ECE 432/532
Programming for Parallel Processors

Example 1

- Write a program that uses MPI and has each MPI process prints:

Hello world from process i of n

using the rank in MPI_COMM_WORLD for i and the size of MPI_COMM_WORLD for n.

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

int main( argc, argv )
int  argc;
char **argv;
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

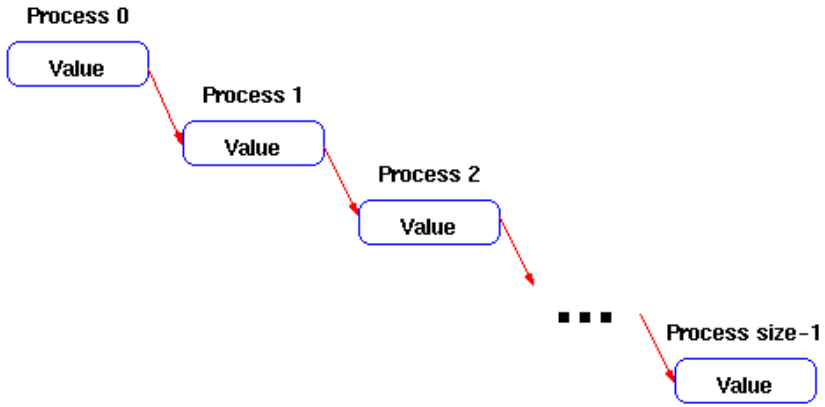
Example 2

- Write a program that reads an integer value from the terminal and distributes the value to all of the MPI processes.
- Each process should print out its rank and the value it received.
- Values should be read until a negative integer is given as input.

```
int main( argc, argv )
int argc;
char **argv;
{
    int rank, value;
    MPI_Init( &argc, &argv );

    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    do {
       if (rank == 0)
           scanf( "%d", &value );

       MPI_Bcast( &value, 1, MPI_INT, 0, MPI_COMM_WORLD );

       printf( "Process %d got %d\n", rank, value );
    } while (value >= 0);

    MPI_Finalize( );
    return 0;
}
```

Example 3

- Write a program that takes data from process zero and sends it to all of the other processes by sending it in a ring.



```
int main( argc, argv ) {
int argc;
char **argv;
{
    int rank, value, size;
    MPI_Status status;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    do {
         if (rank == 0) {
             scanf( "%d", &value );
             MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
         }
         else {
             MPI_Recv( &value, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD,
                       &status );
             if (rank < size - 1)
                  MPI_Send( &value, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD );
         }
         printf( "Process %d got %d\n", rank, value );
    } while (value >= 0);

    MPI_Finalize( );
    return 0; }
```

Example 4

- Write a program that calculates the integral from a to b of the function $f(x) = x^3$

```
int main(void) {
   int my_rank, comm_sz, n = 1024, local_n;
   double a = 0.0, b = 3.0, h, local_a, local_b;
   double local_int, total_int;
   int source;

   MPI_Init(NULL, NULL);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

   h = (b-a)/n;          /* h is the same for all processes */
   local_n = n/comm_sz;  /* So is the number of trapezoids  */
   local_a = a + my_rank*local_n*h;
   local_b = local_a + local_n*h;
   local_int = Trap(local_a, local_b, local_n, h);

   if (my_rank != 0) {
      MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
   } else {
      total_int = local_int;
      for (source = 1; source < comm_sz; source++) {
         MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
         total_int += local_int;
      }
   }
   if (my_rank == 0) {
      printf("With n = %d trapezoids, our estimate\n", n);
      printf("of the integral from %f to %f = %.15e\n", a, b, total_int);
   }
   MPI_Finalize();
   return 0;
} /*  main  */
```

```
double Trap(
      double left_endpt  /* in */,
      double right_endpt /* in */,
      int    trap_count  /* in */,
      double base_len    /* in */) {

   double estimate, x;
   int i;

   estimate = (f(left_endpt) + f(right_endpt))/2.0;
   for (i = 1; i <= trap_count-1; i++) {
      x = left_endpt + i*base_len;
      estimate += f(x);
   }
   estimate = estimate*base_len;

   return estimate;
} /*  Trap  */




double f(double x) {   return x*x*x;} /* f */
```

Example 5

- Write a program that calculates the sum of two vectors

```
int main(void) {
   int n, local_n;
   int comm_sz, my_rank;
   double *local_x, *local_y, *local_z;
   MPI_Comm comm;
   MPI_Init(NULL, NULL);
   comm = MPI_COMM_WORLD;
   MPI_Comm_size(comm, &comm_sz);
   MPI_Comm_rank(comm, &my_rank);

   Read_n(&n, &local_n, my_rank, comm_sz, comm);
   Allocate_vectors(&local_x, &local_y, &local_z, local_n, comm);

   Read_vector(local_x, local_n, n, "x", my_rank, comm);
   Print_vector(local_x, local_n, n, "x is", my_rank, comm);
   Read_vector(local_y, local_n, n, "y", my_rank, comm);
   Print_vector(local_y, local_n, n, "y is", my_rank, comm);

   Parallel_vector_sum(local_x, local_y, local_z, local_n);
   Print_vector(local_z, local_n, n, "The sum is", my_rank, comm);

   free(local_x);
   free(local_y);
   free(local_z);

   MPI_Finalize();

   return 0;
}  /* main */
```

```
void Read_n(
      int*      n_p        /* out */,
      int*      local_n_p  /* out */,
      int       my_rank    /* in  */,
      int       comm_sz    /* in  */,
      MPI_Comm  comm       /* in  */) {

   int local_ok = 1;

   if (my_rank == 0) {
      printf("What's the order of the vectors?\n");
      scanf("%d", n_p);
   }
   MPI_Bcast(n_p, 1, MPI_INT, 0, comm);
   *local_n_p = *n_p/comm_sz;
}  /* Read_n */


void Allocate_vectors(
      double**   local_x_pp  /* out */,
      double**   local_y_pp  /* out */,
      double**   local_z_pp  /* out */,
      int        local_n     /* in  */,
      MPI_Comm   comm        /* in  */) {
   int local_ok = 1;

   *local_x_pp = malloc(local_n*sizeof(double));
   *local_y_pp = malloc(local_n*sizeof(double));
   *local_z_pp = malloc(local_n*sizeof(double));

}  /* Allocate_vectors */
```

```
void Read_vector(
      double    local_a[]   /* out */,
      int       local_n     /* in  */,
      int       n           /* in  */,
      char      vec_name[]  /* in  */,
      int       my_rank     /* in  */,
      MPI_Comm  comm        /* in  */) {

   double* a = NULL;
   int i;
   int local_ok = 1;

   if (my_rank == 0) {
      a = malloc(n*sizeof(double));
      printf("Enter the vector %s\n", vec_name);
      for (i = 0; i < n; i++)
         scanf("%lf", &a[i]);
      MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_n, MPI_DOUBLE, 0, comm);
      free(a);
   } else {
         MPI_Scatter(a, local_n, MPI_DOUBLE, local_a, local_n, MPI_DOUBLE, 0, comm);
   }
}  /* Read_vector */
```

```
void Print_vector(
      double    local_b[]  /* in */,
      int       local_n    /* in */,
      int       n          /* in */,
      char      title[]    /* in */,
      int       my_rank    /* in */,
      MPI_Comm  comm       /* in */) {

   double* b = NULL;
   int i;
   int local_ok = 1;

   if (my_rank == 0) {
      b = malloc(n*sizeof(double));
      MPI_Gather(local_b, local_n, MPI_DOUBLE, b, local_n, MPI_DOUBLE, 0, comm);
      printf("%s\n", title);
      for (i = 0; i < n; i++)
         printf("%f ", b[i]);
      printf("\n");
      free(b);
   } else {
      MPI_Gather(local_b, local_n, MPI_DOUBLE, b, local_n, MPI_DOUBLE, 0, comm);
   }
}  /* Print_vector */
```

```
void Parallel_vector_sum(
      double  local_x[]  /* in  */,
      double  local_y[]  /* in  */,
      double  local_z[]  /* out */,
      int     local_n    /* in  */) {
   int local_i;

   for (local_i = 0; local_i < local_n; local_i++)
      local_z[local_i] = local_x[local_i] + local_y[local_i];
}  /* Parallel_vector_sum */
```