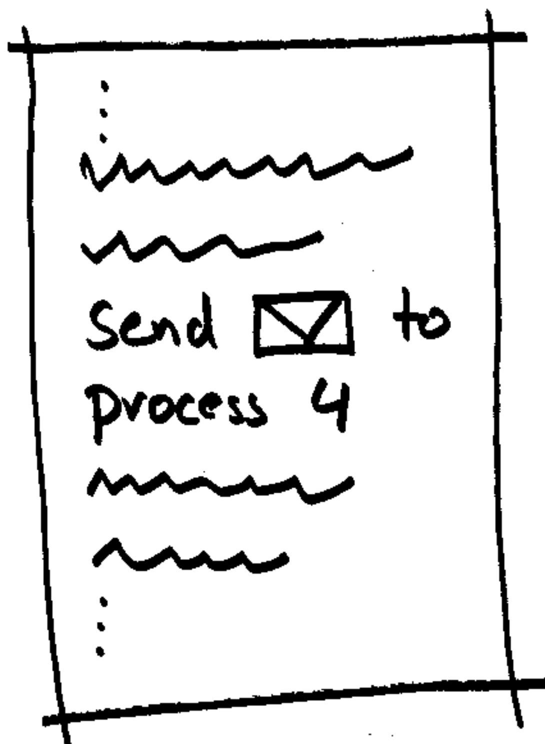
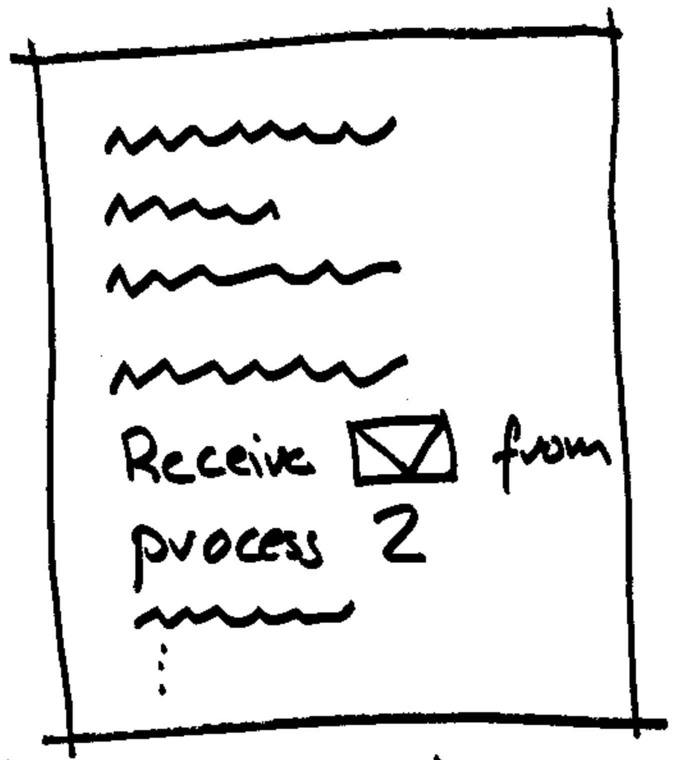
MPI - Message Passing Interface





MPI - The good news

- a standard and portable message-passing system
- runs both on
  - tightly coupled massively parallel machines (MPPs), and
  - network of workstations (NOWs)
- heterogeneous computing
- defines a core of library routines (callable from C or fortran)
- runs on dozens of machines
- Can be very efficient

MPI - The bad news

- Its very difficult to
  - design message passing programs
  - debug message passing programs.
- "the assembly language of parallel computing"

Greetings.c

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"

main(int argc, char* argv[]) {
    int         my_rank;       /* rank of process      */
    int         p;             /* number of processes  */
    int         source;        /* rank of sender       */
    int         dest;          /* rank of receiver     */
    int         tag = 0;       /* tag for messages     */
    char        message[100];  /* storage for message  */
    MPI_Status  status;        /* return status for    */
                               /* receive              */

    /* Start up MPI */
    MPI_Init(&argc, &argv);

    /* Find out process rank  */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* Find out number of processes */
    MPI_Comm_size(MPI_COMM_WORLD, &p);

    if (my_rank != 0) {
        /* Create message */
        sprintf(message, "Greetings from process %d!",
            my_rank);
        dest = 0;
        /* Use strlen+1 so that '\0' gets transmitted */
        MPI_Send(message, strlen(message)+1, MPI_CHAR,
            dest, tag, MPI_COMM_WORLD);
    } else { /* my_rank == 0 */
        for (source = 1; source < p; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source, tag,
                MPI_COMM_WORLD, &status);
            printf("%s\n", message);
        }
    }

    /* Shut down MPI */
    MPI_Finalize();
} /* main */
```

Notes

- Single program multiple data (SPMP)

```
if (my_rank != 0)
        ⋮
else
        ⋮
```

- Output, if run with 4 processes

```
Greetings from process 1!
Greetings from process 2!
Greetings from process 3!
Greetings from process 4!
```

General MPI programs

```
            :
#include "mpi.h"
            :
main(int argc, char* argv[]) {
            :
    /* No MPI functions called before this */
    MPI_Init(&argc, &argv);
            :
    MPI_Finalize();
    /* No MPI functions called after this */
            :
} /* main */
            :
```

Note

- all identifiers start with MPI_
- For constants remaining characters are capitals (ie. MPI_CHAR)
- For functions, capital followed by lower case (ie. MPI_Init)

Finding Out about the Rest of the World.

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

First parameter

- a "communicator"
- basically a collection of processes that can send messages to each other
- MPI_COMM_WORLD predefined as all processes running when program starts

Second Parameter

- the return value
- the rank of the process

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

First Parameter

- a communicator

Second Parameter

- the return value
- the # of processes executing the program

Message = Data + Envelope

Data = a pointer to a block of memory
+ a count of the # of items
+ the data type of the items

Envelope = the rank of the receiver
+ the rank of the sender
+ a "tag" to indicate the purpose of the message
+ a communicator

Sending Messages

```
int MPI_Send(
        void*          message    /* in */,
        int            count      /* in */,
        MPI_Datatype   datatype   /* in */,
        int            dest       /* in */,
        int            tag        /* in */,
        MPI_Comm       comm       /* in */)
```

Notes

- message – a sequence of count values each having MPI type datatype
- "a push communication mechanism"

MPI Datatypes

Predefined MPI datatypes

MPI datatype	*C datatype*
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Why does MPI define its own types?

Receiving Messages

```
int MPI_Recv(
        void*          message    /* out */,
        int            count      /* in  */,
        MPI_Datatype   datatype   /* in  */,
        int            source     /* in  */,
        int            tag        /* in  */,
        MPI_Comm       comm       /* in  */,
        MPI_Status*    status     /* out */)
```

Note

- receiving buffer may be longer than message
- source can be wildcard, MPI_ANY_SOURCE
- tags & communicators partition message space
- tag can be wildcard, MPI_ANY_TAG

Notes on receiving cont.

- Status, information on data received
- Status is a struct

```
Status -> MPI_SOURCE
Status -> MPI_TAG
Status -> MPI_ERROR
```

- Status also contains size of message (ie. # of elements received of given type)

```
int MPI_Get_count(
    MPI_Status*    status     /* in */,
    MPI_Datatype   datatype   /* in */,
    int*           count_ptr  /* out */)
```