PRICING AMERICAN OPTIONS WITH LEAST SQUARES MONTE CARLO ON GPUS
Massimiliano Fatica, NVIDIA Corporation
OUTLINE

- Overview
- Least Squares Monte Carlo
- GPU implementation
- Results
- Conclusions
OVERVIEW

- Valuation and optimal exercise of American-style options is a very important practical problem in option pricing

- Early exercise feature makes the problem challenging:
  - On expiration date, the optimal exercise strategy is to exercise if the option is in the money or let it expire otherwise
  - For all the other time steps, the optimal exercise strategy is to examine the asset price, compare the immediate exercise value of the option with the risk neutral expected value of holding the option and determine if immediate exercise is more valuable
OVERVIEW

- Algorithms for American-style options:
  - Grid based (finite difference, binomial/trinomial trees)
  - Monte Carlo

- GPUs are very attractive for High Performance Computing
  - Massive multithreaded chips
  - High memory bandwidth, high FLOPS count
  - Power efficient
  - Programming languages and tools

- This work will present an implementation of the Least Squares Monte Carlo method by Longstaff and Schwartz (2001) on GPUs
LEAST SQUARES MONTE CARLO

- If N is the number of paths and M is the number of time intervals:
  - Generate a matrix R(N,M) of normal random numbers
  - Compute the asset prices S(N,M+1)
  - Compute the cash flow at M+1 since the exercise policy is known

- For each time step, going backward in time:
  - Estimate the continuation value
  - Compare the value of immediate payoff with continuation value and decide if early exercise

- Discount the cash flow to present time and average over paths
LONGSTAFF - SCHWARTZ

- Estimation of the continuation value by least squares regression using a cross section of simulated data:
  - Continuation function is approximated as linear combination of basis functions
    \[ F(., t_k) = \sum_{k=0}^{p} \alpha_k L_k(S(t_k)) \]
  - Select the paths in the money
  - Select basis functions: monomial, orthogonal polynomials (weighted Laguerre, ...)

\[ L_k(S) = S^k \]

\[
\begin{align*}
L_0(S) &= e^{-S/2} \\
L_1(S) &= e^{-S/2}(1 - S) \\
L_2(S) &= e^{-S/2}(1 - 2S + S^2/2) \\
L_k(S) &= e^{-S/2}e^S \frac{d^k}{k!} \frac{dS^k}{dS} (S^k e^{-S})
\end{align*}
\]
LEAST SQUARES REGRESSION

Select paths in the money at time t

Build matrix using basis functions

Asset price

Cash flow

\[ A \times x = b \]

(ITM,p) (p,1) (ITM,1)

Select corresponding cash flows at time t+1 and discount them at time t
LEAST SQUARES MONTE CARLO

- RNG: Plenty of parallelism
- Moment matching: Plenty of parallelism
- Path generation: Plenty of parallelism if N is large
- Regression: M dependent steps.
- Average: Plenty of parallelism
Random number generation is performed using the CURAND library:

- Single and double precision
- Normal, uniform, log-normal, Poisson distributions
- 4 different generators:
  - XORWOW: xor-shift
  - MTGP32: Mersenne-Twister
  - MRG32K32A: Combined Multiple Recursive
  - PHILOX4-32: Counter-based
RANDOM NUMBER GENERATION

- Choice of:
  - Normal distribution
  - Uniform distribution plus Box-Muller:

\[
\begin{align*}
n_0 &= \sqrt{-2 \log(u_1)} \sin(2\pi u_0) \\
n_1 &= \sqrt{-2 \log(u_1)} \cos(2\pi u_0)
\end{align*}
\]

- Optional moment matching of the data

\[
n_i^* = \frac{(n_i - \mu)}{\sigma}
\]
curandCreateGenerator(&gen, CURAND_RNG_PSEUDO_PHILOX4_32_10);
curandSetPseudoRandomGeneratorSeed(gen, myseed);
if(bm==0) { /* Generate LDA*M double with normal distribution on device */
    curandGenerateNormalDouble(gen, devData, LDA*M, 0., 1.); }
else{
    /* Generate LDA*M doubles with uniform distribution on device and then apply Box-Muller transform */
    curandGenerateUniformDouble(gen, devData, LDA*M);
    box_muller<<<256, 256>>>(devData, LDA*M);
}
.....
curandDestroyGenerator(gen);

__global__ void box_muller(double *in, size_t N) {
    int tid = threadIdx.x;
    int totalThreads = gridDim.x * blockDim.x;
    int ctaStart = blockDim.x * blockIdx.x;
    double s, c;
    for (size_t i = ctaStart + tid; i < N/2; i += totalThreads) {
        size_t ii = 2*i;
        double x = -2*(log(in[ii]));
        double y = 2*in[ii+1];
        sincospi(y, &s, &c);
        in[ii] = sqrt(x)*s;
        in[ii+1] = sqrt(x)*c;
    }
}
The stock price \( S(t) \) is assumed to follow a geometric Brownian motion:

\[
S_i(0) = S_0 \\
S_i(t + \Delta t) = S_i(t) e^{(r - \frac{\sigma^2}{2}) \Delta t + \sigma \sqrt{\Delta t} Z_i}
\]

- **Use of antithetic variables:**
  \[
  S_i(t + \Delta t) = S_i(t) e^{(r - \frac{\sigma^2}{2}) \Delta t + \sigma \sqrt{\Delta t} Z_i}
  \]
  \[
  S_i^*(t + \Delta t) = S_i^*(t) e^{(r - \frac{\sigma^2}{2}) \Delta t - \sigma \sqrt{\Delta t} Z_i}
  \]
  - reduce variance
  - reduce memory footprint
PATH GENERATION

```
__global__ void generatePath(double *S, double *CF, double *devData, double S0, double K,
                             double R, double sigma, double dt, size_t N, int M, size_t LDA)
{
    int i,j;
    int totalThreads = gridDim.x * blockDim.x;
    int ctaStart = blockDim.x * blockIdx.x;

    for (i = ctaStart + threadIdx.x; i < N/2; i += totalThreads) {
        int ii=2*i;
        S[ii]=S0;
        S[ii+1]=S0;
        \\ Compute asset price at all time steps
        for (j=1;j<M+1;j++)
        {
            S[ii + j*LDA]=S[ii + (j-1)*LDA]*exp( (R-0.5*sigma*sigma)*dt + sigma*sqrt(dt)*devData[i+(j-1)*LDA] );
            S[ii+1+j*LDA]=S[ii+1+(j-1)*LDA]*exp( (R-0.5*sigma*sigma)*dt - sigma*sqrt(dt)*devData[i+(j-1)*LDA] );
        }
        \\ Compute cash flow at time T
        CF[ii +M*LDA]=( K-S[ii +M*LDA]) >0. ?  (K-S[ii+ M*LDA]): 0.;
        CF[ii+1+M*LDA]=( K-S[ii+1+M*LDA]) >0. ?  (K-S[ii+1+M*LDA]): 0.;
    }
}
```

Simple parallelization. Each thread computes multiple antithetic paths
LEAST SQUARES SOLVER

- System solved with normal equation approach
  \[ Ax = b \quad \Rightarrow \quad A^T A x = A^T b \]

- The element \((l,m)\) of \(A^T A\) and the element \(l\) of \(A^T b\):
  \[
  \sum_{j \in ITM} L_l(j) L_m(j) \quad \text{and} \quad \sum_{j \in ITM} L_l(j) b(j)
  \]

- The matrix \(A\) is never stored, each thread loads the asset price and cash flow for one path and computes the terms on-the-fly, adding them to the sum if the path is in the money

- Two stages approach, possible use of compensated sum and extended precision
COMPUTATION OF $A^T A$
RESULTS

- CUDA 5.5
- Tesla K20X
  - 2688 cores
  - 732 MHz
  - 6 GB of memory
- Tesla K40
  - 2880 cores
  - Boost clock up to 875 MHz
  - 12 GB of memory
## RNG PERFORMANCE

<table>
<thead>
<tr>
<th>Generator</th>
<th>Distribution</th>
<th>Time (ms) $N=10^7$</th>
<th>Time (ms) $N=10^8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>XORWOW</td>
<td>Normal</td>
<td>12.99</td>
<td>34.03</td>
</tr>
<tr>
<td>XORWOW</td>
<td>Uniform +Box Muller</td>
<td>12.65</td>
<td>30.93</td>
</tr>
<tr>
<td>MTGP32</td>
<td>Normal</td>
<td>3.48</td>
<td>32.95</td>
</tr>
<tr>
<td>MTGP32</td>
<td>Uniform +Box Muller</td>
<td>3.92</td>
<td>37.53</td>
</tr>
<tr>
<td>MRG32K</td>
<td>Normal</td>
<td>4.46</td>
<td>26.44</td>
</tr>
<tr>
<td>MRG32K</td>
<td>Uniform +Box Muller</td>
<td>4.02</td>
<td>22.02</td>
</tr>
<tr>
<td>PHILOX</td>
<td>Normal</td>
<td>2.89</td>
<td>27.40</td>
</tr>
<tr>
<td>PHILOX</td>
<td>Uniform +Box Muller</td>
<td>2.53</td>
<td>24.12</td>
</tr>
</tbody>
</table>
### COMPARISON WITH LONGSTAFF-SCHWARTZ

<table>
<thead>
<tr>
<th>S</th>
<th>σ</th>
<th>T</th>
<th>Finite difference</th>
<th>Longstaff paper</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>.20</td>
<td>1</td>
<td>4.478</td>
<td>4.472</td>
<td>4.473</td>
</tr>
<tr>
<td>36</td>
<td>.20</td>
<td>2</td>
<td>4.840</td>
<td>4.821</td>
<td>4.854</td>
</tr>
<tr>
<td>36</td>
<td>.40</td>
<td>1</td>
<td>7.101</td>
<td>7.091</td>
<td>7.098</td>
</tr>
<tr>
<td>36</td>
<td>.40</td>
<td>2</td>
<td>8.508</td>
<td>8.488</td>
<td>8.501</td>
</tr>
<tr>
<td>38</td>
<td>.20</td>
<td>1</td>
<td>3.250</td>
<td>3.244</td>
<td>3.248</td>
</tr>
<tr>
<td>38</td>
<td>.40</td>
<td>2</td>
<td>3.745</td>
<td>3.735</td>
<td>3.746</td>
</tr>
<tr>
<td>38</td>
<td>.20</td>
<td>1</td>
<td>6.148</td>
<td>6.139</td>
<td>6.138</td>
</tr>
<tr>
<td>38</td>
<td>.40</td>
<td>2</td>
<td>7.670</td>
<td>7.669</td>
<td>7.663</td>
</tr>
<tr>
<td>44</td>
<td>.20</td>
<td>1</td>
<td>1.110</td>
<td>1.118</td>
<td>1.112</td>
</tr>
<tr>
<td>44</td>
<td>.40</td>
<td>2</td>
<td>1.690</td>
<td>1.675</td>
<td>1.684</td>
</tr>
<tr>
<td>44</td>
<td>.20</td>
<td>1</td>
<td>3.948</td>
<td>3.957</td>
<td>3.944</td>
</tr>
<tr>
<td>44</td>
<td>.40</td>
<td>2</td>
<td>5.647</td>
<td>5.622</td>
<td>5.627</td>
</tr>
</tbody>
</table>

Finite differences: implicit scheme with 40000 time steps per year, 1000 steps per stock price. LSMC with 100000 path and 50 time steps. Philox generator for GPU results.
ACCURACY VS QR SOLVER

Put option with strike price=40, stock price=36, variability=.2, r=.06, T=2
Reference value is $4.840$

<table>
<thead>
<tr>
<th>Basis functions</th>
<th>Normal Equation (GPU)</th>
<th>QR (CPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.740095193796793</td>
<td>4.740095193796793</td>
</tr>
<tr>
<td>3</td>
<td>4.815731393932048</td>
<td>4.815731393932048</td>
</tr>
<tr>
<td>4</td>
<td>4.833172186198728</td>
<td>4.833172186198728</td>
</tr>
<tr>
<td>5</td>
<td>4.833251309474664</td>
<td>4.833251309474664</td>
</tr>
<tr>
<td>6</td>
<td>4.83580505904685</td>
<td>4.836251721596485</td>
</tr>
<tr>
<td>7</td>
<td>4.837584550853037</td>
<td>4.837803730367345</td>
</tr>
<tr>
<td>8</td>
<td>4.838283073214879</td>
<td>4.839358646526560</td>
</tr>
</tbody>
</table>

Regression coefficients at the final step for 4 basis functions

<table>
<thead>
<tr>
<th>Normal equation</th>
<th>155.156982160074</th>
<th>-397.156557357517</th>
<th>353.391768458585</th>
<th>-108.545827892269</th>
</tr>
</thead>
<tbody>
<tr>
<td>QR</td>
<td>155.157227263422</td>
<td>-397.157372674635</td>
<td>353.392671772303</td>
<td>-108.546161230726</td>
</tr>
</tbody>
</table>
RESULTS DOUBLE PRECISION

nvprof ./american_dp -g3
American put option N=524288 (LDA=524288) M=50 dt=0.020000
Strike price=40.000000 Stock price=36.000000 sigma=0.200000 r=0.060000 T=1.000000

Generator: MRG
BlackScholes put = 3.844
Normal distribution
RNG generation time = 8.763488 ms
Path generation time = 3.288832 ms
LS time = 7.512192 ms, perf = 136.792 GB/s
GPU Mean price = 4.476522e+00

<table>
<thead>
<tr>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4127ms</td>
<td>1</td>
<td>6.4127ms</td>
<td>6.4127ms</td>
<td>6.4127ms</td>
<td>gen_sequence&lt;curandStateMRG32k3a</td>
</tr>
<tr>
<td>3.4666ms</td>
<td>49</td>
<td>70.746us</td>
<td>69.632us</td>
<td>71.680us</td>
<td>second_kernel</td>
</tr>
<tr>
<td>3.2417ms</td>
<td>1</td>
<td>3.2417ms</td>
<td>3.2417ms</td>
<td>3.2417ms</td>
<td>generatePath</td>
</tr>
<tr>
<td>3.0826ms</td>
<td>49</td>
<td>62.909us</td>
<td>61.856us</td>
<td>63.840us</td>
<td>tall_gemm</td>
</tr>
<tr>
<td>1.9224ms</td>
<td>1</td>
<td>1.9224ms</td>
<td>1.9224ms</td>
<td>1.9224ms</td>
<td>generate_seed_pseudo_mrg</td>
</tr>
<tr>
<td>480.03us</td>
<td>49</td>
<td>9.7960us</td>
<td>9.5360us</td>
<td>10.208us</td>
<td>second_pass</td>
</tr>
<tr>
<td>126.24us</td>
<td>1</td>
<td>126.24us</td>
<td>126.24us</td>
<td>126.24us</td>
<td>redusum</td>
</tr>
<tr>
<td>12.384us</td>
<td>3</td>
<td>4.1280us</td>
<td>3.7760us</td>
<td>4.3200us</td>
<td>[CUDA memset]</td>
</tr>
<tr>
<td>11.936us</td>
<td>1</td>
<td>11.936us</td>
<td>11.936us</td>
<td>11.936us</td>
<td>BlackScholes</td>
</tr>
<tr>
<td>5.7920us</td>
<td>2</td>
<td>2.8960us</td>
<td>2.8480us</td>
<td>2.9440us</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
</tbody>
</table>
RESULTS SINGLE PRECISION

nvprof ./american_sp -g3
American put option N=524288 (LDA=524288) M=50 dt=0.020000
Strike price=40.000000 Stock price=36.000000 sigma=0.200000 r=0.060000 T=1.000000

Generator: MRG
BlackScholes put = 3.844
Normal distribution
RNG generation time = 5.920544 ms
Path generation time = 1.882912 ms
LS time = 6.319168 ms, perf = 162.617 GB/s
GPU Mean price =4.475582e+00

<table>
<thead>
<tr>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5837ms</td>
<td>1</td>
<td>3.5837ms</td>
<td>3.5837ms</td>
<td>3.5837ms</td>
<td>gen_sequenced&lt;curandStateMRG32k3a</td>
</tr>
<tr>
<td>3.0940ms</td>
<td>49</td>
<td>63.142us</td>
<td>61.984us</td>
<td>64.256us</td>
<td>tall_gemm</td>
</tr>
<tr>
<td>2.2367ms</td>
<td>49</td>
<td>45.646us</td>
<td>44.608us</td>
<td>46.688us</td>
<td>second_kernel</td>
</tr>
<tr>
<td>1.9065ms</td>
<td>1</td>
<td>1.9065ms</td>
<td>1.9065ms</td>
<td>1.9065ms</td>
<td>generate_seed_pseudo_mrg</td>
</tr>
<tr>
<td>1.8345ms</td>
<td>1</td>
<td>1.8345ms</td>
<td>1.8345ms</td>
<td>1.8345ms</td>
<td>generatePath</td>
</tr>
<tr>
<td>505.38us</td>
<td>49</td>
<td>10.313us</td>
<td>10.144us</td>
<td>10.656us</td>
<td>second_pass</td>
</tr>
<tr>
<td>127.71us</td>
<td>1</td>
<td>127.71us</td>
<td>127.71us</td>
<td>127.71us</td>
<td>redusum</td>
</tr>
<tr>
<td>12.544us</td>
<td>3</td>
<td>4.1810us</td>
<td>3.8080us</td>
<td>4.3840us</td>
<td>[CUDA memset]</td>
</tr>
<tr>
<td>7.8080us</td>
<td>1</td>
<td>7.8080us</td>
<td>7.8080us</td>
<td>7.8080us</td>
<td>BlackScholes</td>
</tr>
<tr>
<td>5.7280us</td>
<td>2</td>
<td>2.8640us</td>
<td>2.7840us</td>
<td>2.9440us</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
</tbody>
</table>
PERFORMANCE COMPARISON WITH CPU

- 256 time steps, 3 regression coefficients
- CPU and GPU runs with double precision, MRGK32A RNG

<table>
<thead>
<tr>
<th>Paths</th>
<th>Sequential*</th>
<th>Xeon E5-2670* (OpenMP, vect)</th>
<th>K20X</th>
<th>K40</th>
<th>K40 ECC off</th>
</tr>
</thead>
<tbody>
<tr>
<td>128K</td>
<td>4234ms</td>
<td>89ms</td>
<td>26.5ms</td>
<td>22.9ms</td>
<td>21.2ms</td>
</tr>
<tr>
<td>256K</td>
<td>8473ms</td>
<td>171ms</td>
<td>43.9ms</td>
<td>38.0ms</td>
<td>35.1ms</td>
</tr>
<tr>
<td>512K</td>
<td>17192ms</td>
<td>339ms</td>
<td>78.8ms</td>
<td>67.7ms</td>
<td>63.2ms</td>
</tr>
</tbody>
</table>

For the GPU version going from 3 terms to 6 terms only increases the runtime to 66.4ms. The solve phase goes from 27.8ms to 30.8ms.

* Source Xcelerit blog
CONCLUSIONS

- Successfully implemented the Least Squares Monte Carlo method on GPU
- Correct and fast results
- Future work:
  - QR decomposition on GPU

Massimiliano Fatica and Everett Phillips (2013) “Pricing American options with least squares Monte Carlo on GPUs”. In Proceedings of the 6th Workshop on High Performance Computational Finance (WHPCF ’13). ACM, New York, NY, USA,