Lecture 6

Programming Using the Message-Passing Paradigm III

MPI: the Message Passing Interface; Parallelization Application Example - Pi Computation

IKC-MH.57 *Introduction to High Performance and Parallel Computing* at April 14, 2023

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University

Contents

1. **Parallelization Application Example**
   Pi Computation

Pi Computation I

- π by numerically evaluating the integral

$$\int_0^1 \frac{1}{1+x^2}dx = \frac{\pi}{4}$$

- Midpoint Rule for $\int_a^b f(x)dx \approx (b-a)f(x_m)$

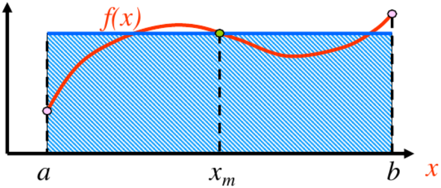


Figure: Midpoint Rule.

- Midpoint Rule becomes

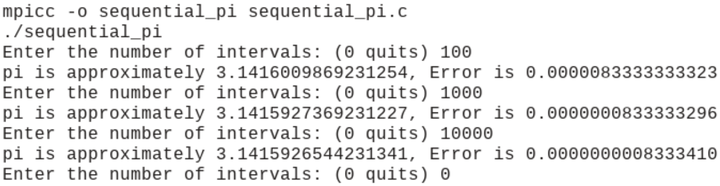
$$\int_0^1 \frac{1}{1+x^2}dx \approx \sum_{i=1}^{n} \frac{1}{1+\left(\frac{i-0.5}{n}\right)^2}$$

Pi Computation II

Sequential Code:

```
#include <stdio.h>
#include <math.h>
int main(int argc, char* argv[])
{
    int done = 0, n, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, h, sum, x;
    while (!done)
    {
        printf("Enter the number of intervals: (0 quits) ");
        scanf("%d",&n);
if (n == 0) break; /* Quit when "0" entered*/
        /* Integral limits are from 0 to 1 */
        h   = (1.0-0.0)/(double)n; /* Step length*/
        sum = 0.0;           /* Initialize sum variable */
        /* loop over interval for integration*/
        for (i = 1; i <= n; i += 1)
        {
            x = h * ((double)i - 0.5); /* Middle point at step */
            sum += 4.0 / (1.0 + x*x);  /* Sum up at each step */
//("i=%d x=%f sum=%f \n",i,x,sum); /* print intermediate steps */
        }
        mypi = h * sum; /* Obtain resulting pi number */
        printf("pi is approximately %.16f, Error is %.16f\n",mypi, \\
        fabs(mypi - PI25DT));
    }
}
```

Pi Computation III



```
mpicc -o sequential_pi sequential_pi.c
./sequential_pi
Enter the number of intervals: (0 quits) 100
pi is approximately 3.1416009869231254, Error is 0.0000083333333323
Enter the number of intervals: (0 quits) 1000
pi is approximately 3.1415927369231227, Error is 0.0000000833333296
Enter the number of intervals: (0 quits) 10000
pi is approximately 3.1415926544231341, Error is 0.0000000008333410
Enter the number of intervals: (0 quits) 0
```

Figure: Sequential Code Output.

Pi Computation IV

- Parallel Code:
  - The master process reads number of intervals from standard input, this number is then sent to the processes.
  - Having received the number of intervals, each process evaluates the total area of **n**/**size** rectangles under the curve.
  - The contributions to the total area under the curve are collected from participating processes by the master process, which then adds them up, and prints the result on standard output.

Pi Computation V

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

int main(int argc, char* argv[])
{
    int done = 0, n, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, h, sum, x;
    int size, rank, me;
    int tag=11;
    MPI_Status  status;
    double mysum;
    double pi;

    MPI_Init(&argc, &argv);                /* Initialize MPI */
    MPI_Comm_size(MPI_COMM_WORLD, &size);/* Get number of processes */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);/* Get own identifier */

    while (!done)
    {
        if (rank == 0) { /* Process 0 does this */
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
            /* Send a message containing number of intervals to all \
                                                   other processes */
            for (i=1; i<size; i++) {
                MPI_Send(&n, 1, MPI_INT, i, tag, MPI_COMM_WORLD); \
                                                  /* Blocking send */
            }
```

Pi Computation VI

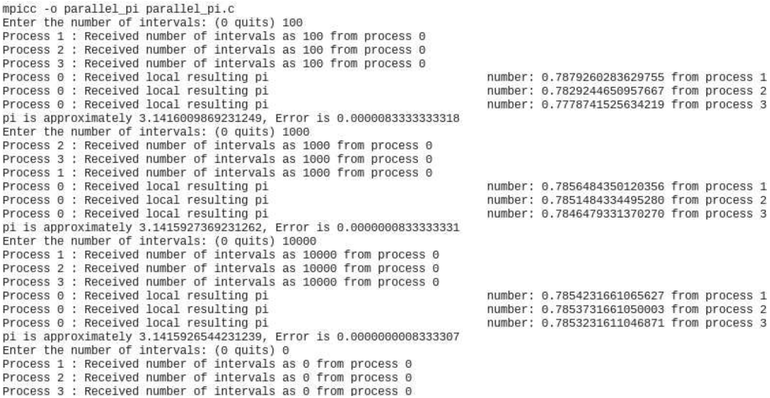
```
            if (n == 0) break; /* Quit when "0" entered */
            /* Computing local pi number for rank 0 process*/
            /* Integral limits are from 0 to 1 */
            h   = (1.0-0.0)/(double)n; /* Step length*/
            mysum = 0.0; /* Initialize sum variable */
            for (i = rank+1; i <= n; i += size) /* Loop over interval \
                                                   for integration */
            {
                x = h * ((double)i - 0.5);  /* Middle point at step */
                mysum += 4.0 / (1.0 + x*x); /* Sum up at each step */
//printf("i=%d x=%f sum=%f \n",i,x,sum); /* Intermediate steps */
            }
            mypi = h * mysum; /* Obtain local resulting pi number */
            /* Receive a message containing local resulting pi number \
                                          from all other processes */
            for (i=1; i<size; i++) {
                MPI_Recv (&pi, 1, MPI_DOUBLE, i, tag, MPI_COMM_WORLD, \
                                    &status); /* Blocking recieve */
                printf("Process 0 : Received local resulting pi \
                              number: %.16f from process %d \n",pi,i);
                mypi=mypi+pi; /* Reduce all local values to mypi \
                                                           variable */
            }
            printf("pi is approximately %.16f, Error is %.16f\n",mypi,  \
                                                 fabs(mypi - PI25DT));
        }
        else /* Other processes do this */
        {
            MPI_Recv (&n, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, \
                                    &status); /* Blocking recieve */
```

Pi Computation VII

```
            printf("Process %d : Received number of intervals as %d \
                                        from process 0 \n",rank, n);
            if (n == 0) break; /* Quit when "0" entered*/
            /* Computing local pi number for other processes*/
            /* Integral limits are from 0 to 1 */
            h   = (1.0-0.0)/(double)n; /* Step length*/
            mysum = 0.0; /* Initialize sum variable */
            for (i = rank+1; i <= n; i += size) /* Loop over interval
                                                   for integration */
            {
                x = h * ((double)i - 0.5); /* Middle point at step */
                mysum += 4.0 / (1.0 + x*x);  /* Sum up at each step */
//printf("i=%d x=%f sum=%f \n",i,x,sum); /* Intermediate steps */
            }
            mypi = h * mysum; /* Obtain local resulting pi number */
            /* Send a message containing local resulting pi number
                                              to master processes */
            MPI_Send(&mypi, 1, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD);
                                                 /* Blocking send */
        }
    }
    MPI_Finalize();
}
```

Pi Computation VIII

Parallelization
Application Example
Pi Computation



```
mpicc -o parallel_pi parallel_pi.c
Enter the number of intervals: (0 quits) 100
Process 1 : Received number of intervals as 100 from process 0
Process 2 : Received number of intervals as 100 from process 0
Process 3 : Received number of intervals as 100 from process 0
Process 0 : Received local resulting pi                                number: 0.7879260283629755 from process 1
Process 0 : Received local resulting pi                                number: 0.7829244650957667 from process 2
Process 0 : Received local resulting pi                                number: 0.7778741525634219 from process 3
pi is approximately 3.1416009869231249, Error is 0.0000083333333318
Enter the number of intervals: (0 quits) 1000
Process 2 : Received number of intervals as 1000 from process 0
Process 3 : Received number of intervals as 1000 from process 0
Process 1 : Received number of intervals as 1000 from process 0
Process 0 : Received local resulting pi                                number: 0.7856484350120356 from process 1
Process 0 : Received local resulting pi                                number: 0.7851484334495280 from process 2
Process 0 : Received local resulting pi                                number: 0.7846479331370270 from process 3
pi is approximately 3.1415927369231262, Error is 0.0000000833333331
Enter the number of intervals: (0 quits) 10000
Process 1 : Received number of intervals as 10000 from process 0
Process 2 : Received number of intervals as 10000 from process 0
Process 3 : Received number of intervals as 10000 from process 0
Process 0 : Received local resulting pi                                number: 0.7854231661065627 from process 1
Process 0 : Received local resulting pi                                number: 0.7853731661050003 from process 2
Process 0 : Received local resulting pi                                number: 0.7853231611046871 from process 3
pi is approximately 3.1415926544231239, Error is 0.0000000008333307
Enter the number of intervals: (0 quits) 0
Process 1 : Received number of intervals as 0 from process 0
Process 2 : Received number of intervals as 0 from process 0
Process 3 : Received number of intervals as 0 from process 0
```

Figure: Parallel Code Output.