CSCI 3366 (Parallel and Distributed Processing), Fall 2017

Homework 1

Credit: 15 points.

1 Reading

Be sure you have read, or at least skimmed, the first few sections of the updated Appendix A (before writing your OpenMP program) and the first few sections of the updated Appendix B (before writing your MPI program).

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", plus one or more of the following about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For written assignments, it should go right after your name and the assignment number; for programming assignments, it should go in comments at the start of your program(s).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

3 Overview

First a general comment: For this assignment, please *do not* discuss the problem or possible solutions with each other or anyone else. I want you to discover any potential pitfalls yourselves!

In class we briefly discussed approximating the value of π by "throwing darts" at a square of side 2 enclosing a circle of radius 1, counting how many darts fall within the circle, and then

[1]Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

dividing that by the total number of darts to get the ratio between the area of the circle (π) and the area of the square (4). For this assignment, your mission is to write, for two different programming environments (OpenMP and MPI), a parallel program that performs this calculation. (The next assignment will add additional programming environments.)

4 Details

4.1 Sequential starter programs

To get you started, I have written a sequential program in C that performs the desired calculation and prints appropriate results:

- C program: monte-carlo-pi.c[2]. Also requires timer.h[3].

Start by downloading this code, compiling it, and running it a few times to get a sense of what inputs you need to get a good approximation of π. (Notice that you will need the flag `-std=c99` or `-std=c11` to compile the C code.)

4.2 Parallel programs

(10 points)

The programs you write (one each using OpenMP and MPI for this assignment, additional environments in the next assignment) should accept the same command-line input and produce the same output as my sequential program, except that:

- The OpenMP program should get the number of threads to use from environment variable `OMP_NUM_THREADS`.
- The MPI Program should get the number of processes in the usual way MPI programs do (parameter to `mpirun`).
- Rather than printing "sequential C program results",
  - the OpenMP program should print "parallel OpenMP program results with N threads" (where N is replaced with the number of threads);
  - the MPI program should print "parallel MPI program results with N processes" (where N is replaced with the number of processes); and

  (You can make other changes to the output format if you really want to, but be sure your code prints the same information mine does.)

You can also make any changes you like to how the programs work internally.

It's up to you how you choose to parallelize the sequential code, but notice that in many respects the calculation here strongly resembles the one in the numerical integration example, so the approaches we used for that example might work well here too. The only thing that's tricky is deciding what

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3366_2017fall/Homeworks/HW01/Problems/monte-carlo-pi.c`

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS3366_2017fall/Homeworks/HW01/Problems/timer.h`

to do about the random numbers (should all the processes/threads generate the same sequence of random numbers? should they generate different ones? if so, how?). For this assignment, I want you to make your best guess about what would be reasonable, implement that, and see how it works. After everyone has turned something in, we'll discuss in class your results and possible improvements.

You can make my grading job a bit easier by using the following names for your programs:

- `monte-carlo-pi-openmp.c` for the OpenMP program.
- `monte-carlo-pi-mpi.c` for the MPI program.

4.3 Performance experiments

After you get each program working, you should try running it repeatedly, varying the number of threads or processes, to see whether more threads/processes really do help. Good machines to use for this purpose may be `Dione` for OpenMP (more processing elements than other machines) and the `Pandora` cluster (names `Pandora01` through `Pandora08`) for MPI (Linux-only so shouldn't be rebooted out from under you).

4.4 Discussion of results

(5 points)

In addition to turning in your source code, briefly answer the following questions about each of your programs:

- Does the program seem to generally give better results as the number of samples increases? Can you say anything interesting or useful about how the seed affects the quality of the results?
- For the same number of samples and seed, does the program give the same results no matter how many processes or threads you use? If not, why do you think it doesn't?
- Does parallelism seem to help the program's performance? You will probably need a fairly large number of samples to observe any benefit, but up to a point more processes or threads should give faster execution. Is this true for your program? Support this with some observations (output of your program showing number of samples, number of processes/threads, etc., plus the name(s) of the machine(s) used). If it's not true, what do you think is going wrong?

Keep in mind that this assignment is a first pass at producing good programs for this problem. Your programs should compile and produce output that's more or less reasonable, but you will have another chance (in Homework 2) to produce something that really works well, so it's okay this time if the output seems slightly off, or the performance is disappointing.

4.5 Hints and tips

- Feel free to borrow code from any of the sample programs linked from the course sample programs page[4]. This page also contains links to my writeups about compiling and running

[4]`http://www.cs.trinity.edu/~bmassing/Classes/CS3366_2017fall/SamplePrograms/index.html`

programs on the lab machines. The course "useful links" page[5] has pointers to documentation on OpenMP and MPI.

- You can develop your programs on any system that provides the needed functionality, but I will test them on the department's Linux classroom/lab machines, so you should probably make sure they work in that environment before turning them in.

5 What to turn in and how

Submit your program source code and discussion of results by sending mail to `bmassing@cs.trinity.edu`. Send program source as attachments. You can put your discussion of results in the body of the message or attach a file in any format readable on our Linux lab machines (plain text, PDF, something openable with OpenOffice, etc.). Please use a subject line that mentions the course number or name and the assignment (e.g., "csci 3366 homework 1" or "parallel hw1").

[5]`http://www.cs.trinity.edu/~bmassing/Classes/CS3366_2017fall/HTML/links.html`