Homework 02: Exercise 3

Bernhard Sehorz

February 15, 2005

Problem: Finding π using MPI collective operations

This exercise presents a simple program to determine the value of π. The algorithm suggested here is chosen for its simplicity. The method evaluates the integral of $\frac{4}{1+x^2}$ between 0 and 1. The method is simple: The integral is approximated by a sum of n intervals; the approximation to the integral in each interval is $\frac{1}{n}\frac{4}{1+x^2}$. The master process (rank 0) asks the user for the number of intervals; the master should then broadcast this number to all of the other processes. Each process then adds up every n'th interval ($x = \frac{rank}{n}, \frac{rank}{n} + \frac{size}{n}, \frac{rank}{n} + 2\frac{size}{n} \ldots$). Finally, the sums computed by each process are added together using a reduction.

Solution: First of all we note that the integral boundaries have to be 0 and 1. Since arctan is an antiderivative of the integrand, we need

$$4 \arctan x|_u^v = 4(\arctan v - \arctan u) = \pi$$

which doesn't hold for the originial boundaries $v = \frac{1}{2}$ and $u = -\frac{1}{2}$ given in the assignment. The equality holds with $v = 1$ and $u = 0$, however, since

$$4(\arctan 1 - \arctan 0) = 4\frac{\pi}{4} = \pi.$$

Furthermore, the method of approximating the integral is sketched in figure 1 for a division of $[0, 1]$ in 3 subintervalls $\left[0, \frac{1}{3}\right]$, $\left[\frac{1}{3}, \frac{2}{3}\right]$ and $\left[\frac{2}{3}, 1\right]$. From the sketch one can see that for the approximation, the function value at the endpoint of the respective interval is used. Another possible alternative would be to take the respective startpoint.

To illustrate which process approximates which subintegral, lets assume that two processes compute the integral. In this case, the he first process would approximate the first and the last subinterval by $\frac{1}{3}\frac{4}{1+(1/9)}$ and $\frac{1}{3}\frac{4}{1+1}$ respectively, whereas the second process would approximate the middle subinterval by $\frac{1}{3}\frac{4}{1+(4/9)}$. In these expressions, the function arguments are of the form given in the assignment. Now it should be clear how the work is distributed among processes.

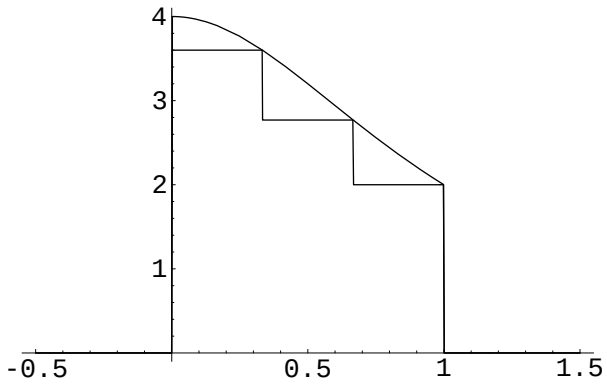

Figure 1: Integration algorithm

The description of the implementation follows as comments in the code. A remark of the computation of the inner points is indicated. If we would follow the algorithm given above exactly, the statement would read `x=h*(double)i;`. This literal implementation also works but the solutions it produces converge much slower to π than in this implementation. (One can see the reason for this on fig. 1: Taking the upper bound of a subinterval as interior mesh point, the function is not "fitted" well.) Instead of taking 0 and 1 as integral boundaries, we use $-\frac{h}{2}$ and $1-\frac{h}{2}$. For $h \to 0$ (that means for large values of n) the boundaries converge to the original. This adaption of the integral bounds makes the simple quadrature fit the function better. (I guess that this is the reason that the tutorial's authors gave $-\frac{1}{2}$ and $\frac{1}{2}$ as integral bounds.)

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

int main(argc, argv)
int argc;
char *argv[];
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264 3;
    double mypi, pi, h, sum, x;

    /*The standard MPI initialization*/
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    while (!done)
    {
```

```
      /*The main process asks the user for some input*/
      if (myid == 0)
      {
      printf("Enter the number of intervals: (0 quits) ");
      scanf("%d",&n);
      }

      /*We use the MPI broadcast function to send the number
      of subintervals used to all processes*/
      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

      if (n == 0) break; /*The user wants to quit ...*/

   /*We determine the length of a subinterval*/
   h   = 1.0 / (double) n;
   sum = 0.0;

   /*This loop is the important part of the program.*/
   for (i = myid + 1; i <= n; i += numprocs)
   {
   /*First we compute the current interior point...*/
   x = h * ((double)i - 0.5);
   /*... and then add it to the former computed
   subinterval approximations*/
   sum += 4.0 / (1.0 + x*x);
   }

   /*mypi is the part of pi that is computed by the
   current process*/
   mypi = h * sum;

   /*We now bring the local results of all processes together
   forming the approximated value of pi*/
   MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

   /*The main process displays the result and compares
   it with the"true" value of pi*/
   if (myid == 0)
   printf("pi is approximately %.16f, Error is %.16f\n",
   pi, fabs(pi - PI25DT));
}

/*exit program*/
MPI_Finalize();
return 0;
}
```