AMD **Accelerated** Parallel Processing TECHNOLOGY

SAMPLE

Monte Carlo Simulation for Asian Option Pricing

1 Overview

1.1 Location

`$(AMDAPPSDKSAMPLESROOT)\samples\opencl\cl\app`

1.2 How to Run

See the *Getting Started* guide for how to build samples. You first must compile the sample.

Use the command line to change to the directory where the executable is located. The pre-compiled sample executable is at `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\bin\x86\` for 32-bit builds, and `$(AMDAPPSDKSAMPLESROOT)\samples\opencl\bin\x86_64\` for 64-bit builds.

Type the following command(s).

1. `MonteCarloAsian`
   Runs with the default options c = 10, i = 50, s = 55, r = 0.06, m = 1.
2. `MonteCarloAsian -h`
   This prints the help file.

1.3 Command Line Options

Table 1 lists, and briefly describes, the command line options.

Table 1 Command Line Options

Short Form	Long Form	Description
`-h`	`--help`	Shows all command options and their respective meaning.
	`--device`	Devices on which the program is to be run. Acceptable values are `cpu` or `gpu`.
`-q`	`--quiet`	Quiet mode. Suppresses all text output.
`-e`	`--verify`	Verify results against reference implementation.
`-t`	`--timing`	Print timing.
	`--dump`	Dump binary image for all devices.
	`--load`	Load binary image and execute on device.
	`--flags`	Specify compiler flags to build the kernel.
`-p`	`--platformId`	Select platformId to be used (0 to N-1, where N is the number of available platforms).
`-d`	`--deviceId`	Select deviceId to be used (0 to N-1, where N is the number of available devices).
`-v`	`--version`	AMD APP SDK version string.
`-c`	`--steps`	Steps of the Monte Carlo simulation.
`-P`	`--initPrice`	Inital price (default = 50).
`-s`	`--strikePrice`	Strike price (default = 55)
`-r`	`--interest`	Interest rate (default = 0.06)

Short Form	Long Form	Description
-m	--maturity	Maturity (default = 1).
-i	--iterations	Number of iterations for kernel execution.
	--dInPersistent	Disables the persistent memory for input buffers.
	--dOutAllocHostPtr	Disables the Alloc host ptr for output buffers.
	--dMapping	Disables mapping/unmapping, and uses read/write buffers.
	--dAsync	Disable Asynchronous.
	--scalar	Run the scalar version of the kernel (Note that the --scalar and --vector options are mutually exclusive.)
	--vector	Run the vector version of the kernel (Note that the --scalar and --vector options are mutually exclusive.)

2 Introduction

The most common definition of an *option* (see reference [1]) is an agreement between two parties, the *option seller* and the *option buyer*, whereby the option buyer is granted a right (but not an obligation), secured by the option seller, to carry out some operation (or *exercise* the option) at some moment in the future. The predetermined price is referred to as the *strike price*; the future date is called the *expiration date*.

There are two basic options types:

- A *call option* grants its holder the right to *buy* the *underlying asset* at a *strike price* at some moment in the future.
- A *put option* gives its holder the right to *sell* the *underlying asset* at a *strike price* at some moment in the future.

There are several types of options, mostly depending on when the option can be exercised.

European options can be exercised only on the expiration date. American-style options are more flexible: they can be exercised any time up to, and including, the expiration date; as such, they are generally priced at least as high as corresponding European options. Other types of options are path-dependent, or have multiple exercise dates (Asian, Bermudian). For a call option, the profit made at the exercise date is the difference between the price of the asset on that date and the strike price, minus the option price paid. For a put option, the profit made at the exercise date is the difference between the strike price and the price of the asset on that date, minus the option price paid. The price of the asset at expiration date and the strike price, therefore, strongly influence how much one is willing to pay for an option.

Other important factors in the price of an option are:

- The time to the expiration date, T: Longer periods imply a wider range of possible values for the underlying asset on the expiration date; thus, there is more uncertainty about the value of the option.
- The riskless rate of return, r, which is the annual interest rate of bonds or other “risk-free” investments: Any amount P of dollars is guaranteed to be worth $P \cdot e^{rT}$ dollars T years from now if placed today in one of these investments. In other words, if an asset is worth P dollars T years from now, it is worth $P \cdot e^{-rT}$ today.

3 Monte Carlo simulation for Asian Option

Monte Carlo analysis (see reference [1]) is a cornerstone for implementing financial models. These simulations have many advantages, including the ease of implementation, as well as the applicability to multi-dimensional problems commonly encountered in finance. Option pricing can be represented as expectations. An example is an Asian Option Call, which is a financial contract dependent on the average security price over discrete dates in the future. The assent price at some time, *t*, in the future follows the classic Black-Scholes model as follows.

Equation 1 $S_t = S_0 \, e^{(r - 0.5\,\sigma)\,t + \sigma Wt}$

Where *r* is the risk-free rate of return, σ is volatility of the asset price, and *dWt* is the increment of standard Brownian motion. The price of this option is a function of the strike price, *K*, and the option maturity, *T*, shown as follows

Equation 2 $P\,(T, K) = e^{-rT}\mathbb{E}\,\{\max\,(S_a - K, 0) \mid S_0 = s_0\}$

where the average asset price is:

Equation 3
$$S_a = \sum_{i=1}^{n} S_{ti}$$

The combination of these equations does not have a closed-form solution. We use a Monte Carlo simulation to solve this pricing problem. However, for risk management, hedging, and stress testing of a portfolio, the price-sensitivity as a function of changes to model inputs (*greeks*, as they are commonly known) becomes quite valuable. One greek of interest is *vega*, the option-price sensitivity to changes in the securities volatility, which is:

Equation 4
$$\text{vega} = \frac{dP(T, K)}{d\sigma}$$

The price and vega calculation using Monte Carlo techniques is very time-consuming for several reasons. For simulation accuracy, many Brownian motion trajectories are used for price determination. For each option simulation, there are several contract dates during the option maturity; monthly dates for an annual contract. Also, an accurate picture of price volatility is achieved by rerunning the simulation with many different values for the volatility, σ. The option trader faced with minimizing risk to their client-base and portfolio may want to have this price and volatility analysis before making trades or in post-closing analysis. For very large, multi-commodity portfolios, analysts frequently wait hours for model simulations. This affects their ability to respond quicklly to dynamic market situations or to complete a risk analysis before the next day of trading.

4 Implementation Details

Each work-item calculates the vector of four samples of price and vega from given vectors of size four of the strike price: stock price, interest, maturity and sigma. The final value of the price and vega are calculated from all the samples (1024) of price and vega on the host side. See reference [1] for more details on how to calculate the final price and vega for a given sigma.

Runtime optimizations are done using:

1. Zero copy buffers.
2. Asynchronous algorithm.
3. Mapping/unmapping and/or read/write buffer for transfer.

The implementation contains two kernels: a scalar kernel and a vector kernel. The scalar kernel works better on current generation GCN cards. The vector kernel kernel works better on VLIW and previous generation cards. The user can select the kernel using the --scalar or --vector options; otherwise, preferred vector-width for the device is queried and used. If both --scalar and --vector are specified, both the options are ignored and the default vector-width will be used.

The results for the following system are shown below.

OS : Window 7
RAM : 4GB
CPU : AMD A10-5800K APU @ 3.80GHz
GPU : AMD Radeon HD 7700—Capeverde
Mother board : Gigabyte F2A85X-UP4

Results

```
MonteCarloAsian.exe -q -t -i 10 --dMapping --dAsync --dOutAllocHostPtr
```

Steps	Time (sec)	[Transfer+Kernel] (sec)	Samples used / sec
10	1.4808	0.151386	744059

```
MonteCarloAsian.exe -q -t -i 10 --dMapping --dAsync --dOutAllocHostPtr --dInPersistent
```

Steps	Time (sec)	[Transfer+Kernel] (sec)	Samples used / sec
10	1.48124	0.147342	764479

```
MonteCarloAsian.exe -q -t -i 10 --dMapping --dAsync
```

Steps	Time (sec)	[Transfer+Kernel] (sec)	Samples used / sec
10	1.46617	0.131741	855009

```
MonteCarloAsian.exe -q -t -i 10
```

Steps	Time (sec)	[Transfer+Kernel] (sec)	Samples used / sec
10	1.42432	0.0851146	1323390

```
MonteCarloAsian.exe -q -t -i 10 --dMapping
```

Steps	Time (sec)	[Transfer+Kernel] (sec)	Samples used / sec
10	1.38753	0.0660028	1706590

5 Recommended Input Option Settings

For best performance, enter the following on the command line: `-c 256 -i 5 -q -t`

6 References

1. http://www.interactivesupercomputing.com/success/pdf/caseStudy_financialmodeling.pdf

Contact

Advanced Micro Devices, Inc.
One AMD Place
P.O. Box 3453
Sunnyvale, CA, 94088-3453
Phone: +1.408.749.4000

For AMD Accelerated Parallel Processing:
URL: developer.amd.com/appsdk
Developing: developer.amd.com/
Support: developer.amd.com/appsdksupport
Forum: developer.amd.com/openclforum