Monte Carlo Method for Stock Options Pricing

Sample User's Guide

Intel® SDK for OpenCL* Applications - Samples

Document Number: 329764-004US

Contents

Legal Information

Optimization Notice
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

About Monte Carlo Method for Stock Options Pricing

This sample demonstrates implementation of the Monte Carlo simulation for the European stock option pricing. Underlying algorithm is an OpenCL* kernel that unifies three major algorithm components:

- Mersenne twister - generation of uniformly distributed pseudorandom numbers
- Box-Muller transform - generation of normally distributed random numbers
- Option price calculation using Black-Scholes stock pricing model.

The exact Black-Scholes model is implemented as native code on the host for comparison with the results, generated with Monte Carlo.

Algorithm

Monte Carlo option model uses the **Monte Carlo method** to calculate the value of an option with multiple sources of uncertainty or complicated features. You can apply this technique to generate thousand random possible price paths for the underlying asset, and then to calculate the "payoff "– the associated exercise value of the option for each path. These payoffs are then averaged and adjusted to today numbers. This result is the value of the option. See http://en.wikipedia.org/wiki/Monte_Carlo_option_model for more information.

Mersenne Twister Random Number Generator

Monte Carlo option pricing model relies on random number generation for random price paths generation. This sample uses pseudorandom number generator – the Mersenne twister. The Mersenne twister is based on a matrix linear recurrence over a finite binary field, and provides fast generation of pseudorandom numbers, designed specifically to rectify many of the flaws found in older algorithms. For a *k*-bit word length, the Mersenne twister generates numbers with an almost uniform distribution in the range of [0, 2^k-1].

For comprehensive understanding of this pseudo-random number generator and algorithm details refer to http://en.wikipedia.org/wiki/Mersenne_twister. Intel OpenCL implementation of pseudorandom number generator is similar to pseudo code from the original paper. For more information see *Matsumoto, M.; Nishimura, T. (1998). "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator"*.

Box-Muller Transform

Monte Carlo option model requires sequence of normally distributed pseudorandom numbers for possible price paths generation. The Mersenne twister algorithm provides uniformly distributed numbers. The Box-Muller transform converts the uniformly distributed numbers to normally distributed numbers. The Box–Muller transform is a pseudo-random number sampling method for generating pairs of independent, standard, normally distributed (zero expectation, unit variance) random numbers, given a source of uniformly distributed random numbers. The basic form takes two samples from the uniform distribution on the interval of (0, 1] and maps them to two standard, normally distributed samples. Suppose U_1 and U_2 are independent random variables that are uniformly distributed in the interval of (0, 1]. Let

$$Z_0 = \sqrt{-2 * lnU_1} * \cos(2\pi U_2)$$

and

$$Z_1 = \sqrt{-2 * lnU_1} * \sin(2\pi U_2).$$

Then Z_0 and Z_1 are independent random variables with a standard normal distribution. For more information on the Box-Muller transform see http://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform.

Monte-Carlo Method and Black-Scholes Model

European options pricing has exact closed form solution described by the Black-Scholes formula. This sample uses exact Black-Scholes formula implementation in native C code on the host device to demonstrate correctness of the Monte Carlo method for options pricing. The Black–Scholes model or Black–Scholes–Merton is a mathematical model of a financial market containing certain derivative investment instruments. From the model, one can deduce the Black–Scholes formula, which gives the price of the European-style options. See http://en.wikipedia.org/wiki/Option_(finance) for details.

The partial differential equation, now called the Black–Scholes equation, governs the price of the option over time. The key idea behind the derivation is to hedge the option by buying and selling the underlying asset in just the right way and consequently "eliminate risk". This hedge is called “delta hedging” and is the basis of more complicated hedging strategies. The hedge implies only one right price for the option and it is given by the Black–Scholes formula. For more information on Black-Scholes method, see http://en.wikipedia.org/wiki/Black%E2%80%93Scholes.

Use the Monte-Carlo methods to estimate the price of an European option, and first consider the case of the ”usual” European Call, which is priced by the Black Scholes equation. At maturity, a call option is worth

$$c_T = \max(0, S_T - X),$$

where S_T is a stock price and X is an exercise price (strike price).

At the earlier date *t*, the option value is the expected present value of this

$$c_t = E[PV(\max(0, S_T - X))]$$

The important simplifying feature of option pricing is the “risk neutral result”, which enables treating the (suitably transformed) problem as the decision of a risk neutral decision maker, if you modify the expected return of the underlying asset such that this earns the risk free rate.

$$c_t = e^{-r(T-t)} E^*[\max(0, S_T - X)]$$

where $E^*[.]$ is a transformation of the original expectation and *r* is a risk free rate. One way to estimate the value of the call is to simulate a large number of sample values of S_T according to the assumed price process, and find the estimated call price as the average of the simulated values. By appealing to a law of large numbers, this average converges to the actual call value, where the rate of convergence depends on how many simulations you perform.

Lognormal variables are simulated as follows. Let $\tilde{x}$ be normally distributed with mean zero and variance one. If S_t follows a lognormal distribution, then the one-period-later price S_{t+1} is simulated as

$$S_{t+1} = S_t e^{\left(r-\frac{1}{2}\sigma^2\right)+\sigma\tilde{x}},$$

where $\sigma$is volatility. Or more generally, if the current time is *t* and terminal date is *T*, with a time between *t* and *T* of *(T-t)*,

$$S_T = S_t e^{\left(r-\frac{1}{2}\sigma^2\right)(T-t)+\sigma\sqrt{T-t}\tilde{x}}.$$

For the purposes of doing the Monte Carlo estimation of the price if an European call

$$c_t = e^{-r(T-t)} E^*[\max(0, S_T - X)]$$

You need to simulate the terminal price of the underlying, S_T, the price of the underlying at any time between *t* and *T* is not relevant for pricing.

The sample simulates lognormally distributed random variables, which gives us a set of observations of the terminal price S_T. If we let $S_{T,1}$, $S_{T,2}$, $S_{T,3}$, ... $S_{T,n}$ denote the *n* simulated values, estimate
$E[\max(0, S_T - X)]$ as the average of option payoffs at maturity, discounted at the risk free rate. [6]

$$\hat{c_t} = e^{-r(T-t)}\left(\sum_{i=1}^{n}\max(0, S_{T,i} - X)\right)$$

Analogously, put option price estimate is

$$\hat{p_t} = e^{-r(T-t)}\left(\sum_{i=1}^{n}\max(0, X - S_{T,i})\right)$$

OpenCL* Implementation

The described random number generation and option pricing algorithm are implemented as an OpenCL kernel.

Single OpenCL work-item calculates a pair of European option prices: put and call. The algorithm is implemented both for single and double precision floating point calculations. This feature is ruled by `--arithmetic` command-line option, which can be *float* (default) and *double* value. Global work size is defined by the `--options` command-line option. The default value is 65536. The OpenCL kernel contains inner loop over the Monte Carlo samples, ruled by `--samples` command-line option. The default samples number is 262144. Inner loop iteration encapsulates the Mersenne twister pseudorandom number generator, the Box Muller transform and the Black-Scholes stock options pricing model. The inner loop step is 2 because this is convenient for the Box Muller transform that operates with pair of uniformly distributed random numbers to generate pair of the normally distributed random numbers. Because of this atomic iteration of inner loop calculates pair of the Monte Carlo samples for final average. All underlying algorithms are interleaved together in the one inner loop to avoid the call overheads that can arise in the case of separate implementation of underlying algorithms as individual helper functions called from the main kernel. As a result, the inner loop contains some code duplications, but it helps to achieve performance gains.

The default work-group size for the OpenCL NDRange call is `NULL`, which means that it is up to the OpenCL runtime implementation to choose appropriate work-group number. You can try to adjust this parameter using the `--work_group_size` command-line option to get better performance. The work-group size of 16 work-items is optimal for the Intel® Xeon Phi™ coprocessor, as this size provides maximal granularity (or large enough work-group size to saturate computing facilities of the Intel Xeon Phi coprocessor) and enables an automatic 16-way vectorization done by the OpenCL runtime compiler. The sample application prompts you if the work group size choice is not valid.

Host part of the sample initializes three buffers, which contain randomly generated values for time to maturity, current stock price, and option strike price. Risk-free and volatility parameters are fixed. All this data is passed to the OpenCL kernel to calculate the resulting option put and call prices.

The `--validation` command-line option enables calculation of option prices on the host side using the Black and Scholes formula. Then these values are used for comparison with results calculated using the Monte Carlo simulation.

You can obtain the full list of sample command-line options by using the `--help` command-line option.

APIs Used

- `clGetPlatformIDs`
- `clGetPlatformInfo`
- `clGetDeviceIDs`
- `clGetDeviceInfo`
- `clCreateContext`
- `clCreateCommandQueue`
- `clCreateProgramWithSource`
- `clBuildProgram`
- `clGetProgramBuildInfo`
- `clCreateKernel`
- `clGetKernelWorkGroupInfo`
- `clCreateBuffer`
- `clSetKernelArg`
- `clEnqueueNDRangeKernel`
- `clEnqueueMapBuffer`
- `clEnqueueUnmapMemObject`
- `clFinish`
- `clReleaseMemObject`
- `clReleaseKernel`
- `clReleaseProgram`
- `clReleaseCommandQueue`
- `clReleaseContext`

Reference (Native) Implementation

Reference implementation is done in the `checkValidity` routine of the `montecarlo.cpp` file. This is single-threaded code that performs options price calculation according to Black and Scholes formula algorithm in native C as described in the “Algorithm” section.

Controlling the Sample

This sample is a command-line application. Use the following command-line options to control this sample:

Option	**Description**
`-h, --help`	Show this help text and exit.
`-p, --platform number-or-string`	Select platform, devices of which are used.
`-t, --type all | cpu | gpu | acc | default | <OpenCL constant for device type>`	Select the device by type on which the OpenCL kernel is executed.
`-d, --device number-or-string`	Select the device on which all stuff is executed.
`-o, --options <integer>`	Number of options to model.
`-g, --work-group-size <integer>`	OpenCL work group size. Set to 0 for work-group size auto selection.

-s, --samples <integer>	Number of Monte Carlo samples.
-i, --iterations <integer>	Number of kernel invocations. For each invoction, performance information will be printed.
-e, --threshold <float>	Validation threshold (if validation is enabled).
-a, --arithmetic float | double	Type of elements and all calculations.
--validation	Enables validation procedure on host (slow for big task sizes).

Choosing Sample Parameters

Algorithm is implemented both for single and double precision floating point calculations. This specific data type is selected by `--arithmetic` command-line option which can be float (default) and double value.

Global work size is defined by `--options` command-line option. Default value is 65536.

The kernel contains inner loop over Monte Carlo samples ruled by `--samples` command line option. Default samples number is 262144. Inner loop iteration encapsulates Mersenne twister pseudorandom number generator, Box Muller transform, and Monte Carlo sampling from Black and Scholes equation. All underlying algorithms are interleaved together in the one inner loop to avoid function call overheads.

The default work-group size for the OpenCL NDRange call is `NULL`, which means that it is up to the OpenCL runtime implementation to choose appropriate work-group number. You can try to adjust this parameter using the `--work_group_size` command-line option to get better performance. The work-group size of 16 work-items is optimal for the Intel® Xeon Phi™ coprocessor, as this size provides maximal granularity (or large enough work-group size to saturate computing facilities of the Intel Xeon Phi coprocessor) and enables an automatic 16-way vectorization done by the OpenCL runtime compiler. The sample application prompts you if the work group size choice is not valid.

Host part of the sample initializes three buffers, which contain randomly generated values for time to maturity, current stock price, and option strike price. Risk-free and volatility parameters are fixed. All this data is passed to the OpenCL kernel to calculate the resulting option put and call prices.

The `--validation` command-line option enables calculation of option prices on the host side using the Black and Scholes formula. Then these values are used for comparison with results calculated using the Monte Carlo simulation.

Understanding the Sample Output

The following is an example of possible sample output (with validation enabled):

```
$ ./montecarlo --validation
Platforms (1):
 [0] Intel(R) OpenCL [Selected]
Devices (2):
[0]                 Genuine Intel(R) CPU  @ 2.60GHz
[1] Intel(R) Many Integrated Core Acceleration Card [Selected]
Build program options: "-D__DO_FLOAT__ -cl-denorms-are-zero
-cl-fast-relaxed-math -cl-single-precision-constant -DNSAMP=262144"
Running Monte Carlo options pricing for 65536 options, with 262144 samples
Size of memory region for each array: 262144 bytes
Using risk free rate = 0.05 and volatility = 0.2
Host time: 2.63781 sec.
Host perf: 24844.8 Options per second
VALIDATION PASSED
Host time: 2.6169 sec.
Host perf: 25043.4 Options per second
Host time: 2.63238 sec.
Host perf: 24896.1 Options per second
Host time: 2.62573 sec.
Host perf: 24959.1 Options per second
Host time: 2.66394 sec.
Host perf: 24601.1 Options per second
Host time: 2.6682 sec.
Host perf: 24561.9 Options per second
Host time: 2.60798 sec.
Host perf: 25129.1 Options per second
Host time: 2.62947 sec.
Host perf: 24923.6 Options per second
Host time: 2.65792 sec.
Host perf: 24656.9 Options per second
Host time: 2.63868 sec.
Host perf: 24836.7 Options per second
```

First, the sample outputs all available platforms and marks (look at the line with [Selected]). Then goes the list of devices for the selected platform. The selected device is also marked.

Then follows a "Build program options" section which is exact build options line passed to the `clBuildProgram` OpenCL call.

Finally the sample calls the kernel several times and for each iteration it prints two numbers: "Host time" and "Host perf". Host time is the time measured on host for complete kernel invocation without any data transfer to or from device. Host perf is a number of modeled options calculated based on Host time.

In the case when --validation key is set in the command line, validation procedure prints validation status just after the first kernel iteration. It looks as "VALIDATION PASSED" if validation succeeded and "VALIDATION FAILED" otherwise. An example above demonstraits usage of the --validation key.

References

- http://en.wikipedia.org/wiki/Monte_Carlo_option_model
- http://en.wikipedia.org/wiki/Mersenne_twister
- Matsumoto, M.; Nishimura, T. (1998). "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". ACM Transactions on Modeling and Computer Simulation 8 (1): 3–30.
- http://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform
- http://en.wikipedia.org/wiki/Black%E2%80%93Scholes
- http://finance.bi.no/~bernt/gcc_prog/recipes/recipes/node12.html