

for MMD / SMP



• - MPI (Message Passing Interface)

Requirements → a) Separate processes on different machine.

b) Method of sending & receiving messages.

• To communicate processes need a way to communicate with each other. Each process

is assigned a unique identifier called rank. Start with 0 / End at size-1

• All processes of MPI program are members of default communicator MPI_COMM_WORLD

(int x arg, char ** arg) Redefinition

MPI_Init (Initialize MPI)

MPI_Finalize (Terminates MPI)

MPI_Comm_Size (Determines no. of processes)

MPI_Comm_Rank ()

MPI_Send (Send message)

MPI_Recv (Receive message)

Return codes (MPI_SUCCESS & MPI_ERROR)

MPI_COMM_WORLD (root communicator)

Communicator → MPI_Comm

• Generic Structure:

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

```
void main (int argc, char* argv [7])
```

```
{ int np, rank, ierr;
```

```
  ierr = MPI_Init (&argc, &argv)
```

```
  MPI_Comm_rank (MPI_COMM_WORLD, &rank)
```

```
  MPI_Comm_size (MPI_COMM_WORLD, &np);
```

```
  // --
```

```
  // --
```

```
  ierr = MPI_Finalize();
```

```
}
```

• MPI_Send (void* data, int count, MPI_Datatype type, int dest,
int tag, MPI_Comm comm)

MPI_Recv (void* data, int count, MPI_Datatype type, int
source, int tag, MPI_Comm comm, MPI_Status* status)

→ Examples

MPI_Send (&data, 9, MPI_CHAR, 1, 0, MPI_COMM_WORLD)

MPI_Recv (&data, 9, MPI_CHAR, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE)

MPI_Get_Count (&status, MPI_INT, &number_count)

- - Non blocking Send & Receive.

`MPI_ISEND (buf, count, datatype, dest, tag, comm, request)`

`MPI_IRECV (buf, count, datatype, dest, tag, comm, request)`

`MPI_WAIT (request, status)`

`MPI_TEST (request, flag, status)`

- - MPI_Probe.

• - `MPI_Bcast (void* buf, int count, MPI_Datatype dtype, int root, MPI_Comm comm)`
Same data \rightarrow many sources

`MPI_Bcast (message, 1, MPI_CHAR, root, MPI_COMM_WORLD)`

- - `MPI_Scatter (void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`
different data \rightarrow many sources

`MPI_Scatter (send, 1, MPI_INT, recv, 1, MPI_INT, 0, MPI_COMM_WORLD)`

- - `MPI_Gather (void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)`

`MPI_Gather (send, 3, MPI_INT, recvbuf, 3, MPI_INT, 0, MPI_COMM_WORLD)`
many sources \rightarrow 1

`MPI_Scatterv (const void * sendbuf, const int sendcount[], const int displs[],
MPI_Datatype sendtype, void * recvbuf, int recvcount,
MPI_Datatype recvtype, int root, MPI_Comm comm)`

• Transpose.

`MPI_Allgather (buf1, 2, MPI_INT, buf2, 2, MPI_INT,
MPI_COMM_WORLD)`

• Synchronization

① `MPI_Barrier (MPI_COMM_WORLD)`

② `MPI_Reduce (void * sendbuf, void * recvbuf, int count,
MPI_Datatype type, MPI_Op op, int root,
MPI_Comm comm)`

`MPI_Reduce (sum, &global_sum, 1, MPI_INT, MPI_SUM, 0,
MPI_COMM_WORLD)`

③

`MPI_Allreduce (---, int * result, ---)`

↓
it contains
all processes
reduce result.