MPI Tutorial

V. Balaji
GFDL Princeton University

PICASSO Parallel Programming Workshop
Princeton NJ
4 March 2004
Getting started

- [http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html](http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html) Most of what you need will be provided up here on the screen, but please get a little used to navigating this user guide, just in case ...
Getting started

- [http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html](http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html) Most of what you need will be provided up here on the screen, but please get a little used to navigating this user guide, just in case ...

- `ssh-keygen -t dsa`
  (hit return when asked for passphrase)
  `cd ~/.ssh/`
  `cat id_dsa.pub >> authorized_keys`
  `cp /home/known_hosts .`
  `chmod 644 authorized_keys`

- `cp /home/balaji/hello.f .`
  `mpif77 hello.f -o hello.x`

- `cp /home/balaji/hello.c .`
  `mpicc hello.c -o hello.x`

- `mpirun -np 2 hello.x`
Exercise 1: “Hello world” in Fortran

```
cp /home/balaji/hello.f.

program hello
include 'mpif.h'
integer rank, size, ierror

call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)

print *, 'I am MPI process ', rank, ' of ', size

call MPI_FINALIZE(ierr)
end
```
Exercise 1: “Hello world” in C

cp /home/balaji/hello.c.

#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
{
    int rank, size;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    printf("I am MPI process %d of %d\n", rank, size);

    MPI_Finalize();

    return 0;
}
Exercise 1 comments

Try it at different process counts. How high can you go?
Exercise 1 comments

Try it at different process counts. How high can you go?

MPI processes are not necessarily bound to a processor: you could schedule many processes on a single PE. (It’s probably not the most efficient thing to do.)
Exercise 1 comments

Try it at different process counts. How high can you go?

MPI processes are not necessarily bound to a processor: you could schedule many processes on a single PE. (It’s probably not the most efficient thing to do.)

What’s remarkable about the output?
Exercise 1 comments

Try it at different process counts. How high can you go?

MPI processes are not necessarily bound to a processor: you could schedule many processes on a single PE. (It’s probably not the most efficient thing to do.)

What’s remarkable about the output?

The order in which lines get written to stdout is not predictable (a race condition).
Exercise 2: neighbour on a ring

Assume processes are in a ring, and find the rank of your neighbour on the left or right.

Use blocking sends and receives (MPI_Send/Recv)

- In Fortran:

```fortran
integer buf, count, dest, source, tag, ierror, status(MPI_STATUS_SIZE)
call MPI_SEND(buf, count, MPI_INTEGER, dest, tag, MPI_COMM_WORLD, ierror)
call MPI_RECV(buf, count, MPI_INTEGER, source, tag, MPI_COMM_WORLD, status, ierror)
```

- In C:

```c
int buf, count, dest, source, tag;
MPI_Status status;
MPI_Send(&buf, count, MPI_INTEGER, dest, tag, MPI_COMM_WORLD);
MPI_Recv(&buf, count, MPI_INTEGER, source, tag, MPI_COMM_WORLD, &status);
```
Exercise 2: neighbour on a ring

Assume processes are in a ring, and find the rank of your neighbour on the left or right.

Use blocking sends and receives (MPI_Send/Recv)

- In Fortran:

  ```fortran
  integer buf, count, dest, source, tag, ierror, status(MPI_STATUS_SIZE)
  call MPI_SEND(buf, count, MPI_INTEGER, dest, tag, MPI_COMM_WORLD, ierror)
  call MPI_RECV(buf, count, MPI_INTEGER, source, tag, MPI_COMM_WORLD, status, ierror)
  
  Hint: use mod(rank+1, size) and mod(rank+size-1, size) for your ring neighbours.
  ```

- In C:

  ```c
  int buf, count, dest, source, tag;
  MPI_Status status;
  MPI_Send(&buf, count, MPI_INTEGER, dest, tag, MPI_COMM_WORLD);
  MPI_Recv(&buf, count, MPI_INTEGER, source, tag, MPI_COMM_WORLD, &status);
  
  Hint: use (rank+1)%size and (rank+size-1)%size for your ring neighbours.
  ```
Exercise 2: a solution (Fortran)

program hello
include 'mpif.h'
integer :: rank, size, ierror, tag=99, left,
& status(MPI_STATUS_SIZE)

call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)

call MPI_SEND( rank, 1, MPI_INTEGER, mod(rank+1,size), tag,
& MPI_COMM_WORLD, ierr )
call MPI_RECV( left, 1, MPI_INTEGER, mod(rank+size-1,size), tag,
& MPI_COMM_WORLD, status, ierr )
print *, 'I am MPI process ', rank, ' of ', size,
& ', on my left is ', left

call MPI_FINALIZE(ierr)
end
Exercise 2: a solution (C)

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
{
    int rank, size, left, tag = 99;
    MPI_Status status;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Send(&rank, 1, MPI_INTEGER, (rank + 1) % size, tag, MPI_COMM_WORLD);
    MPI_Recv(&left, 1, MPI_INTEGER, (rank + size - 1) % size, tag, MPI_COMM_WORLD, &status);

    printf("I am MPI process %d of %d, on my left is %d\n", rank, size, left);

    MPI_Finalize();

    return 0;
}
```
Exercise 3: pass an array around the ring

Instead of a scalar, let’s try passing an array.

- In Fortran:

```fortran
parameter (MAX=100)
integer a(MAX)

do i = 1,MAX
   a(i) = rank*MAX + i - 1
end do
```

- In C:

```c
#define MAX 100

int i, array[MAX];

for ( i=0; i<MAX; i++ ) {
   array[i] = rank*MAX + i;
}
```

Try it with MAX set to a large number (100000). What happens?
Exercise 3: pass an array around the ring

Instead of a scalar, let’s try passing an array.

- In Fortran:

```fortran
parameter (MAX=100)
integer a(MAX)

do i = 1,MAX
    a(i) = rank*MAX + i - 1
end do
```

- In C:

```c
#define MAX 100

int i, array[MAX];

for ( i=0; i<MAX; i++ ) {
    array[i] = rank*MAX + i;
}
```

Try it with MAX set to a large number (100000). What happens?
Problems with blocked communication: deadlock

- On PE 0:
  
  ```c
  MPI_Send( buf, count, type, 1, tag, comm);
  MPI_Recv( buf, count, type, 1, tag, comm, &status);
  ```

- On PE 1:
  
  ```c
  MPI_Send( buf, count, type, 0, tag, comm);
  MPI_Recv( buf, count, type, 0, tag, comm, &status);
  ```

The `send()` on PE 0 cannot complete until PE 1 calls `recv()`; and vice versa.
Resolving a deadlock: reverse *send/recv*

Reversing the order of *send/recv* on one of the processes will work:

- On PE 0:

  
  ```
  MPI_Recv( buf, count, type, 1, tag, comm, &status);
  MPI_Send( buf, count, type, 1, tag, comm);
  ```

- On PE 1:

  ```
  MPI_Send( buf, count, type, 0, tag, comm);
  MPI_Recv( buf, count, type, 0, tag, comm, &status);
  ```
Resolving a deadlock: reverse \texttt{send/receiv}

Reversing the order of \texttt{send/receiv} on one of the processes will work:

- On PE 0:

  
  \begin{verbatim}
  MPI_Recv( buf, count, type, 1, tag, comm, &status);
  MPI_Send( buf, count, type, 1, tag, comm);
  \end{verbatim}

- On PE 1:

  
  \begin{verbatim}
  MPI_Send( buf, count, type, 0, tag, comm);
  MPI_Recv( buf, count, type, 0, tag, comm, &status);
  \end{verbatim}

On how many processes do you need to reverse \texttt{send/receiv} to guarantee no deadlocks?
Why was there no deadlock on short messages?

- On PE 0:

  MPI_Send( buf, count, type, 1, tag, comm);
  MPI_Recv( buf, count, type, 1, tag, comm, &status);

- On PE 1:

  MPI_Send( buf, count, type, 0, tag, comm);
  MPI_Recv( buf, count, type, 0, tag, comm, &status);

Under the covers, MPI is using internal buffers (the “message envelope”) to cache messages. A blocked comm pattern may work for some values of count, and then fail as count is increased.
Why was there no deadlock on short messages?

- On PE 0:
  
  ```c
  MPI_Send( buf, count, type, 1, tag, comm);
  MPI_Recv( buf, count, type, 1, tag, comm, &status);
  ```

- On PE 1:
  
  ```c
  MPI_Send( buf, count, type, 0, tag, comm);
  MPI_Recv( buf, count, type, 0, tag, comm, &status);
  ```

Under the covers, MPI is using internal buffers (the “message envelope”) to cache messages. A blocked comm pattern may work for some values of `count`, and then fail as `count` is increased.

Find the size of the message envelope on the cluster.
MPI: non-blocking send and receive

A better solution is to make at least one of send/recv non-blocking. A non-blocking call returns control to the caller after initiating communication. The status of the message buffer is undefined until a corresponding `wait()` call is posted to check the status of the message.

- On PE 0:

```c
MPI_Request request;
MPI_Isend( buf, count, type, 1, tag, comm, &request);
... // other work that does not modify or free buf
MPI_Wait( &request, &status );
    buf = ... 
```

- On PE 1:

```c
MPI_Irecv( buf, count, type, 0, tag, comm, &request);
... // other work that does not require the contents of buf
MPI_Wait( &request, &status );
    ... = buf ...
```

`MPI_Wait()` is a blocking call. `MPI_Test()` can be used as an alternative to check if the pending communication is complete, without blocking.
Isend/recv instead of send/recv

- In Fortran:

```fortran
integer request
call MPI_ISEND( buf, count, MPI_INTEGER, dest, tag,
                 MPI_COMM_WORLD, request, ierror )
call MPI_RECV( buf, count, MPI_INTEGER, source, tag,
                MPI_COMM_WORLD, status, ierror )
call MPI_WAIT( request, status, ierror )
```

- In C:

```c
MPI_Request request;
MPI_Isend(buf, count, MPI_INTEGER, dest, tag, MPI_COMM_WORLD, &request
MPI_Recv(buf, count, MPI_INTEGER, source, tag, MPI_COMM_WORLD, &status
MPI_Wait( &request, &status );
```
Isend/recv instead of send/recv

- In Fortran:

  integer request  
call MPI_ISEND( buf, count, MPI_INTEGER, dest, tag,  
                 MPI_COMM_WORLD, request, ierror )  
call MPI_RECV( buf, count, MPI_INTEGER, source, tag,  
               MPI_COMM_WORLD, status, ierror )  
call MPI_WAIT( request, status, ierror )

- In C:

  MPI_Request request;  
  MPI_Isend(buf, count, MPI_INTEGER, dest, tag, MPI_COMM_WORLD, &request  
  MPI_Recv(buf, count, MPI_INTEGER, source, tag, MPI_COMM_WORLD, &status  
  MPI_Wait( &request, &status );

Can you reverse the order of Isend and recv?
Last exercise: diffusion equation

\[ \frac{\partial u}{\partial t} + K \frac{\partial^2 u}{\partial x^2} = 0 \]  \hspace{1cm} (1)

In discrete form:

\[ u_{i}^{n+1} = u_{i}^{n} + c \frac{\Delta t}{2 \Delta x} (u_{i+1}^{n} + u_{i-1}^{n} - 2u_{i}^{n}) \]  \hspace{1cm} (2)

Assume \( P < N \), and that \( P \) is an exact divisor of \( N \).