*x+a2*x+a3
- Writes to standard output, x and y
- Do ‘man MPI_Scatter’ etc. to get parameter list.

Reduction

```
sum = 0
do i = 1, nmax
  sum = sum + x(i)
enddo
```

↓

```
sum = 0
subsum = 0
do i = 1, npart
  subsum = subsum + x(i)
enddo
call MPI_Reduce(subsum, sum, 1, MPI_INTEGER,
    MPI_SUM, 0, MPI_COMM_WORLD, ierr)
```

Exercise 2

- Modify exercise2.f or exercise2.c to do the sum in parallel
- Use MPI_Scatter and MPI_Reduce

Performance

- For best performance, minimize communication.
  - Minimize the amount of data transferred and the number of calls to message passing routines.
- Next best thing: Minimize communication time relative to computation time.
- Or overlap communication with calculation
- Avoid synchronization steps.
- Make sure that all processes are busy (load balancing)

Benchmarking

- For any parallel program, it is important to know the parallel efficiency of the program
- time the real time it takes to run the program using various number of processors

Exercise 3

- compile sample1long.f (or .c)

```
mpif77 -o sample1long sample1long.f
```

- submit script3.pbs. This will run the program four times, changing -n to 1, 2, 4, and 8
- look at the real time (‘grep real out.time’)
- plot using www.wolframalpha.com
- plot time

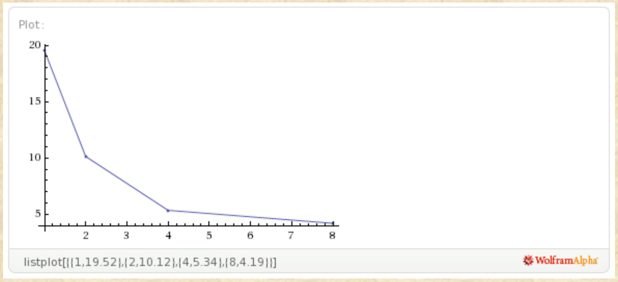
```
listplot[{{1,t1},{2,t2},{4,t4},{8,t8}}]
```

- plot speedup

```
listplot[{{1,1},{2,t1/t2},{4,t1/t4},{8,t1/t8}}]
```

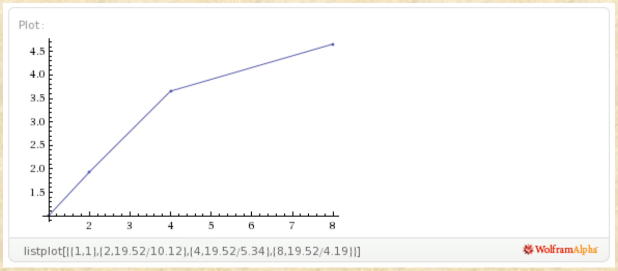
Benchmark results

Time



```
listplot[{{1,19.52},{2,10.12},{4,5.34},{8,4.19}}]
```

Speedup



```
listplot[{{1,1},{2,19.52/10.12},{4,19.52/5.34},{8,19.52/4.19}}]
```

Performance analysis with OpenMPI/MPE

- Set up environment

```
module load mpi/openmpi-1.2.5
```

- Compile program

```
mpefc -o sample1 sample1.f -mpilog
mpecc -o sample1 sample1.c -mpilog
```

- Run program

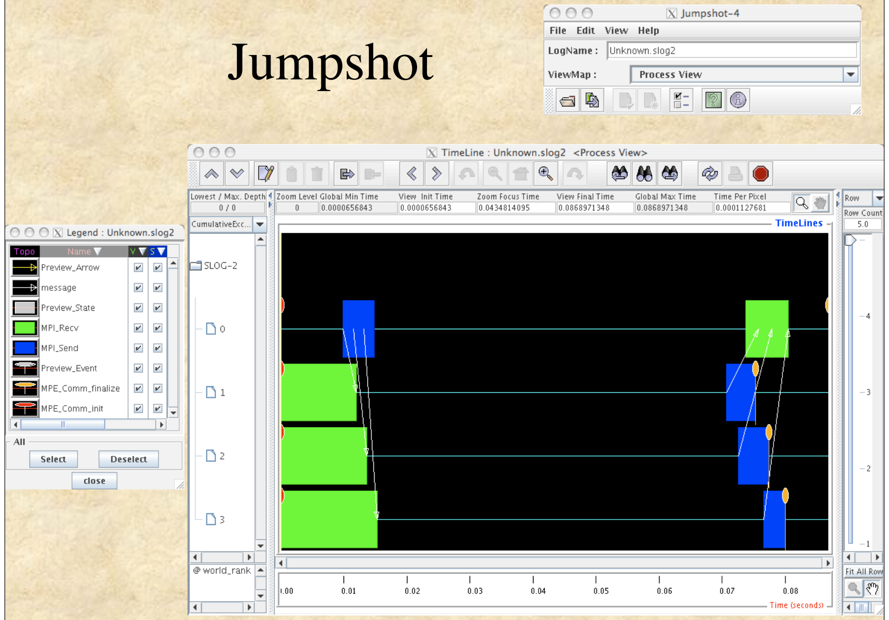
```
mpiexec -np 4 ./sample1
```

- View results

```
clog2TOslog2 Unknown.clog2
jumpshot Unknown.slog2
```

(sample1.clog2 and sample.slog2 for c)

Jumpshot



MPI_Wtime()

- Returns elapsed (wall) time on the calling processor
  - Time in seconds since an arbitrary time in the past

```
real*8 time
time = MPI_Wtime()
Calculate…
write(*,*)' elapsed time =',MPI_Wtime()-time
---------------------------------------------------------------
double time;
time = MPI_Wtime();
Calculate ...
printf("elaspsed time = %f\n",MPI_Wtime()-time);
```

Blocking and Completion

- MPI_Send and MPI_Recv block the calling process. i.e. they do not return until the communication operation is complete
- MPI_Recv is complete when the message is copied to the output variable.
- MPI_Send is complete when the message has been passed off to MPI

Deadlock

- When two or more blocked processes are waiting for each other and cannot make progress.

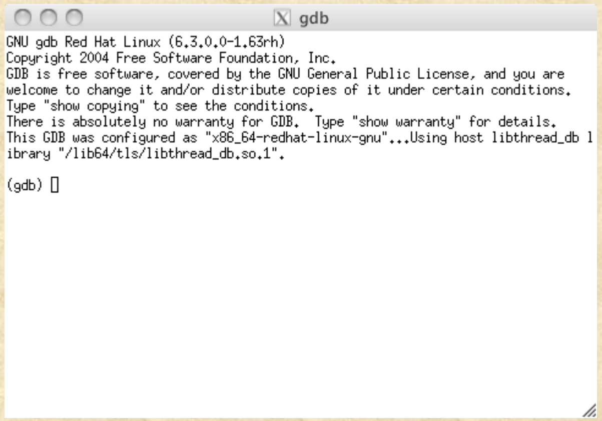
```
if( rank .eq. 0)then
    call MPI_Recv(x, nmax, MPI_REAL, 1, 1, MPI_COMM_WORLD, status, ierr)
    call MPI_Send(y, nmax, MPI_REAL, 1, 2, MPI_COMM_WORLD, ierr)
else if ( rank .eq. 1)then
    call MPI_Recv(y, nmax, MPI_REAL, 0, 2, MPI_COMM_WORLD, status, ierr)
    call MPI_Send(x, nmax, MPI_REAL, 0, 1, MPI_COMM_WORLD, ierr)
endif
```

Debugging deadlock - 1

- mpif77 -o deadlock0 deadlock0.f -g
- mpiexec -n 2 ./deadlock0
- ctrl-c to exit program

Debugging deadlock - 2

- mpiexec -n 2 xterm -e gdb ./deadlock0

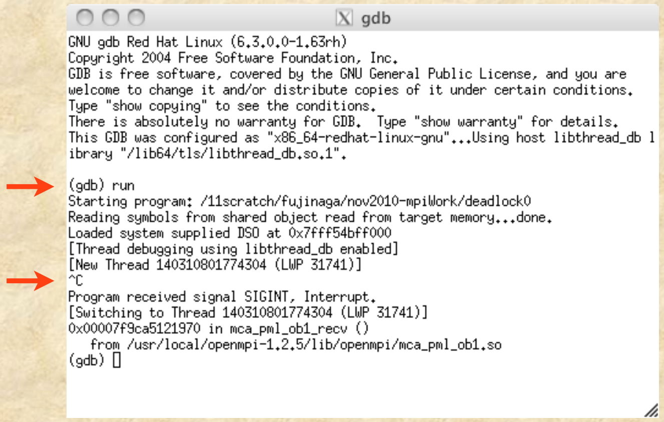


```
GNU gdb Red Hat Linux (6.3.0.0-1.63rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu"...Using host libthread_db l
ibrary "/lib64/tls/libthread_db.so.1".

(gdb) 
```

Debugging deadlock - 3

- In each window:
  - type 'run' to start program
  - ctrl-c to halt program

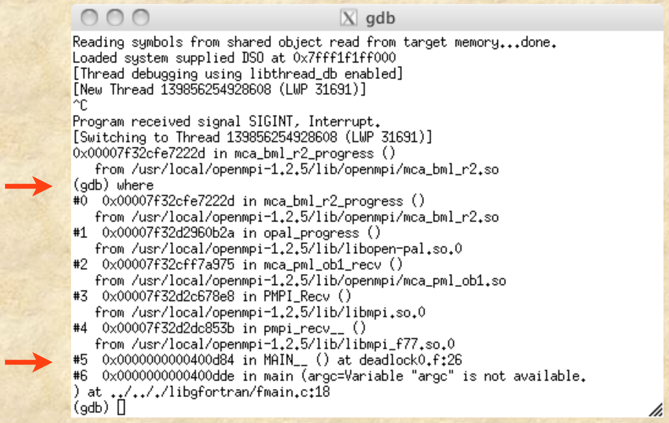


```
GNU gdb Red Hat Linux (6.3.0.0-1.63rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu"...Using host libthread_db l
ibrary "/lib64/tls/libthread_db.so.1".

(gdb) run
Starting program: /11scratch/fujinaga/nov2010-mpiWork/deadlock0
Reading symbols from shared object read from target memory...done.
Loaded system supplied DSO at 0x7fff54bff000
[Thread debugging using libthread_db enabled]
[New Thread 140310801774304 (LWP 31741)]
^C
Program received signal SIGINT, Interrupt.
[Switching to Thread 140310801774304 (LWP 31741)]
0x00007f9ca5121970 in mca_pml_ob1_recv ()
   from /usr/local/openmpi-1.2.5/lib/openmpi/mca_pml_ob1.so
(gdb) 
```

Debugging deadlock - 4

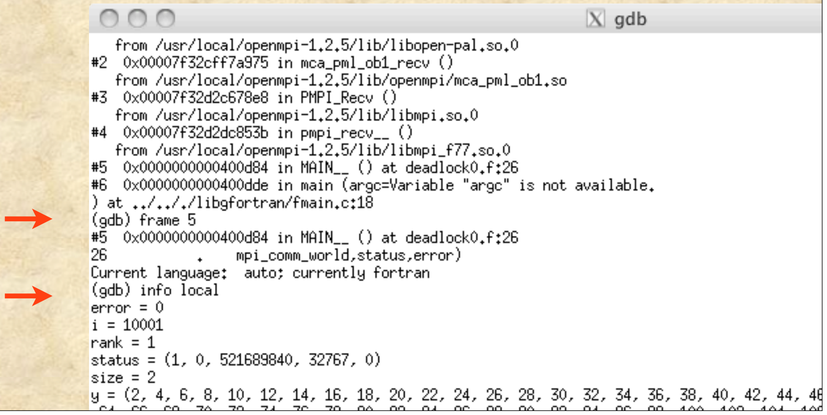
- type ‘where’



```
Reading symbols from shared object read from target memory...done.
Loaded system supplied DSO at 0x7fff1f1ff000
[Thread debugging using libthread_db enabled]
[New Thread 139856254928608 (LWP 31691)]
^C
Program received signal SIGINT, Interrupt.
[Switching to Thread 139856254928608 (LWP 31691)]
0x00007f32cfe7222d in mca_bml_r2_progress ()
   from /usr/local/openmpi-1.2.5/lib/openmpi/mca_bml_r2.so
(gdb) where
#0  0x00007f32cfe7222d in mca_bml_r2_progress ()
   from /usr/local/openmpi-1.2.5/lib/openmpi/mca_bml_r2.so
#1  0x00007f32d2960b2a in opal_progress ()
   from /usr/local/openmpi-1.2.5/lib/libopen-pal.so.0
#2  0x00007f32cff7a975 in mca_pml_ob1_recv ()
   from /usr/local/openmpi-1.2.5/lib/openmpi/mca_pml_ob1.so
#3  0x00007f32d2c678e8 in PMPI_Recv ()
   from /usr/local/openmpi-1.2.5/lib/libmpi.so.0
#4  0x00007f32d2dc853b in pmpi_recv__ ()
   from /usr/local/openmpi-1.2.5/lib/libmpi_f77.so.0
#5  0x0000000000400d84 in MAIN__ () at deadlock0.f:26
#6  0x0000000000400dde in main (argc=Variable "argc" is not available.
) at ../../../libgfortran/fmain.c:18
(gdb) 
```

Debugging deadlock - 5

- type ‘frame 5’ (or whichever number corresponds to the user code)
- type ‘info local’



```
   from /usr/local/openmpi-1.2.5/lib/libopen-pal.so.0
#2  0x00007f32cff7a975 in mca_pml_ob1_recv ()
   from /usr/local/openmpi-1.2.5/lib/openmpi/mca_pml_ob1.so
#3  0x00007f32d2c678e8 in PMPI_Recv ()
   from /usr/local/openmpi-1.2.5/lib/libmpi.so.0
#4  0x00007f32d2dc853b in pmpi_recv__ ()
   from /usr/local/openmpi-1.2.5/lib/libmpi_f77.so.0
#5  0x0000000000400d84 in MAIN__ () at deadlock0.f:26
#6  0x0000000000400dde in main (argc=Variable "argc" is not available.
) at ../../../libgfortran/fmain.c:18
(gdb) frame 5
#5  0x0000000000400d84 in MAIN__ () at deadlock0.f:26
26              .     mpi_comm_world,status,error)
Current language:  auto; currently fortran
(gdb) info local
error = 0
i = 10001
rank = 1
status = (1, 0, 521689840, 32767, 0)
size = 2
y = (2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46
```

Deadlock - solution 1

```
if( rank .eq. 0)then
   call MPI_Send(y, nmax, MPI_REAL, 1, 2, MPI_COMM_WORLD, ierr)
   call MPI_Recv(x, nmax, MPI_REAL, 1, 1, MPI_COMM_WORLD, status, ierr)
else if ( rank .eq. 1)then
   call MPI_Send(x, nmax, MPI_REAL, 0, 1, MPI_COMM_WORLD, ierr)
   call MPI_Recv(y, nmax, MPI_REAL, 0, 2, MPI_COMM_WORLD, status, ierr)
endif
```

- cp deadlock0.f deadlock1.f
- or
- cp deadlock0.c deadlock1.c

- Make the above changes, compile and run the program

Deadlock - solution 1 continued

- change the value of nmax from 100 to 10000
- recompile and run the program again

Deadlock - solution 2

```
if( rank .eq. 0)then
    call MPI_Recv(x, nmax, MPI_REAL, 1, 1, MPI_COMM_WORLD, status, ierr)
    call MPI_Send(y, nmax, MPI_REAL, 1, 2, MPI_COMM_WORLD, ierr)
else if ( rank .eq. 1)then
    call MPI_Send(x, nmax, MPI_REAL, 0, 1, MPI_COMM_WORLD, ierr)
    call MPI_Recv(y, nmax, MPI_REAL, 0, 2, MPI_COMM_WORLD, status, ierr)
endif
```

More debugging

- Setting breakpoints

  eg. break deadlock2.f:17
- Check for deadlock and unbalanced send/ receive.
- Make use of tags to make sure that the correct message is received.
- Write statements to make sure that the contents of messages are correct.
- Add MPI_Barrier to synchronize processes.

Nonblocking Sends and Receives

- Separate send (or receive) into initiation and completion
- Initiation is nonblocking thus allowing other instructions to be processed
- Completion stage can either be a blocking wait or a nonblocking test

MPI_ISEND/MPI_IRECV

- Similar to MPI_Send/MPI_Recv except for an addition of a request handle and the lack of a status in MPI_Irecv

MPI_Send(buf, count, type, dest, tag, comm, ierr)

MPI_Isend(buf, count, type, dest, tag, comm, **req**, ierr)

MPI_Recv(buf, count, type, source, tag, comm, **status**, ierr)

MPI_Irecv(buf, count, type, source, tag, comm, **req**, ierr)

Completion waiting and testing

- Completion waiting blocks until the initiated process is completed

```
MPI_Wait(req, status, ierr)
```

- Completion testing returns immediately with flag set to true if the process is complete

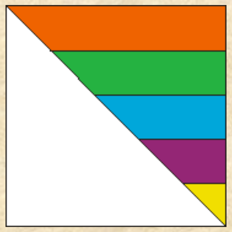
```
MPI_Test(req, flag, status, ierr)
```

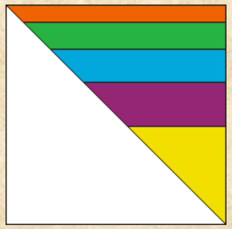
Load balancing

- Make sure that all the processes are busy.
- Make sure that each process has the same amount of work.
- sample - calculating distances

```
do j = 1, n-1
  do i = j+1, n
    dist(i,j) = sqrt((x(i)-x(j))**2)
  enddo
enddo
```

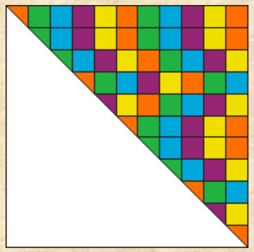
Load balancing continued





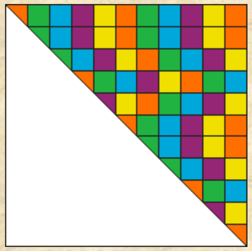
Dynamic scheduling

- Small chunks of work are given to each process
- As each process finishes the chunk of work, it gets the next chunk of work



Master - part 1

```
irank=1
do j=1,n
do i=j,n
  if (irank .lt. size) then
    send data to irank
    irank = irank +1
  else
    receive result from ifree
    send data to ifree
  endif
enddo
enddo
```



Master - part 2

```
do irank = 1, size-1
  receive the remaining results
  send termination signal to processes
enddo
```

Slave

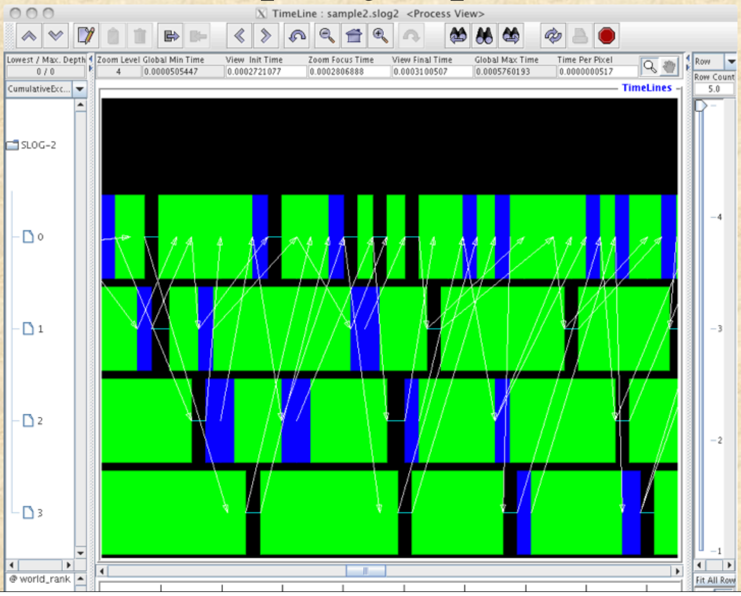
```
do
  receive data
  if( termination signal ) exit
  calculate
  send results
enddo
```

Sample2

```
mpefc -o sample2 sample2.f -mpilog
mpiexec -n 4 ./sample2
clog2TOslog2  Unknown.clog2
jumpshot  Unknown.slog2
```

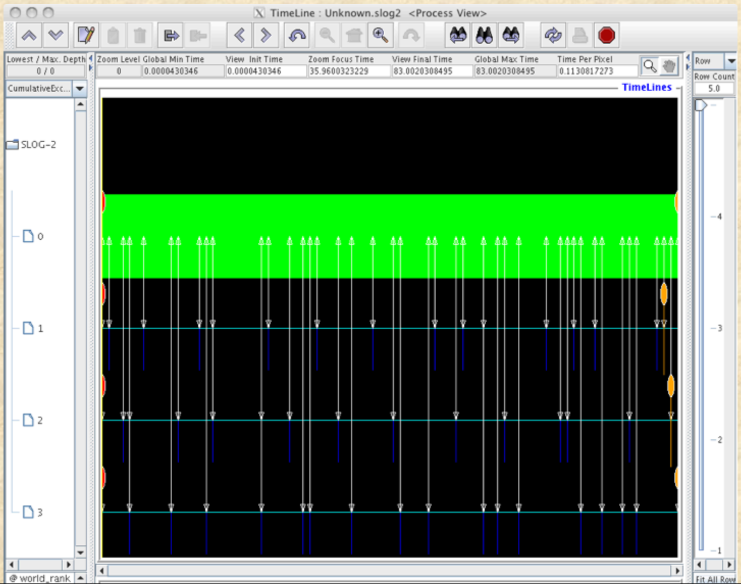
```
mpecc -o sample2 sample2.c -mpilog
mpiexec -n 4 ./sample2
clog2TOslog2  sample2.clog2
jumpshot  sample2.slog2
```

sample2 jumpshot


TimeLine : sample2.slog2 <Process View>

Lowest / Max. Depth: 0 / 0
Zoom Level: 4
Global Min Time: 0.0000505447
View Init Time: 0.0002721077
Zoom Focus Time: 0.0002806888
View Final Time: 0.0003100507
Global Max Time: 0.0005760193
Time Per Pixel: 0.0000000517
Row Count: 5.0

sample2a jumpshot


TimeLine : Unknown.slog2 <Process View>

Lowest / Max. Depth: 0 / 0
Zoom Level: 0
Global Min Time: 0.0000430346
View Init Time: 0.0000430346
Zoom Focus Time: 35.9600323229
View Final Time: 83.0020308495
Global Max Time: 83.0020308495
Time Per Pixel: 0.1130817273
Row Count: 5.0

- Books
  - Using MPI: Portable Parallel Programming with the Message Passing Interface
    - William Gropp, Ewing Lusk, and Anthony Skjellum
  - Parallel Programming with MPI
    - Peter Pacheco
- Websites
  - MPI:The Complete Reference
    - http://www.netlib.org/utk/papers/mpi-book/mpi-book.html
  - Introduction to MPI
    - http://ci-tutor.ncsa.uiuc.edu/browse.php
  - MPI and MPE routines
    - http://www-unix.mcs.anl.gov/mpi/www
- research.support@ualberta.ca