#include "mpi.h"

#include <stdio.h>

#include <math.h>

int main( int argc, char \*argv[])

{

int n, i;

double PI25DT = 3.141592653589793238462643;

double pi, h, sum, x;

int numprocs, myid;

double startTime, endTime;

/\* Initialize MPI and get number of processes and my number or rank\*/

MPI\_Init(&argc,&argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&numprocs); //used to find an individual process's rank in a communicator

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&myid); //used to find the number of processes in a communicator

/\* Processor zero sets the number of intervals and starts its clock\*/

if (myid==0)

{

n=numprocs\*100000000;

startTime=MPI\_Wtime();

for(int i = 1; i < numprocs ; i++){

MPI\_Send(&n, 1, MPI\_INT, i, myid, MPI\_COMM\_WORLD);

}

}

else {

MPI\_Recv(&n, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

/\* Broadcast number of intervals to all processes \*/

//MPI\_Bcast(&n, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

/\* Calculate the width of intervals \*/

h = 1.0 / (double) n;

/\* Initialize sum \*/

sum = 0.0;

/\* Step over each inteval I own \*/

for (i = myid+1; i <= n; i += numprocs)

{

/\* Calculate midpoint of interval \*/

x = h \* ((double)i - 0.5);

/\* Add rectangle's area = height\*width = f(x)\*h \*/

sum += (4.0/(1.0+x\*x))\*h;

}

/\* Get sum total on processor zero \*/

// MPI\_Reduce(&sum,&pi,1,MPI\_DOUBLE,MPI\_SUM,0,MPI\_COMM\_WORLD);

if (myid==0)

{

double local\_sum;

for(int i = 1; i < numprocs ; i++){

MPI\_Recv(&local\_sum, 1, MPI\_DOUBLE, i, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

pi += local\_sum;

}

pi += sum;

}

else {

MPI\_Send(&sum, 1, MPI\_DOUBLE, 0, 0, MPI\_COMM\_WORLD);

}

/\* Print approximate value of pi and runtime\*/

if (myid==0)

{

printf("pi is approximately %.16f, Error is %e\n", pi, fabs(pi - PI25DT));

endTime=MPI\_Wtime();

printf("runtime is=%.16f",endTime-startTime);

}

MPI\_Finalize();

return 0;

}

**Result**

I ran the program 3 times and used the median to get the value used for calculating speedup and scaled efficiency.

**Discussion**

The results of the original code are slightly slower than the modified one after 8 processors although I’m unsure as I only ran it 3 times and took the median. Having more runs would probably confirm whether this is true or not. The other results seem pretty consistent and close to each other.