Accelerating Option Risk Analytics in R Using GPUs

Matthew Dixon\textsuperscript{1}, Sabbir Khan\textsuperscript{2} and Mohammad Zubair\textsuperscript{2}

\textsuperscript{1}Department of Analytics
School of Management
University of San Francisco
Email: mfdixon@usfca.edu

\textsuperscript{2}Department of Computer Science
Old Dominion University
Norfolk, VA
Email: zubair@cs.odu.edu

GTC’14
27th March, 2014
Background resources

- **Paper:**

- **Development version of R package:**
  https://github.com/mfrdixon/gpusvcalibration
Overview of presentation

- Learn why stochastic volatility (SV) models are used extensively across the capital markets for pricing and risk management of exchange traded financial options.
- Understand some of the factors which make calibration of SV models computationally intensive.
- Gain insight into the pros and cons of using the R statistical software environment for calibrating SV models.
- Learn about the gpusvcalibration R package for accelerating stochastic volatility model calibration on GPUs, which provides a factor of up to 760x performance improvement on a NVIDIA Tesla K20c (Kepler architecture) consisting of 2496 cores.
Implementation Gap

- Application domain experts make design trade-offs without full view of parallel performance implications
- Expert parallel programmer with limited knowledge of application design trade-offs

Diagram:
- End User
- Target Application
- Application Developer
- Expert Parallel Programmer
- HW Platform
- SW Infrastructure
- Parallel Platform
Market makers quote options for strike-expiry pairs which are illiquid or not listed.

Pricing engines, which are used to price exotic options and which are based on far more realistic assumptions than Black-Scholes-Merton model, are calibrated against an observed Implied Volatility Surface (IVS).

The IVS given by a listed market serves as the market of primary hedging instruments against volatility and gamma risk (second-order sensitivity with respect to the spot).

Risk managers use stress scenarios defined on the IVS to visualize and quantify the risk inherent to option portfolios.
Evidence of the Leverage Effect

Daily changes of squared volatility indices versus daily returns. Using the volatility indices as the proxy of volatility, the leverage effect can clearly be seen. Left: S&P 500 data from January 2004 to December 2007, in which the VIX is used as a proxy of the volatility; Right: Dow Jones Industrial Average data from January 2005 to March 2007 in which the Chicago Board Options Exchange (CBOE) DJIA Volatility Index (VXD) is used as the volatility measure.

Stochastic Volatility Modeling

1. Choose a SV model, e.g. Bates Model

   **Definition (Bates Model)**
   
   \[
   \frac{dS_t}{S_t} = \mu dt + \sqrt{V_t} dW^1_t + (Y - 1) S_t dN_t, \tag{1}
   \]
   
   \[
   \frac{dV_t}{V_t} = \kappa(\theta - V_t) dt + \sigma \sqrt{V_t} dW^2_t. \tag{2}
   \]

   - \(V_t\) is given by a mean reverting square root process constrained by \(2\kappa\theta - \sigma^2 > 0\).
   - \(N_t\) is a standard Poisson process with intensity \(\lambda > 0\) and \(Y\) is the log-normal jump size distribution, where \(\mu_j = \ln(1 + a) - \frac{\sigma_j^2}{2}, a > -1\) and \(\sigma_j \geq 0\).

2. Estimate the European Option Price

   **Definition (Bates European Option Pricing Model)**

   - Call price of a vanilla European option is
     
     \[
     C(S_0, K, \tau; p) = S_0 P_1 - K \exp\{-(r - q)\tau\} P_2,
     \]
     
   - \(P_1\) and \(P_2\) can be expressed as:
     
     \[
     P_j = \frac{1}{2} + \frac{1}{\pi} \int_0^\infty \Re \left[ \exp\left\{-iu K\sigma_j^2 \right\} \frac{\phi_j(S_0, \tau; u; p)}{iu} \right] du, j = 1, 2.
     \]
     
   - where the parameter set \(p := [\theta, \kappa, \rho, \nu_0, \mu_j, \sigma_j, \lambda]\).

3. Imply the volatility surface from the fitted model prices

4. Trading desks use the implied volatility surface to price non-quoted options and price and hedge exotics
Error convergence of FFT versus Fourier-Cosine\textsuperscript{1}

Figure: Comparison of the error convergence rates of the Fourier-Cosine, fixed second order Gauss-Legendre quadrature and Carr-Madan FFT methods applied to the Heston pricing model.

\textsuperscript{1}F. Fang, C.W. Oosterlee, SIAM Journal on Scientific Computing 31 (2), 2008, 826-848
There is a need to refrequently re-calibrate the models.

**Figure**: The maximum point-wise error across the volatility surface for options on ZNGA against time. The red-line shows the error resulting from calibration of the Bates model every 30 seconds versus calibrating at the start of the period (black-line).
Global calibration in R

1. Call the Differential Evolution Algorithm - this is a stochastic parallel direct search evolution strategy optimization method. Specify a maximum number of candidates and a tolerance on the error function. The DE algorithm is available in the DEoptim package\(^2\).

2. Call an iterative constraint based optimizer: specify tolerances on various errors, e.g. relative function error or relative solution changes. The NLopt package provides various local optimizers.

3. **Performance penalty using R**: can not quickly calibrate the model which hinders modeling, testing and productionization.

For calibrating the option price model we consider a sample chain of $n$ option data $ch[n]$, where the $i^{th}$ chain data has the following properties:

- $ch[i].u$: Underlying asset price
- $ch[i].s$: Strike price
- $ch[i].m$: Time to maturity
- $ch[i].p$: Option price
Sequential Error Function $(p)$

1. $\text{rmse} \leftarrow 0$
2. for $i = 0$ to $n - 1$ do
3. \hspace{1em} $\text{vp} \leftarrow V(ch[i].u, ch[i].s, ch[i].m, p)$
4. \hspace{1em} $\text{diff} \leftarrow ch[i].p - \text{vp}$
5. \hspace{1em} $\text{rmse} \leftarrow \text{rmse} + \text{diff} \times \text{diff}$
6. end for
7. $\text{rmse} \leftarrow \sqrt{\text{rmse}/n}$
8. return $\text{rmse}$
Sequential implementation of the calibration program in R

<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>AMZN</th>
<th>BP</th>
<th>CSCO</th>
<th>GOOG</th>
<th>MSFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEoptim</td>
<td>293</td>
<td>111</td>
<td>69</td>
<td>62</td>
<td>255</td>
<td>70</td>
</tr>
<tr>
<td>nloptr</td>
<td>148</td>
<td>56</td>
<td>35</td>
<td>32</td>
<td>130</td>
<td>36</td>
</tr>
<tr>
<td>ErrorFunction</td>
<td>1.46</td>
<td>0.55</td>
<td>0.34</td>
<td>0.31</td>
<td>1.28</td>
<td>0.35</td>
</tr>
<tr>
<td>Total ErrorFunction</td>
<td>440</td>
<td>166</td>
<td>103</td>
<td>93</td>
<td>385</td>
<td>105</td>
</tr>
<tr>
<td>Total Time</td>
<td>441</td>
<td>167</td>
<td>104</td>
<td>94</td>
<td>386</td>
<td>106</td>
</tr>
<tr>
<td>% ErrorFunction</td>
<td>99.8%</td>
<td>99.4%</td>
<td>99.1%</td>
<td>99.0%</td>
<td>99.7%</td>
<td>99.1%</td>
</tr>
</tbody>
</table>

Table: Performance results for the R code in seconds. Each column represents a different option chain.
## Sequential implementation of the calibration program in C

<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>AMZN</th>
<th>BP</th>
<th>CSCO</th>
<th>GOOG</th>
<th>MSFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEoptim</td>
<td>88</td>
<td>33</td>
<td>21</td>
<td>18</td>
<td>76</td>
<td>22</td>
</tr>
<tr>
<td>Nlopt</td>
<td>44</td>
<td>17</td>
<td>10</td>
<td>9</td>
<td>38</td>
<td>11</td>
</tr>
<tr>
<td>ErrorFunction</td>
<td>0.44</td>
<td>0.17</td>
<td>0.1</td>
<td>0.1</td>
<td>0.38</td>
<td>0.11</td>
</tr>
<tr>
<td>Total ErrorFunction</td>
<td>131</td>
<td>49</td>
<td>30</td>
<td>27.6</td>
<td>113</td>
<td>32</td>
</tr>
<tr>
<td>Total Time</td>
<td>132</td>
<td>50</td>
<td>31</td>
<td>27</td>
<td>114</td>
<td>33</td>
</tr>
<tr>
<td>% ErrorFunction</td>
<td>99.2%</td>
<td>98.0%</td>
<td>96.8%</td>
<td>99.9%</td>
<td>99.1%</td>
<td>97.0%</td>
</tr>
</tbody>
</table>

**Table:** Performance results for the C code in seconds.
GPU Implementation

Block_1

bx=0

tx=0

tx=255

Shared Memory

Block_2

bx=1

tx=0

tx=255

Shared Memory

Block_n

bx=n-1

tx=0

tx=255

Shared Memory
Each option in the chain is mapped to a thread block.

Each thread computes a term of the Fourier-Cosine series.

Each intermediate result is stored in shared memory.

Block_1

- \( bx = 0 \)
- \( tx = 0 \) to \( tx = 255 \)

Block_2

- \( bx = 1 \)
- \( tx = 0 \) to \( tx = 255 \)

Block_n

- \( bx = n-1 \)
- \( tx = 0 \) to \( tx = 255 \)
Parallel-Fourier-Cosine(p)

1: shared memory smem[]
2: tx ← threadIdx.x
3: bx ← blockIdx.x
4: bd ← blockDim.x
5: j ← bd
6: smem[tx] ← CHARACTERISTICFUNCTION(T[bx], p)
7: for i = 1 to log2(bd) do
8:   j ← j/2
9:   if tx < j then
11:   end if
12: end for
13: if tx = 0 then
14:   V[bx] ← K[bx] × exp(−r0 × T[bx] × smem[0])
15: end if
16: return V[bx]
Parallel reduction using shared memory

(tx=0, tx=1, tx=2, tx=3, tx=4, tx=5, tx=6, tx=7)

i=1, j=4
i=2, j=2
i=3, j=1
Overview of the *gpusvcalibration* library

- The *gpusvcalibration* library accelerates stochastic volatility model calibration by off-loading the error function on to the GPU.
- The package is designed to hide the parallelism from the R user.
- The package is designed for use with existing non-convex optimization CRAN packages such as DEoptim and nloptr.
- The library currently supports European option pricing under four different stochastic volatility models and more models are planned for the future.
R Wrapper for C++/CUDA implementation

ErrorFunction <- function(z) {
  if (!is.loaded('gpuMapReduce')) {
    dyn.load('gpuMapReduce.so')
  }
  RMSE <- .Call("ErrorFunction", as.numeric(p))
  return (RMSE)
}
Sample code for performance benchmarking

gpusvcalibration

```
library("gpusvcalibration")
library("DEoptim")
library("nloptr")
chain <- Load_Chain(fileName)
Copy_Data(chain)
Set_Model('Heston') # {"Heston","Bates","VG",CGMY"}
Set_Block_Size(256)

l <- c(eps,eps,eps,-1.0 + eps, eps)
u <- c(5.0-eps,1.0-eps,1.0-eps,1.0-eps,1.0-eps)
args <- list(NP=100, itermax=25)
DEres<- DEoptim(fn=Error_Function, lower=l, upper=u, control=args)
res <- nloptr(x0=as.numeric(DEres$optim$bestmem), eval_f=Error_Function,
              lb = l, ub = u,
              opts=list("algorithm"="NLOPT_LN_COBYLA", "xtol_rel" = xtol))
Dealloc_Data()
```
Performance benchmarks of the GPU implementation

<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>AMZN</th>
<th>BP</th>
<th>CSCO</th>
<th>GOOG</th>
<th>MSFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEoptim</td>
<td>0.31</td>
<td>0.12</td>
<td>0.08</td>
<td>0.074</td>
<td>0.29</td>
<td>0.08</td>
</tr>
<tr>
<td>nloptr</td>
<td>0.16</td>
<td>0.063</td>
<td>0.044</td>
<td>0.041</td>
<td>0.15</td>
<td>0.044</td>
</tr>
<tr>
<td>ErrorFunction (ms)</td>
<td>1.41</td>
<td>0.56</td>
<td>0.36</td>
<td>0.33</td>
<td>1.25</td>
<td>0.36</td>
</tr>
<tr>
<td>Total ErrorFunction</td>
<td>0.42</td>
<td>0.17</td>
<td>0.11</td>
<td>0.10</td>
<td>0.38</td>
<td>0.11</td>
</tr>
<tr>
<td>Total Time</td>
<td>0.53</td>
<td>0.23</td>
<td>0.17</td>
<td>0.16</td>
<td>0.44</td>
<td>0.18</td>
</tr>
<tr>
<td>% ErrorFunction</td>
<td>79.2%</td>
<td>73.9%</td>
<td>64.7%</td>
<td>62.5%</td>
<td>86.3%</td>
<td>61.1%</td>
</tr>
</tbody>
</table>

Table: Performance results for the RGPU code. Timings are shown in seconds unless stated otherwise.

Based on an Intel Core i5 processor and NVIDIA Tesla K20c (Kepler) with 2496 cores.
### Sequential implementation of the calibration program in R

<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>AMZN</th>
<th>BP</th>
<th>CSCO</th>
<th>GOOG</th>
<th>MSFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEoptim</td>
<td>293</td>
<td>111</td>
<td>69</td>
<td>62</td>
<td>255</td>
<td>70</td>
</tr>
<tr>
<td>nloptr</td>
<td>148</td>
<td>56</td>
<td>35</td>
<td>32</td>
<td>130</td>
<td>36</td>
</tr>
<tr>
<td>ErrorFunction</td>
<td>1.46</td>
<td>0.55</td>
<td>0.34</td>
<td>0.31</td>
<td>1.28</td>
<td>0.35</td>
</tr>
<tr>
<td>Total ErrorFunction</td>
<td>440</td>
<td>166</td>
<td>103</td>
<td>93</td>
<td>385</td>
<td>105</td>
</tr>
<tr>
<td>Total Time</td>
<td>441</td>
<td>167</td>
<td>104</td>
<td>94</td>
<td>386</td>
<td>106</td>
</tr>
<tr>
<td>% ErrorFunction</td>
<td>99.8%</td>
<td>99.4%</td>
<td>99.1%</td>
<td>99.0%</td>
<td>99.7%</td>
<td>99.1%</td>
</tr>
</tbody>
</table>

**Table:** Performance results for the R code in seconds. Each column represents a different option chain.
Comparative Performance of the GPU implementation

<table>
<thead>
<tr>
<th></th>
<th>AAPL</th>
<th>AMZN</th>
<th>BP</th>
<th>CSCO</th>
<th>GOOG</th>
<th>MSFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>R/RGPU</td>
<td>1042</td>
<td>992</td>
<td>971</td>
<td>933</td>
<td>1024</td>
<td>961</td>
</tr>
<tr>
<td>C/RGPU</td>
<td>313</td>
<td>297</td>
<td>288</td>
<td>278</td>
<td>303</td>
<td>297</td>
</tr>
<tr>
<td>CGPU/RGPU</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table: Performance comparison of the offloaded ErrorFunction to a serial R implementation, a serial C/C++ implementation and a CGPU implementation.

---

4 Based on an Intel Core i5 processor and NVIDIA Tesla K20c (Kepler) with 2496 cores.
Summary

- Stochastic volatility (SV) models are used extensively across the capital markets for pricing and risk management of exchange traded financial options.
- Calibration of SV models is computationally intensive and requires specialist optimization routines.
- Our *gpusvcalibration* R package accelerates SV model calibration on GPUs and can be used with R optimization packages.
- Demonstrated a factor of up to 760x performance improvement on a NVIDIA Tesla K20c (Kepler architecture).
Resources

- **Paper:**

- **Development version of R package:**
  https://github.com/mfrdixon/gpusvcalibration