ECE 2036 Spring 2017

Lab 7: Multithreaded Numerical Integration

Assigned: April 7, 2017 *Due: 11:59 PM April 21, 2017*

In this lab we will be writing code which allows the user to input a polynomial function of x of arbitrary order. The code will calculate the integral of that polynomial between $x = 0$ and $x = 100$ using several techniques. First, you will use your knowledge of calculus to write code to analytically evaluate the integral. Then, you will write code to numerically calculate the integral for various numbers of grid points (subdivisions). Finally, you will write code to farm out the numerical integration to 10 different cores on the deepthought cluster.

Numerical Integration (Quadrature)

Numerical integration is a powerful technique for integrating any function. In this case, we happen to be numerically integrating a polynomial that could be analytically integrated (integrated using pencil and paper to get a closed-form solution). But the numerical integration code you write will also be capable of integrating functions that are impossible to integrate analytically.

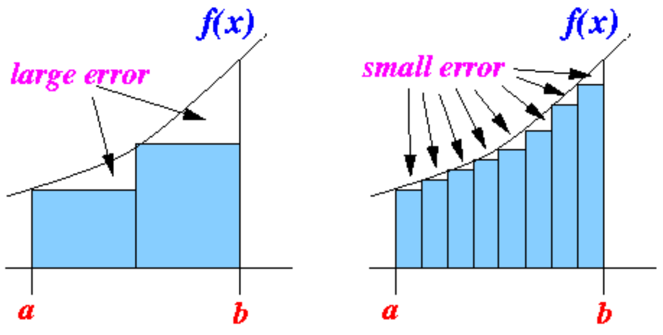

Fig. 1. Illustration of the use of the 'rectangle rule' for numerical integration. The sum of the areas of the rectangles will approximately equal the integral of the function. A larger number of thinner rectangles will yield a better approximation.

To understand numerical integration, we must recall the definition of an integral: it is just the area underneath a function. We will use the simplest numerical approach to (approximately) calculate that area; namely, breaking up the area into rectangles, as shown in Fig 1. The area of each rectangle is easy to find, just height times width. As shown in Fig. 2, the width of each rectangle is δx, which is called the grid spacing. The height of each rectangle is the function evaluated at one grid point, $f(x_i)$, where x_i is a grid point. To numerically calculate the integral, we simply sum up the value of the function at all of the grid points times δx:

$$\int_a^b f(x)dx \approx \delta x \sum_i f(x_i)$$

As shown in Fig. 1, in general a larger number of (thinner) rectangles will result in a better approximation of the integral.

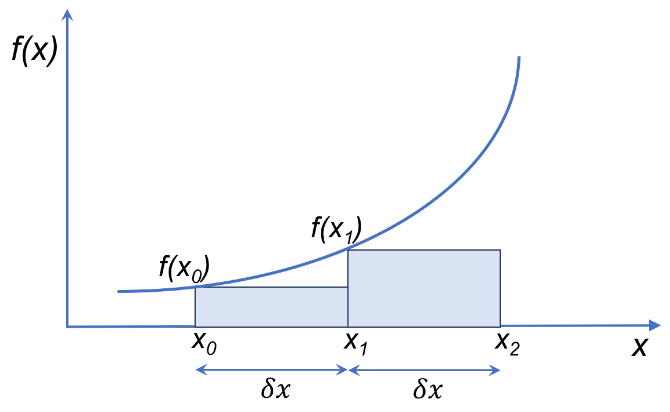

Fig. 2. Illustrating the meaning of the grid points and grid spacing.

Multithreading

As you know, most modern computer microprocessors have multiple cores. (It has proven more practical to increase the number of cores rather than increase the clock speed further, which causes excess heating.) The deepthought cluster is composed of many microprocessors linked together, each of which has many cores. This incredible parallel computing power does not automatically accrue benefits to our programs, however. In order to exploit the parallel processing capabilities of a cluster like deepthought, we must write code that splits up its tasks into multiple threads, each of which can be performed on a separate core. Some problems are more amenable to multithreading, and others are less so. Numerical integration happens to benefit greatly from multithreading, as any integral can be split into a sum of sub-integrals over smaller ranges:

$$\int_0^{20} f(x)dx = \int_0^{10} f(x)dx + \int_{10}^{20} f(x)dx$$

Essentially, we will send each sub-integral to a different core to be evaluated, then sum those results to get the total result.

Lab Programming Instructions

You must download three files from t-square: integration.cc, gthread.cc, and gthread.h. You should ONLY modify integration.cc; leave gthread.cc and gthread.h unchanged. The gthread files were written by Dr. George Riley to provide easy-to-use multithreading functionality, and their use will be described in class. For the purposes of this lab, you will need to use three functions provided by gthread: `CreateThread`, `EndThread`, and `WaitAllThreads`.

You should transfer all of the above files to a subdirectory named 'Lab7' which you will create in your home directory on deepthought. Your code must run on deepthought. The correct command to use when compiling this code on deepthought is as follows:

```
g++ -std=c++0x integration.cc gthread.cc -lpthread
```

You must add code to the `integration.cc` file. The code you should add is described in extensive comments in that file. Please see the file for more details.

Numerical Notes

Numerical calculations have many peculiarities. Here I will describe two that you need to pay attention to for this lab.

It is problematic to add a very small number to a very large number on a computer (or subtract two very large numbers that have a tiny difference). The reason is that the computer only stores a finite number of significant digits. Let's say the computer stores 8 digits. If I want to add 1 to 10^{10}, the result on the computer will be: 1.000000e10 + 1.0e0 = 1.0000000e10. The computer incorrectly decides that 10^{10} is unchanged by adding one, because it lacks enough digits to store the change.

This presents a challenge when we are trying to calculate our grid points x_i, because the difference between them (δx) can be very tiny. To mitigate this issue, we should use the following equation to calculate each x_i:

$$x_i = i * \delta x + x_{min}$$

Where x_{min} is the lower bound of the integral.

Another consideration is error. We can define the error as the difference between the exact result (obtained analytically) and the numerical result. However, the error will never go to zero, so we need to be able to judge how much error is acceptable. If our calculated error is 1000, that sounds bad; but in fact it might be very good if the value of the integral is 10^{15}. What is important, then, is relative error, which is the difference between the exact result and the numerical result divided by the exact result:

$$\text{relative error} = \frac{\int f(x)dx\ (\text{exact}) - \int f(x)dx\ (\text{numerical})}{\int f(x)dx\ (\text{exact})}$$

Multiplying this by 100 yields the percent relative error.

Turn-in Procedures

You must submit your source code on the deepthought cluster by using the turnin script, as you did for Lab 0. You MUST ensure that your code compiles and runs properly on deepthought.

Depending on your section, from your home directory enter one of the following at the command prompt:

```
turnin-ece2036a Lab7
```

or

```
turnin-ece2036b Lab7
```

This automatically copies everything in your Lab7 directory to a place that we can access (and grade) it.

Grading Rubric

GRADING POINT DEDUCTIONS

Element	**Percentage Deduction**	**Details**
Does not compile	30%	Does not compile on deepthought.
Does not correctly calculate the analytic integral	20%	Self-explanatory
Does not correctly calculate the numerical integral	20%	Does not achieve a relative error of less than 10^{-6} for a typical cubic polynomial
Does not correctly use threads to calculate the numerical integral	20%	Does not show a substantial decrease in execution time using 10 threads while calculating the same result
Does not use vector and/or array properly	10%	Does not use vector and/or array as indicated in comments

LATE POLICY

Element	**Percentage Deduction**	**Details**
First Day	20%	By Monday after the due date
Each Additional Day	20% per day	The weekend (Sat/Sun) will count as one day

Sample output from completed code

```
-bash-4.1$ ./a.out

Welcome to the numerical integrator.

Please enter the coefficient of the 0th-degree term of your
polynomial: 12.3

Please enter the coefficient of the term of degree 1, or 'x' to
stop: 1.9

Please enter the coefficient of the term of degree 2, or 'x' to
stop: -6.7

Please enter the coefficient of the term of degree 3, or 'x' to
stop: 0.5

Please enter the coefficient of the term of degree 4, or 'x' to
stop: x

The coefficients you entered were: 12.3     1.9     -6.7     0.5

The analytically calculated integral of your polynomial is:
1.02774e+07

The relative error of the numerical integration with 1000 points
is: -0.0021086

The relative error of the numerical integration with 100000000
points is: -2.10749e-08

The computation time with one thread was: 20 seconds.

The relative error of the numerical integration with 100000000
points and 10 threads is: -2.10749e-08

The computation time with 10 threads was: 2 seconds.
```