Introduction: SEND and RECEIVE

General MPI commands

```
/* Preprocessor directive */
#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 */


} /* end of main program */
```

Two basic calls

```
/* returns the number of nodes currently being used */
MPI_Comm_size(MPI_COMM_WORLD, &numnodes);

/* returns the number of the node which calls this function */
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

Sending and Receiving Messages

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

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 */)
```

Description of parameters

message - actual data being transmitted

count - how many items

datatype - MPI datatype such as: MPI_INT, MPI_FLOAT, MPI_CHAR

dest and source - processors numbers for sending to and receiving from

tag - message id

comm - group of processors in communicatior

status - returns the source and tag of received message

A Simple Example (Pacheco's book)

```
/* greetings.c -- greetings program
 *
 * Send a message from all processes with rank != 0 to process 0.
 *    Process 0 prints the messages received.
 *
 * Input: none.
 * Output: contents of messages received by process 0.
 *
 * See Chapter 3, pp. 41 & ff in PPMPI.
 */
#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 */
```