This is a continuation of the prior MPI/C assignment for performing integration, however, this time, instead of using MPI/C, you will be using Pthreads/C. You will keep (or improve if necessary any of the prior code with MPI), and add to the project a new file “pth-mc-integration-parallel.c”. This file will be the pthread version of the parallel MPI integration program from the last project.

You will add the compilation of this execution to your Makefile. The makefile will compile all the prior executables from the prior project, and this new addition as well.

The behavior of this new code will be as follows:

./pth-mc-integration-parallel <num\_threads> <a> <b> <n>

The new parameter is <num\_threads>, just like in the matrix-vector multiply examples and also the calculate pi examples from the slides / videos.

You will use the timer.h to get the timing data, just as before.

It will produce the following output at the console:

a b n approximation\_of\_integration exec\_time p

where p here is the num\_threads. You’ll then test on your own machine, and also over on COMET just as before, and in accordance with the same specifications found later in this document.

**Technical note**. I want you to make use of a mutex lock to control access to the critical section represented by the global sum. Also, I want you to place the critical section in a location that will minimize the contention to the lock. After watching the prior videos, you’ll know what I’m taking about.

Looking at some of the reports from MC06, (but I haven’t graded them yet), I’m seeing some data that looks like I would expect it to. For example, look at the performance as the problem size increases (*taken from Mr. Catoe’s report* – *good job by the way!*):

**A close up of a map

Description automatically generatedA close up of a map

Description automatically generatedA close up of a map

Description automatically generated**

(see how the actual speedup approaches the ideal speedup as the problem size increases).

If you’re not seeing something at least “similar” to the above, in terms of trends, then you need to look more deeply at what is going on with the application most likely. Also, if you have mistakes in the prior assignment, or you did something wrong, you should attempt to figure that out, and work on it, and bring any new knowledge to bear on this assignment.

Specification in prior MPI/C version of the assignment:

In this assignment, you will use the Monte Carlo Method perform the following calculation:

The function f(x) will be defined at the top of your program as follows:

double f(double x) {

return x\*x; // we should be able to change this

}

You will use drand48() to provide your pseudorandom number sequence. You will seed it properly.

The CLI (commandline interface) for the sequential code will be as follows:

./mc-integration-serial <a> <b> <n>

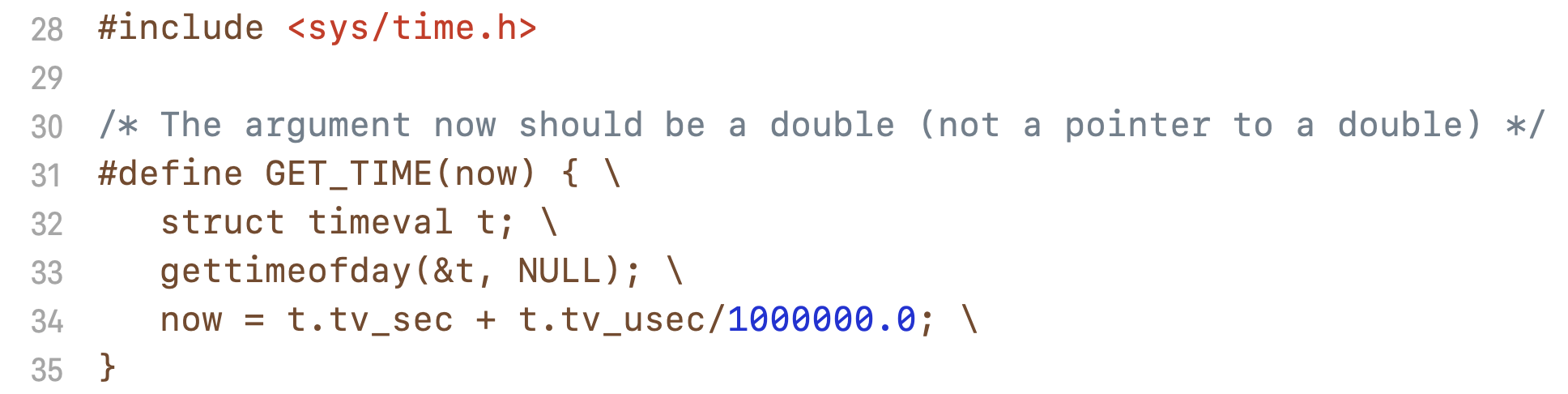
where a and b are the bounds of integration above, and n is the total number of (xi,yi) points generated.

Running this command will report to the console the a single line:

a b n approximation\_of\_integration exec\_time

Where exec\_time is the execution time of the program in seconds. Use the linked timer.h

file that I have provided in Moodle. Here’s the basic idea:



Just include the timer.h file, and call it from your program like this:

double start\_time, end\_time, elapsed\_time;

GET\_TIME(start\_time);

// code you want to time

GET\_TIME(end\_time);

elapsed\_time = end\_time – start\_time;

*(as a side note, it is an interesting endeavor to see how this macro works)*

You will use this utility to observe the behavior of using larger and larger values of n. Exact Details TBD in class.

You will then develop a parallel version using MPI that will adhere to the following:

mpirun -np <p> ./mc-integration-parallel <a> <b> <n>

to do the same thing, but dividing the workload among the <p> processes. Only the main process should produce any output, and the output it produces should be the same as before, except with <p> also written out.

a b n approximation\_of\_integration exec\_time p

You will then use COMET / XSEDE to test out the performance of this code, for 3 different sizes of n, and for a number of processes that goes from 1, 2, 4, 8, 16, and finally 24. I want tables and plots of the data for the time, speedup and efficiency.

Make sure that the X axis has 1, 2, 4, 8, 16 and 24 spread out linearly (i.e. the space between the tick marks for 4 and 8 should be twice as much as the space between the 2 and 4 tick marks. Also, 24 should be spaced out appropriately. In other words, I want both the X and Y axis to be linear. I noticed that some folks didn’t have this back in their reports / plots at the beginning of the course.

You will include all this in your **report**, and also your report will discuss Monte Carlo, what you did, how it works, and also a comparison for this numerical integration technique to an analytical solution for the same f(x)’s across the same intervals. And demonstrate how much error there is between the two, for 2 to 3 different values of n.

**Submission to Moodle.**

Tar GZ the entire project directory (while being in the parent folder of the folder you’re trying to compress) up using:

**tar -czvf your\_file\_name.tar.gz ./the\_folder\_to\_compress**

Submit to moodle.