

Homework 1: A Parallel Monte Carlo Simulation for Black-Scholes Option Valuation

UNIVERSITY OF CALIFORNIA
Computer Science Division
Prof. Kathy Yelick
CS194

Due Sept 11, 2007 at 5pm

1 Introduction

In this assignment you will parallelize a Monte Carlo simulation of the Black-Scholes Option Valuation model. The goals of this assignment are to introduce

1. Multithreaded programming;
2. Monte Carlo simulations; and
3. Benchmarking for performance and accuracy.

We use the Black-Scholes model as a vehicle to teach these concepts. No prior knowledge of finance is necessary to successfully complete the assignment. If you are interested in delving deeper into the topics discussed, Wikipedia has good articles and links on these topics.

2 Financial options

In this assignment you will be writing a program to compute the behavior of the options market to determine reasonable prices. You do not need to understand the details of how this works, since we will be giving you code for the basic calculation. But to understand what you will be computing, consider a scenario in which I call you today with the following offer:

“In 3 months’ time you will have the option to purchase Microsoft Corp. shares from me at a price of \$25 per share.”

The key point is that you have the *option* to buy the shares. Three months from now, you may check the market price and decide whether or not to exercise the option. (In practice,

you would exercise the option if and only if the market price were greater than \$25, in which case you could immediately re-sell for an instant profit.) This deal has no downside for you – three months from now you either make a profit or walk away unscathed. I, on the other hand, have no potential gain and an unlimited potential loss. To compensate, there will be a cost for you to enter into the option contract. You must pay me some money up front.

The *option valuation problem* is to compute a fair value for the option [2]. More precisely, it is to compute a fair value at which the option may be bought and sold on an open market. The option described above is a *European call*. The Microsoft shares are an example of an *asset* - a financial quantity with a given current value but an uncertain future value. Formalizing the idea and introducing some notation, we have:

Definition: A *European call option* gives its holder the opportunity to purchase from the writer an *asset* at an agreed *expiry time* T at an agreed *exercise price* E . Given a time t , we will let $S(t)$ denote the asset value at time t , so $S(T)$ is the value of the asset at the expiry time. The final payoff to the purchaser is $\max\{S(T) - E, 0\}$, because

- if $S(T) > E$, the option will be exercised for a profit of $S(T) - E$, whereas
- if $S(T) \leq E$, the option will not be exercised.

In 1973, Robert C. Merton published a paper presenting a mathematical model which can be used to calculate a rational price for trading options [3]. (He later won a Nobel prize for his work.) In that same year, options were first traded in the open market. Since then, the demand for option contracts has grown to the point that trading options typically far outstrips that for the underlying assets. Merton's work expanded on that of two other researchers, Fischer Black and Myron Scholes (see [1]), and the pricing model became known as the *Black-Scholes* model. The model depends on a constant σ (a Greek letter, pronounced "sigma") representing how volatile the market is for the given asset, as well as the continuously compounded interest rate r .

3 Monte Carlo methods

Computers are often used to predict the behavior of physical systems such as the airflow around a new automobile or airplane design, the effect of an earthquake on a bridge or building, or the behavior of financial markets under specified conditions. One class of algorithms for such simulations are called *Monte Carlo* methods¹, and are distinguished from other methods in their use of random numbers² to select a set of points at which to evaluate a function. The Monte Carlo method of calculating π that was described in lecture and discussion section selects a set of random set of points in a unit square, and counts the fraction of those points that are inside a quadrant of the unit circle. It then uses this ratio and the known formula for the area of a circle to estimate π .

¹http://en.wikipedia.org/wiki/Monte_Carlo_method

²In practice, these are not actually random, but *pseudorandom*. Please refer to the tutorial on the class website for more information.

Monte Carlo algorithms are often used to find solutions to mathematical problems that cannot easily be solved by other means and are relatively easy to program on parallel machines, because each processor can evaluate the function independently on a subset of the points.

4 Sequential algorithm

In this assignment you will be using the Monte Carlo technique to calculate the Black-Scholes pricing model. It will take as input a number of trials M . As with any Monte Carlo calculation, a higher value will give us a more accurate answer, but take more time. Pseudocode for a sequential algorithm for this problem is given below. In addition to M , the pseudocode refers to the input variables described in Section 2. They are summarized here for your convenience.

- S : asset value function
- E : exercise price
- r : continuously compounded interest rate
- σ : volatility of the asset³
- T : expiry time
- M : number of trials

The pseudocode also uses several internal variables:

- `trials`: array of size M , each element of which is an independent trial (iteration of the Black-Scholes Monte Carlo method)
- `mean`: arithmetic mean of the M entries in the `trials` array
- `randomNumber()`, when called, returns successive (pseudo)random numbers chosen from a Gaussian distribution. NOTE: this function will be provided for you. See Section 5.2 for more details.
- `mean(a)` computes the arithmetic mean of the values in an array `a`
- `stddev(a, mu)` computes the standard deviation of the values in an array `a` whose arithmetic mean is `mu`.
- `confwidth`: width of confidence interval
- `confmin`: lower bound of confidence interval
- `confmax`: upper bound of confidence interval

³Do not confuse this with the standard deviation of the trials, even though the Greek letter σ is often used to denote standard deviation.

Algorithm 1 A sequential Monte Carlo simulation of the Black-Scholes model

```
1: for  $i = 0$  to  $M - 1$  do
2:    $t := S \cdot \exp\left(\left(r - \frac{1}{2}\sigma^2\right) \cdot T + \sigma\sqrt{T} \cdot \text{randomNumber}()\right)$   $\triangleright t$  is a temporary value
3:   trials [  $i$  ] :=  $\exp(-r \cdot T) \cdot \max\{t - E, 0\}$ 
4: end for
5: mean := mean( trials )
6: stddev := stddev( trials , mean)
7: confwidth :=  $1.96 \cdot \text{stddev} / \sqrt{M}$   $\triangleright$  Treat 1.96 as a magic number; you don't have to
   understand why it's there.
8: confmin := mean - confwidth
9: confmax := mean + confwidth
```

5 Parallel programming basics

This section provides links to summary and reference information about parallel programming. Completing this assignment successfully will require a good understanding of threads and related synchronization mechanisms, such as locks. The following sections provide more details.

5.1 Parallel programming with POSIX threads

Slides from Lecture 2 and the online tutorial from Lawrence Livermore National Laboratory⁴ provide a very good overview of using threads. The lecture list on the course web page has links to both of these.

We will use the POSIX Threads (Pthreads) API for this assignment. Pthreads are very much like the threads that were discussed in CS162. An overview of locks, mutexes, semaphores, and condition variables can also be found in prior CS162 semester notes. The online tutorial walks the reader through the basics and provides examples that should be sufficient for completing this assignment.

5.2 Parallel random number generation

The Monte Carlo method depends on having a high-quality pseudorandom number generator (PRNG). Imagine, for example, that in the π program example, that all of the random points (darts on our boards) end up in a narrow area of the unit square. Then our calculation of π would not be accurate. Computers cannot generate truly random numbers, but can produce a stream of “pseudorandom” numbers that behave “random enough” for our purposes. Parallelism adds further complications, as the random number generator function must behavior correctly when two or more threads call it simultaneously, i.e., it must be *thread-safe*. For further information about PRNG's, please refer to the tutorial on the course website.

⁴See <http://www.llnl.gov/computing/tutorials/parallel.comp/>.

5.2.1 Thread-safe PRNG API

The interface to the thread-safe pseudorandom number generator can be found in `random.h` and `gaussian.h`. First, before spawning any threads, call `init_prng()` with a seed argument – either the same seed each time (for debugging) or a random seed generated by `random_seed()` (in `random.h`). Then, in each thread, call `spawn_prng_stream()` with the ID number of that thread, and use the returned opaque object as the `f_state` parameter of `gaussrand1()` (in `gaussian.h`). The function pointer parameter of `gaussrand1()` should be `uniform_random_double`.

6 Assignment

This homework involves making changes to a working, sequential implementation of the Black-Scholes Monte Carlo method. You should start out by running the sequential code and getting an understanding for the flow of the program as well as what outputs you should expect for a given set of input parameters.

A working Makefile has been included that will build the program. Instructions on running it can be found within `main.c`. Note that there is an input file named `params` which passes the values needed to evaluate the Black-Scholes confidence intervals. These input variables are the same as the pseudocode: asset value function S ; exercise price E ; continuously compounded interest rate r ; volatility of the asset σ ; expiry time T ; and number of trials M . You will want to vary these values once you get a working parallel version of the code and see its effect on the runtime of your program.

The main functions that you will be editing can be found in `black_scholes.c`. These changes will require you to add threading constructs to ensure that the program can be run in parallel. It has to fork as many threads as specified by the command-line argument `nthreads`. None of the threads should be doing any duplicate work. For example, if $M = 100$ and `nthreads = 4`, each of the four threads must generate 25 values. Then, one thread can do the calculation of the confidence intervals. You'll want to handle the case when M is not evenly divisible by `nthreads`.

The output of the program consists of the input parameters, the time it took for `black_scholes()` to finish executing, and the lower/upper bounds of the Black-Scholes confidence intervals. These confidence intervals should be nearly the same in both sequential and parallel execution, assuming all input parameters except `nthreads` are the same. However, your execution time should be less in parallel. This may not be true if you did not choose a good parallel strategy or if the cost of thread creation overruns the need for parallelism in certain cases with a small number of trials.

6.1 Provided for you:

A working, serial implementation of the Black-Scholes Monte Carlo method has been provided for you. You should only have to modify some of the functions found within `black_scholes.c` and also learn to use the PRNG `gaussrand1` function successfully.

6.2 Hints:

The following are characteristics of a correctly working program:

- As the number of trials increase, the confidence interval becomes smaller.
- For a given number of trials, the time it takes to calculate the confidence interval should decrease as you increase the number of threads, up to a certain number of threads which often (but not always!) corresponds to the number of processors. Beyond that number,

the time starts increasing due to factors like thread creation and destruction overhead overwhelming potential parallelism, and possible saturation of memory bandwidth.

7 What to turn in

1. Turn in all source and header files, along with the Makefile and PBS batch scheduler script. To do this, connect to your homework 1 directory, type “make clean” and then “submit hw1.” (We are hoping the submit system will be fully functional by then; we will provide more detailed instructions as the deadline approaches.) We should be able to make your program simply by typing make, and we should be able to run your program without having to add any additional files. Any command-line arguments that we need to supply should be clearly documented.⁵
2. Comment your code, as your grade does not depend solely on whether it runs in parallel. We will be looking at your parallel strategy as well as correctness in your use of threading constructs. Your comments will help us ensure that you get credit for the above.
3. You will also need to submit a report on some performance results from your completed assignment. This should include the following:
 - (a) Document the experimental setup: what type of processor? How many processors (cores) are available on this machine? Multicore or SMP? Did you run under a batch scheduler or interactively? What timer did you use? Did you compute its resolution (the smallest time interval that it usually measures on average)?
 - (b) Experiment with the Black-Scholes parameters to see if they affect the rate of trials per second that you can do. Do this for various numbers of threads.
 - (c) Experiment with the number of threads, and report the best speedup you see with your code (given a fixed number of trials M).
 - (d) Produce a speedup table or graph, showing the speedup for various numbers of threads created (from 1 to 8 at least).
 - (e) Explain why you believe the performance behaves as it does.
 - (f) Feel free to include any additional benchmarking or test results that you find interesting, as well as anything you think we should know about your code. Here is where you should document any special instructions for building and/or running your code.
4. Your written report should be in a file starting with the name **results** and with any of the following extensions:

⁵We recommend that you include a “test” target in the Makefile which tests the code for a common case, so that we can type “make test” to see if it runs.

- (a) `.pdf` (for PDF, such as you might produce using \LaTeX or by saving an OpenOffice or MS Word document as PDF);
 - (b) `.doc` (for Microsoft Word; we'd like to be able to open it in OpenOffice too, so test if it can be read in OpenOffice if you get the chance);
 - (c) `.txt` (for ASCII text; no special fonts or “smart quotes” please!).
5. We will grade your homework by reading your written report, running the code, and looking at selected parts of the code you have written.

8 Deadline

This homework assignment will be due on Tuesday, September 11th by 5:00 pm. Please use `submit` to turn in the assignment. The `submit` script timestamps your submission, so we know whether it was submitted before or after the deadline. You can submit as many times as you like, and we strongly encourage you to submit at least one version more than 24 hours before the deadline, to make sure your submission has the right files in it and everything is working. We will only grade the last submission, so your earlier one does not have to be completely finished.⁶

You have two “flex days” (48 hours) to use throughout the semester. You can use two flex days on any of the 4 homework assignments (6 due dates, since some will have two parts) if you need extra time to finish up. We will round up to the nearest hour past 5pm when calculating the the amount of flex time you have used. You only get 48 “flex hours” for the entire semester, so use them wisely! If you run out of flex hours, we will deduct significant late penalties, starting at 25% and increasing from there.

We expect this assignment to take you a few hours, but could take much more if you get “stuck.” The amount of programming (number of lines of code) is relatively small (a C coder who knows threads can finish it in a day; we know because we tried it!), but it will take a while to figure out our code and the interfaces for threads, to do the performance experiments, and write up your results.

Note that if everybody tries to finish the assignment an hour before the due date, the batch scheduler may not be able to fit all your jobs in. As a result, you should get the benchmarking done as early as possible.

References

- [1] F. BLACK AND M. S. SCHOLLES, *The pricing of options and corporate liabilities*, Journal of Political Economy, 81 (1973), pp. 637–654. Available at <http://ideas.repec.org/a/ucp/jpolec/v81y1973i3p637-54.html>.

⁶You may find this useful as a backup strategy!

- [2] D. J. HIGHAM, *Black-Scholes for scientific computing students*, Computing in Science and Engineering, 6 (2004), pp. 72–79.
- [3] R. C. MERTON, *Rational theory of option pricing*, Bell Journal of Economics and Management Science, (1973).