Learning MPI by Examples: Part III

Blocking and non-blocking communications
Learning MPI by Examples: Part III

• Previous parallel programming with MPI blocking Send/Receive, which means
  – process 1 (or processes other than 0) is ready for receiving message from process 0. Once it is ready, it deserves for receiving. If process 0 doesn't send a message, the process 1 is idle and waiting for receiving the message
  – it is not synchronous communication, which means sender would send message until it receives confirmation from receiver
Learning MPI by Examples: Part III

- blocking communication should be avoided
- The use of non-blocking communication can be used to provide dramatic improvements in the performance of message passing programs
- Use MPI_Isend() and MPI_Irecv()
  • I stands for immediate
- Use MPI_Wait() to complete the non-blocking communication
Learning MPI by Examples: Part III

- Parallel programming with MPI non-blocking Send/Receive
  - do not make processes idle
  - Using the following MPI functions
    - MPI_Init and MPI_Finalize
    - MPI_Comm_rank
    - MPI_Comm_size
    - MPI_Irecv
    - MPI_Isend
    - MPI_Wait()
Learning MPI by Examples: Part III

• MPI _ISend() syntax:

```
int MPI_Isend ( void* buffer /* in */,
               int count /* in */,
               MPI_Datatype /* in */,
               int dest /* in */,
               int tag /* in */,
               MPI_Comm comm /* in */,
               MPI_Request* request /* out */)
```

Learning MPI by Examples: Part III

- **MPI\_IRecv() syntax:**

```c
int MPI\_IRecv ( void* buffer /*in */,
                int count /* in */,
                MPI\_Datatypes /*in */,
                int source /*in*/,
                int tag /*in*/,
                MPI\_Comm comm /*in*/,
                MPI\_Request* request /*out */)```

/* nbtrap.c -- Parallel Trapezoidal Rule, nonblocking
 * sending
 * Input: None.
 * Output: Estimate of the integral from a to b of f(x)
 *         using the trapezoidal rule and n trapezoids.
 *
 * Algorithm:
 * 1. Each process calculates "its" interval of integration.
 * 2. Each process estimates the integral of f(x)
 *    over its interval using the trapezoidal rule.
 * 3a. Each process != 0 sends its integral to process 0.
 * 3b. Process 0 sums the calculations received from the individual processes and prints the result.
Notes:
* 1. $f(x)$, $a$, $b$, and $n$ are all hardwired.
* 2. The number of processes ($p$) should evenly divide the number of trapezoids ($n = 1024$)

```c
#include <stdio.h>

/* We'll be using MPI routines, definitions, etc. */
#include "mpi.h"

main(int argc, char** argv)
{
    int my_rank;    /* My process rank */
    int p;         /* The number of processes */
    float a = 0.0; /* Left endpoint */
```

Aug. 6-7, 2009  Iowa HPC Summer School 2009
float b = 1.0;  /* Right endpoint */
int n = 1024;  /* Number of trapezoids in each subintegrals */
float h;      /* Trapezoid base length */

/* local_a and local_b are the bounds for each integration performed in individual process */

float local_a;  /* Left endpoint my process */
float local_b;  /* Right endpoint my process */
float local_h;  /* trapezoid base length for each subintegral */
float integral; /* Integral over my interval */
float total;   /* Total integral */
int source;    /* Process sending integral */
int dest = 0;  /* All messages go to 0 */
int tag = 0;
MPI_Status status;
MPI_Request send_req;

/* Trap function prototype. Trap function is used to calculate local integral */

float Trap(float local_a, float local_b, int local_n);

/* Let the system do what it needs to start up MPI */
MPI_Init(&argc, &argv);

/* Get my process rank */
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

/* Find out how many processes are being used */
MPI_Comm_size(MPI_COMM_WORLD, &p);
h = (b-a)/n;    /* h is the same for all processes */
local_h = h/p;  /* So is the number of trapezoids */

local_a = a + my_rank*local_h*n;
local_b = local_a + local_h*n;
integral = Trap(local_a, local_b, n);

if (my_rank == 0)
{
/* Add up the integrals calculated by each process */
total = integral;   /* this is the integral calculated by process 0 */
for (source = 1; source < p; source++)
{
    MPI_Recv(&integral, 1, MPI_FLOAT, source, tag,
              MPI_COMM_WORLD, &status);
    total = total + integral;
}
}
else
{
    printf("The integral calculated from process %d is %f\n", my_rank, integral);
}

/* MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD) */

MPI_Isend(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
MPI_Wait(&send_req, &status);

/* Print the result */
if (my_rank == 0)
{
    printf("With n = %d trapezoids, our estimate\n", n);
    printf("of the integral from %f to %f = %f\n", a, b, total);
}
/* Shut down MPI */
MPI_Finalize();
}

float Trap (  
    float local_a /* in */,
    float local_b /* in */,
    int local_n /* in */)
{
    float integral; /* Store result in integral */
    float x;
    int i;
    float local_h;

    float f(float x); /* function we're integrating */
local_h=(local_b-local_a)/local_n;

integral = (f(local_a) + f(local_b))/2.0;
x = local_a;
for (i = 1; i <= local_n-1; i++)
{
    x = x + local_h;
    integral = integral + f(x);
}
integral = integral*local_h;
return integral;
float f(float x)
{
    float return_val;
    /* Calculate f(x). */
    /* Store calculation in return_val. */
    return_val = x*x;
    return return_val;
} /* f */
Learning MPI by Examples: Part II

- Example of parallel programming using non-blocking Sending

```bash
mpirun -np 8 a.out
```

The integral calculated from process 4 is 0.039714
The integral calculated from process 5 is 0.059245
The integral calculated from process 7 is 0.110026
The integral calculated from process 2 is 0.012370
The integral calculated from process 3 is 0.024089
The integral calculated from process 1 is 0.004557
The integral calculated from process 6 is 0.082682

With \( n = 1024 \) trapezoids, our estimate of the integral from 0.000000 to 1.000000 = 0.333333