gpusvcalibration: Fast Stochastic Volatility Model Calibration using GPUs

Matthew Dixon[1], Sabbir Khan[2] and Mohammad Zubair[2]

[1]Department of Analytics
School of Management
University of San Francisco
Email: mfdixon@usfca.edu
[2]Department of Computer Science
Old Dominion University
Norfolk, VA
Email: zubair@cs.odu.edu

R/Finance 2014
17th May

Accompanying resources

- Alpha version of R package:

```
library(devtools)
install_github("gpusvcalibration",username = "mfrdixon")
```

  Note: installation requires NVIDIA CUDA compiler (nvcc)
- Papers:
  - M.F. Dixon, S. Khan and M. Zubair, gpusvcalibration: A R Package for Fast Stochastic Volatility Model Calibration using GPUs, R/Finance, Chicago, 2014
  - M.F. Dixon, S. Khan and M. Zubair, Accelerating Option Risk Analytics in R using GPUs, Proceedings of HPC'14, Tampa, 2014.
  - M.F. Dixon and M. Zubair, Calibration of Stochastic Volatility Models on a Multi-Core CPU Cluster, In Proceedings of the Workshop on High Performance Computational Finance, SC13, November, 2013.

Overview of presentation

- Background on General Purpose Computing on Graphics Processing Units (GPUs)
- Review of stochastic volatility modeling and calibration
- Overview of the gpusvcalibration R package for accelerating stochastic volatility model calibration on GPUs - provides a factor of up to 760x performance improvement by offloading bottle-neck computations to the GPU.

Shift to Parallelism

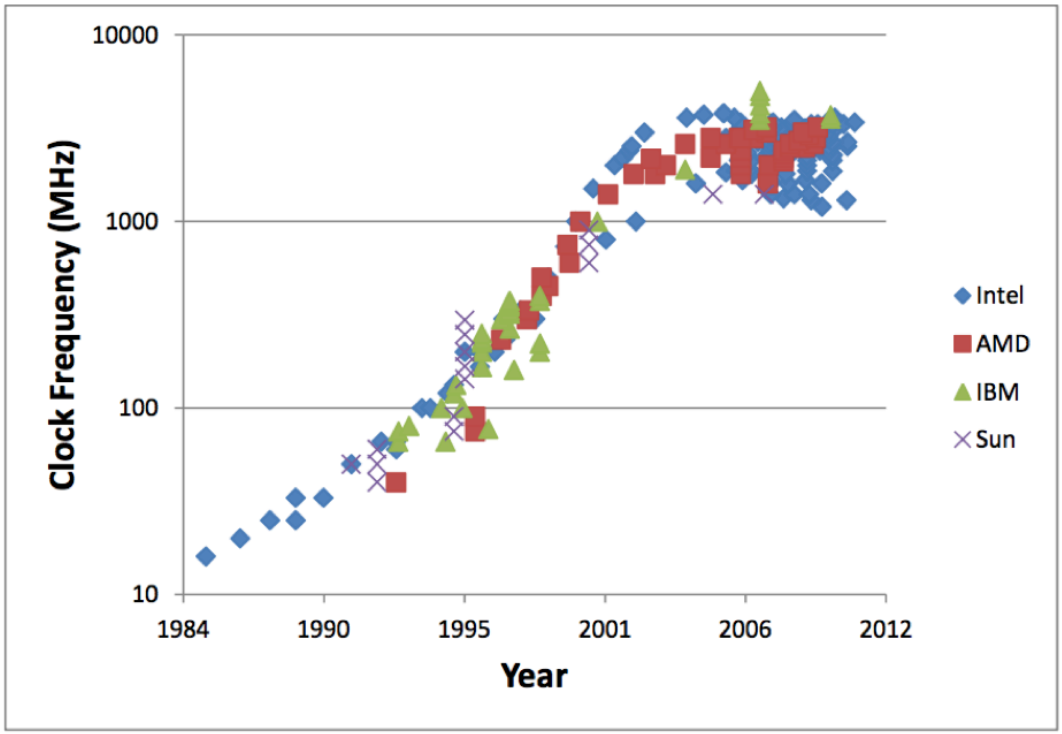
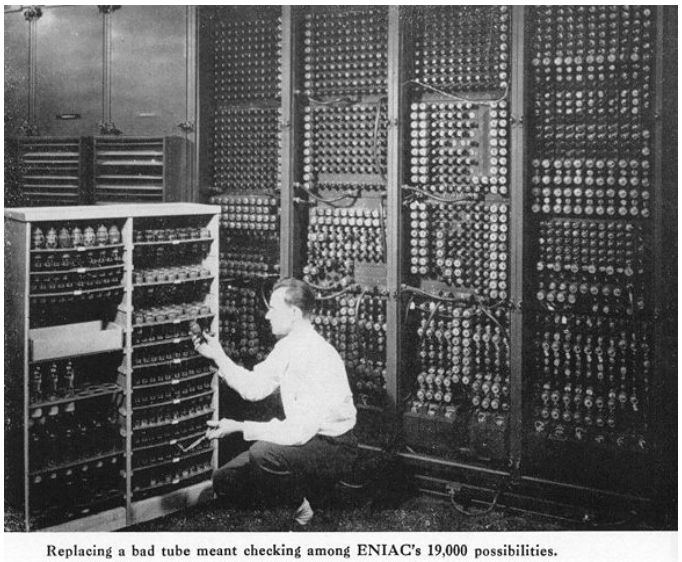


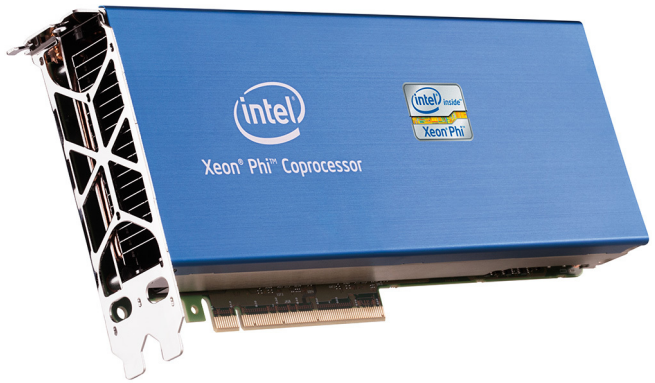
Figure 1.1: Scaling of the processor clock speeds.

Computer architecture transformation in just 75 years...

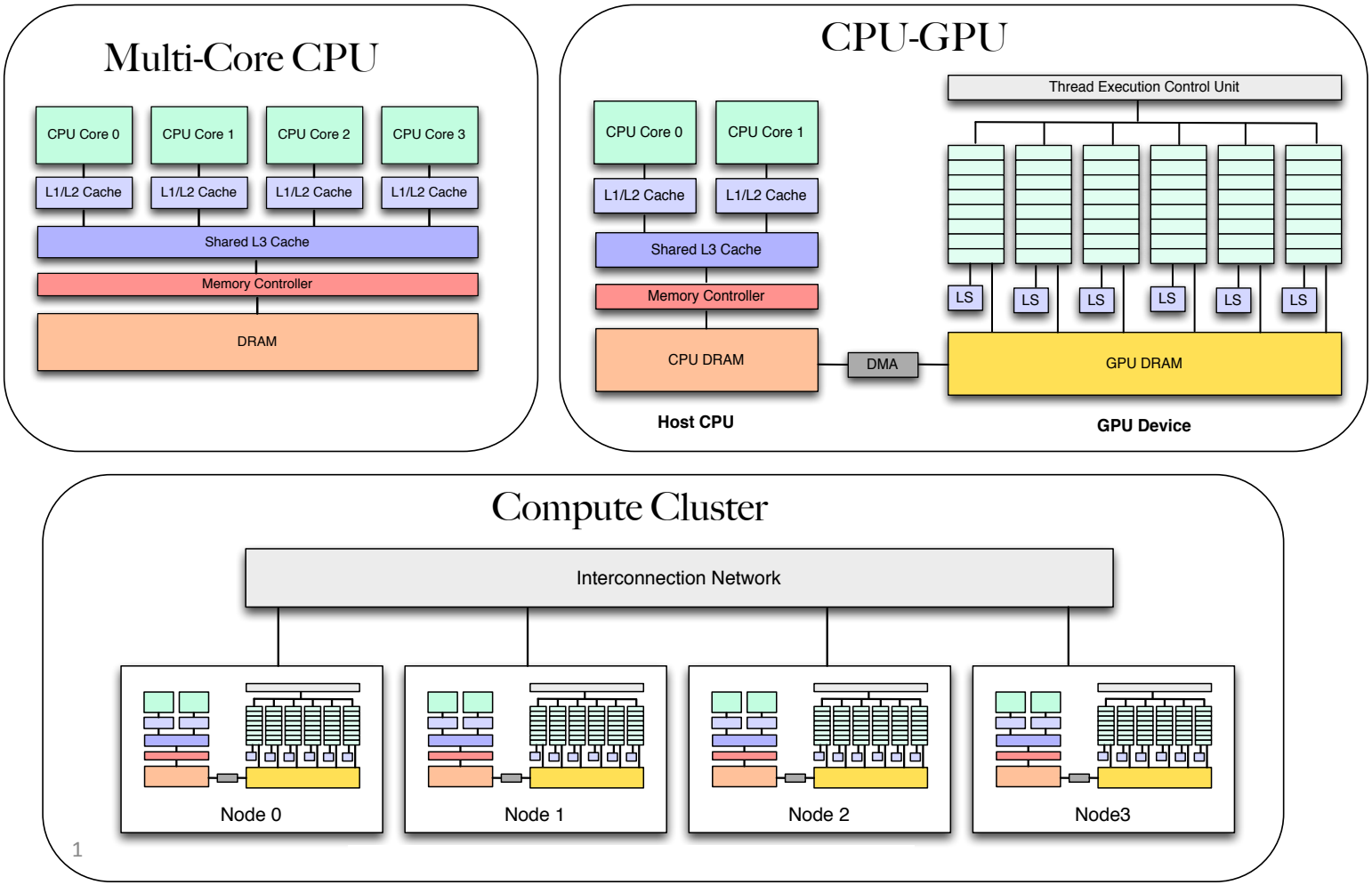


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

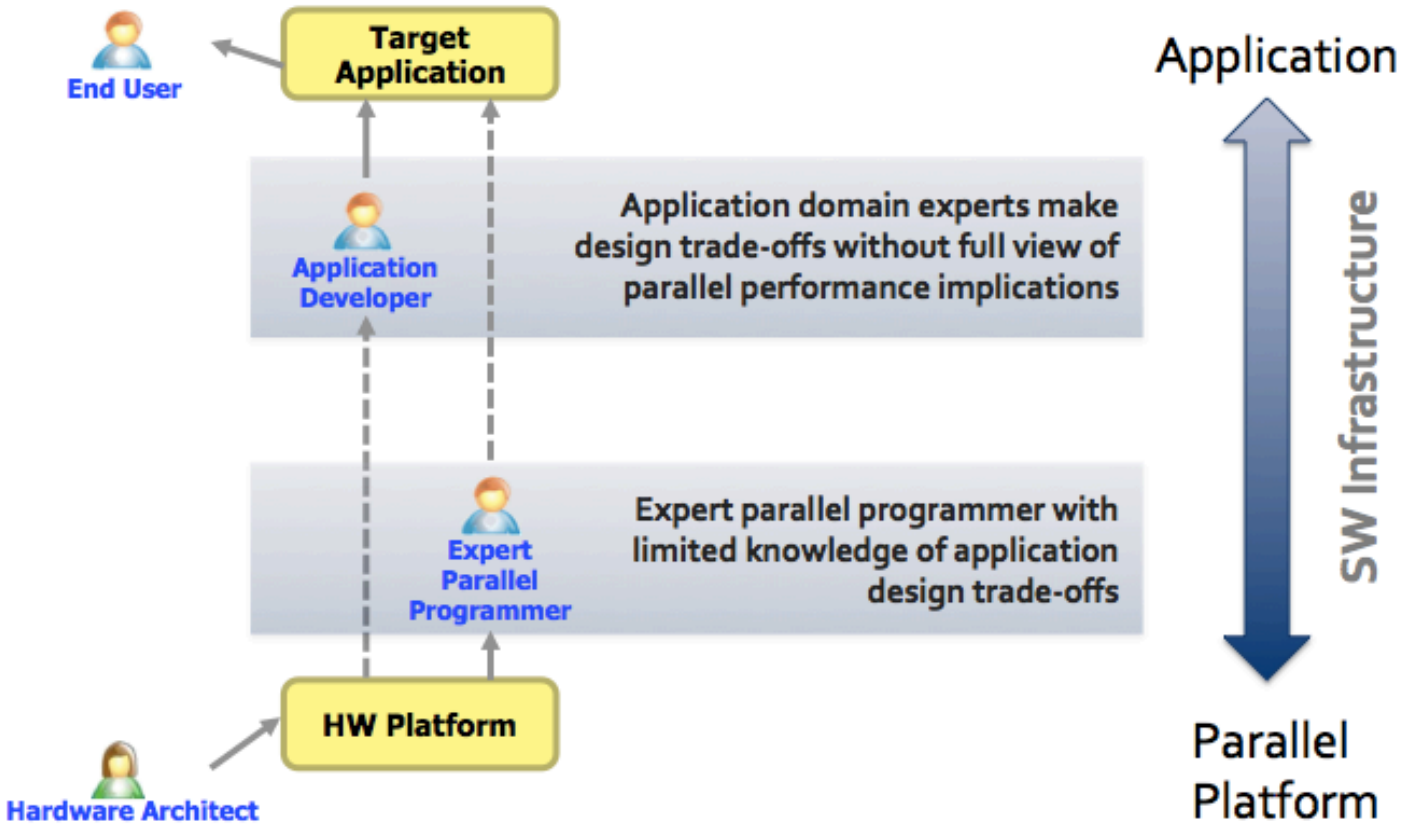
ENIAC: The first numerical weather simulation run by John Von Neumann (Los Alamos National Laboratory, 1946)



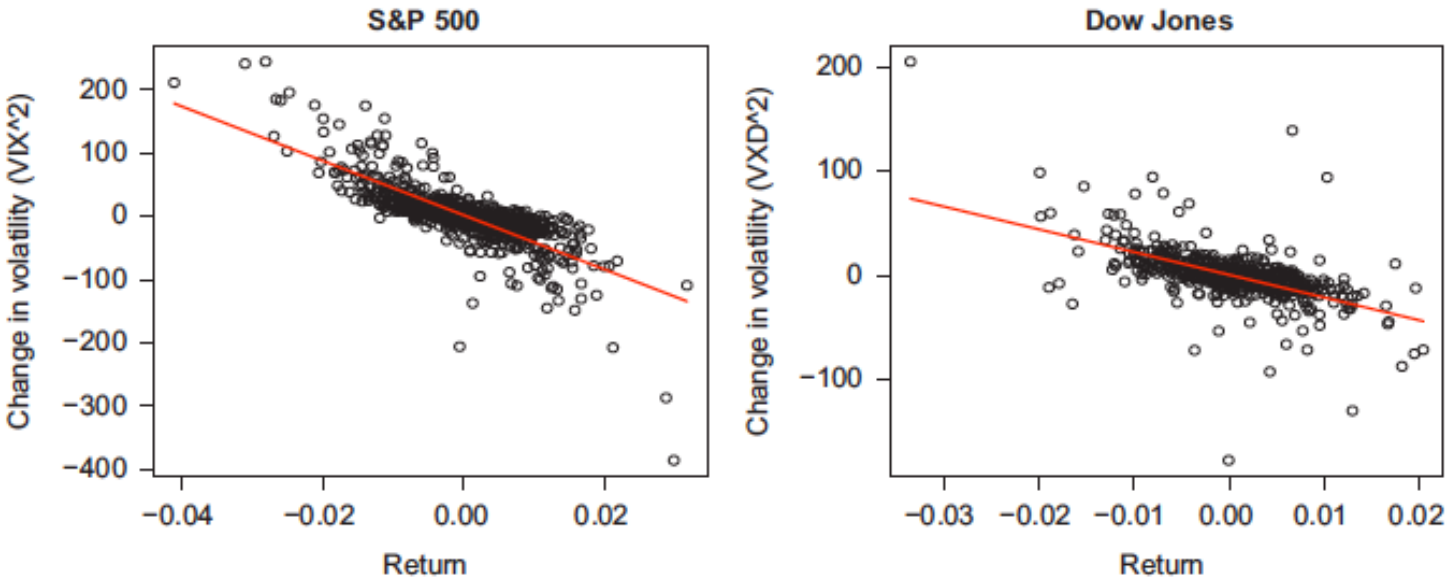
Variety of Parallel Hardware



Implementation Gap



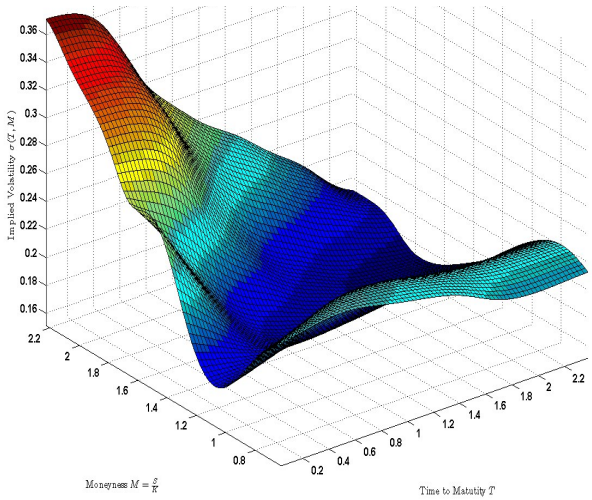
Evidence of the Leverage Effect



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.

Source: Yacine Ait-Sahalia, Jianqing Fan and Yingying Li, The leverage effect puzzle: Disentangling sources of bias at high frequency, Journal of Financial Economics 109 (2013): 224–249

Stochastic Volatility Modeling



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

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

Definition (Bates Model)

$$\frac{dS_t}{S_t} = \mu dt + \sqrt{V_t} dW_t^1 + (Y-1) S_t dN_t, \quad (1)$$

$$\frac{dV_t}{V_t} = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^2, \quad (2)$$

- 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*

$$C(S_0, K, \tau; \mathbf{p}) = S_0 P_1 - K \exp\{-(r-q)\tau\} P_2,$$

- P_1 *and* P_2 *can be expressed as:*

$$P_j = \frac{1}{2} + \frac{1}{\pi} \int_o^\infty Re\left[\frac{\exp\{-iulnK\}\phi_j(S_0, \tau, u; \mathbf{p})}{iu}\right] du, j = 1, 2.$$

- *where the parameter set* $\mathbf{p} := [\theta, \sigma, \kappa, \rho, v_0, \mu_j, \sigma_j, \lambda]$.

Error convergence of FFT versus Fourier-Cosine

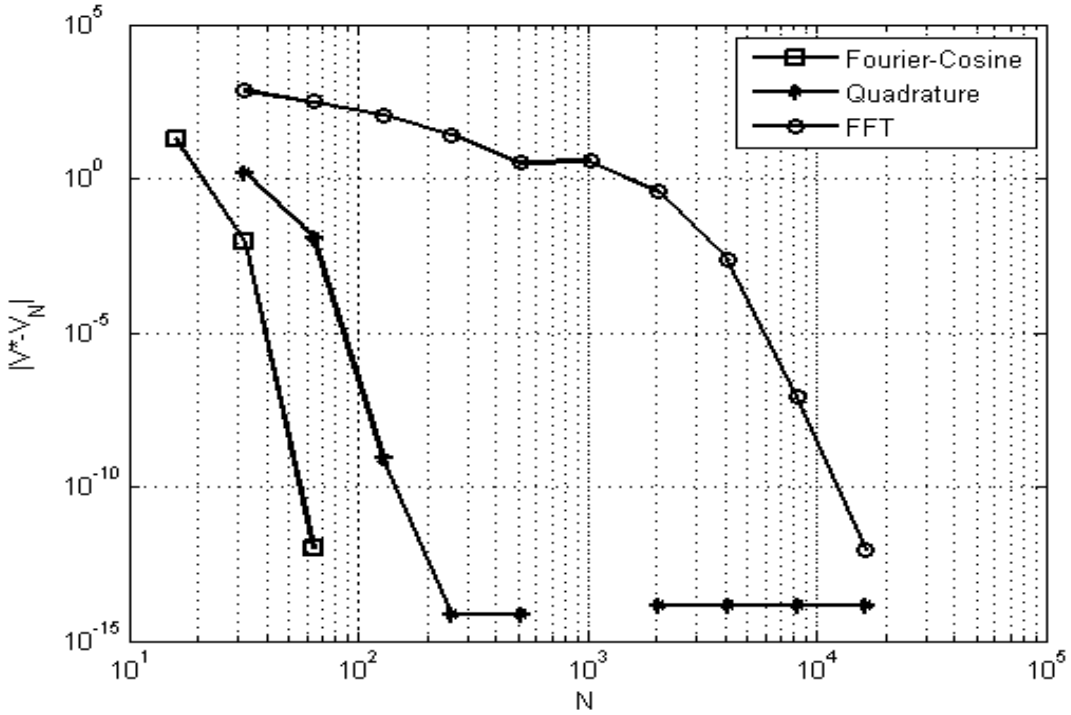


Figure : Comparison of the error convergence rates of the Fourier-Cosine (Fang & Oosterlee, 2008), fixed second order Gauss-Legendre quadrature and Carr-Madan FFT methods applied to the Heston pricing model.

How often should we re-calibrate the model?

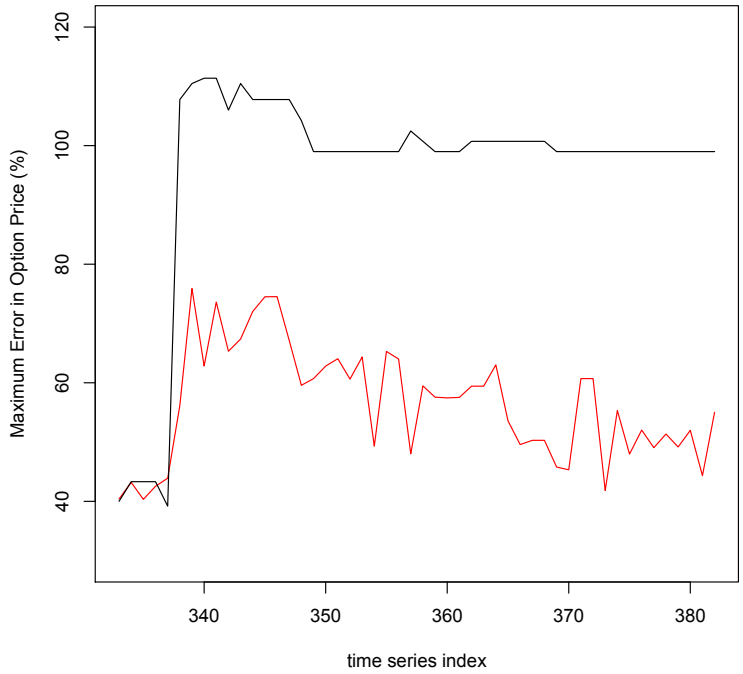


Figure : The maximum point-wise error across the volatility surface for options on ZNGA against time. The red-line shows the error resulting from calibration of the Bates model every 30 seconds versus calibrating at the start of the period (black-line).

Definition (Constrained Least Squares Optimization)

$$\min_{\mathbf{p}} f(\mathbf{p}) = \left(\sum_{i=1}^{|\mathcal{K}|} \sum_{j=1}^{|\mathcal{T}|} w_{ij} [V(S, K_i, T_j; \mathbf{p}) - \hat{V}_{ij}]^2 \right)^{1/2},$$

subject to $a_i \leq p_i \leq b_i$ *and* $2\kappa\theta - \sigma^2 > 0$ *(the Feller condition).*

- $\hat{V}_{ij}$ *denotes the quoted mid-price of the option with an underlying price* S*, maturity* T_i *and strike* K_j*.*
- *Choose* $w_{ij} = 1/(\hat{V}_{ij}^{ask} - \hat{V}_{ij}^{bid})$

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[1].
2. Call an iterative constraint based optimizer: specify tolerances on various errors, e.g. relative function error or relative solution changes. The *NLoptr* package provides various local optimizers.
3. *Performance penalty using R*: can not quickly calibrate the model which hinders modeling, testing and productionization.

[1] D. Ardia, J. David, O. Arango and N.D.G. Gomez, Jump-Diffusion Calibration using Differential Evolution, Willmott Magazine, 55 Sep, 76-79, 2011.

Chain data

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

$S[i]$: Underlying asset price
$K[i]$: Strike price
$T[i]$: Maturity
$\hat{V}[i]$: Market price

Sequential implementation of the calibration program in R

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

Table : Performance results for the R code in seconds. Each column represents a different option chain.

Sequential ErrorFunction(p)

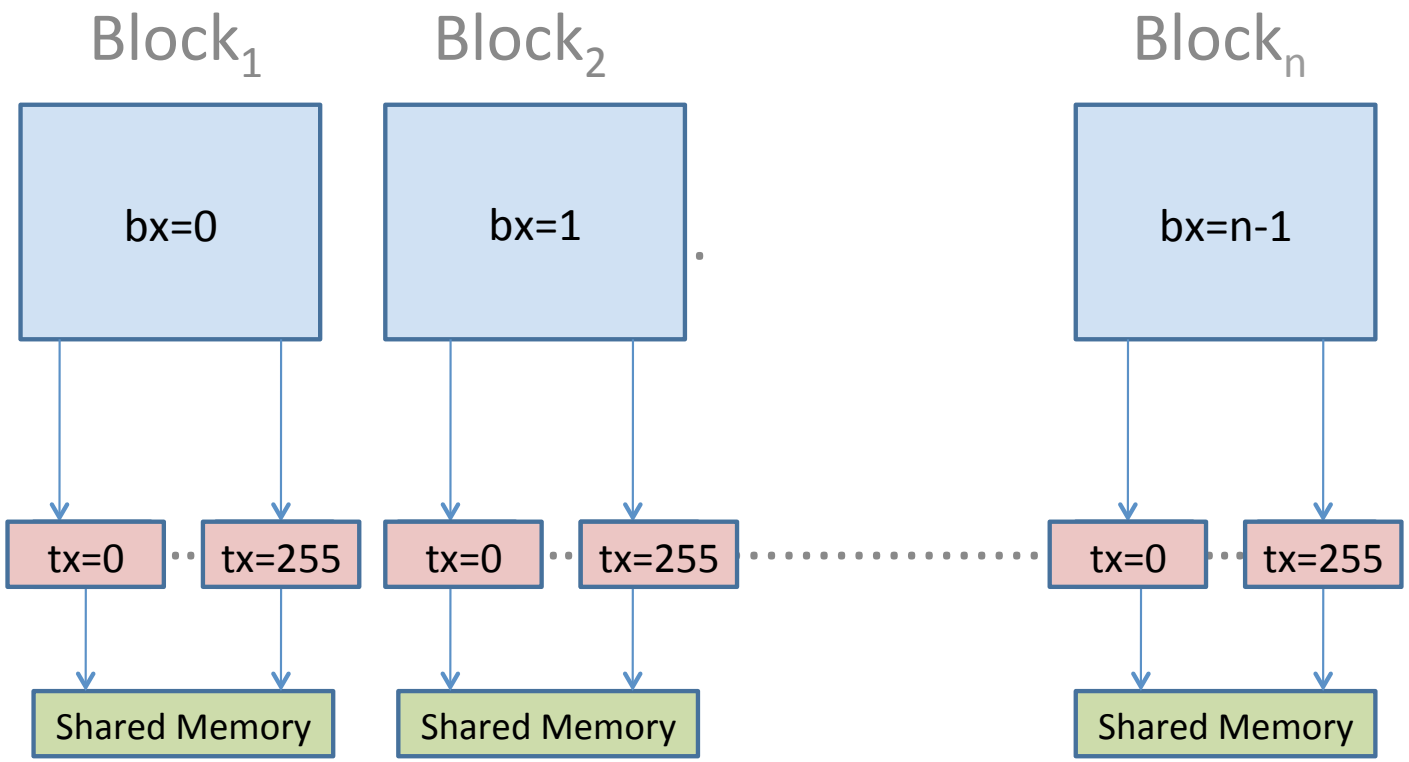
1: rmse $\leftarrow 0$
2: **for** $i = 0$ to $n - 1$ **do**
3: $V \leftarrow$ Price $(S[i], K[i], T[i], p)$
4: $diff \leftarrow \hat{V}[i] - V$
5: $rmse \leftarrow rmse + diff \times diff$
6: **end for**
7: rmse $\leftarrow$ SQRT(rmse$/n$)
8: **return** rmse

Sequential implementation of the calibration program in C

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

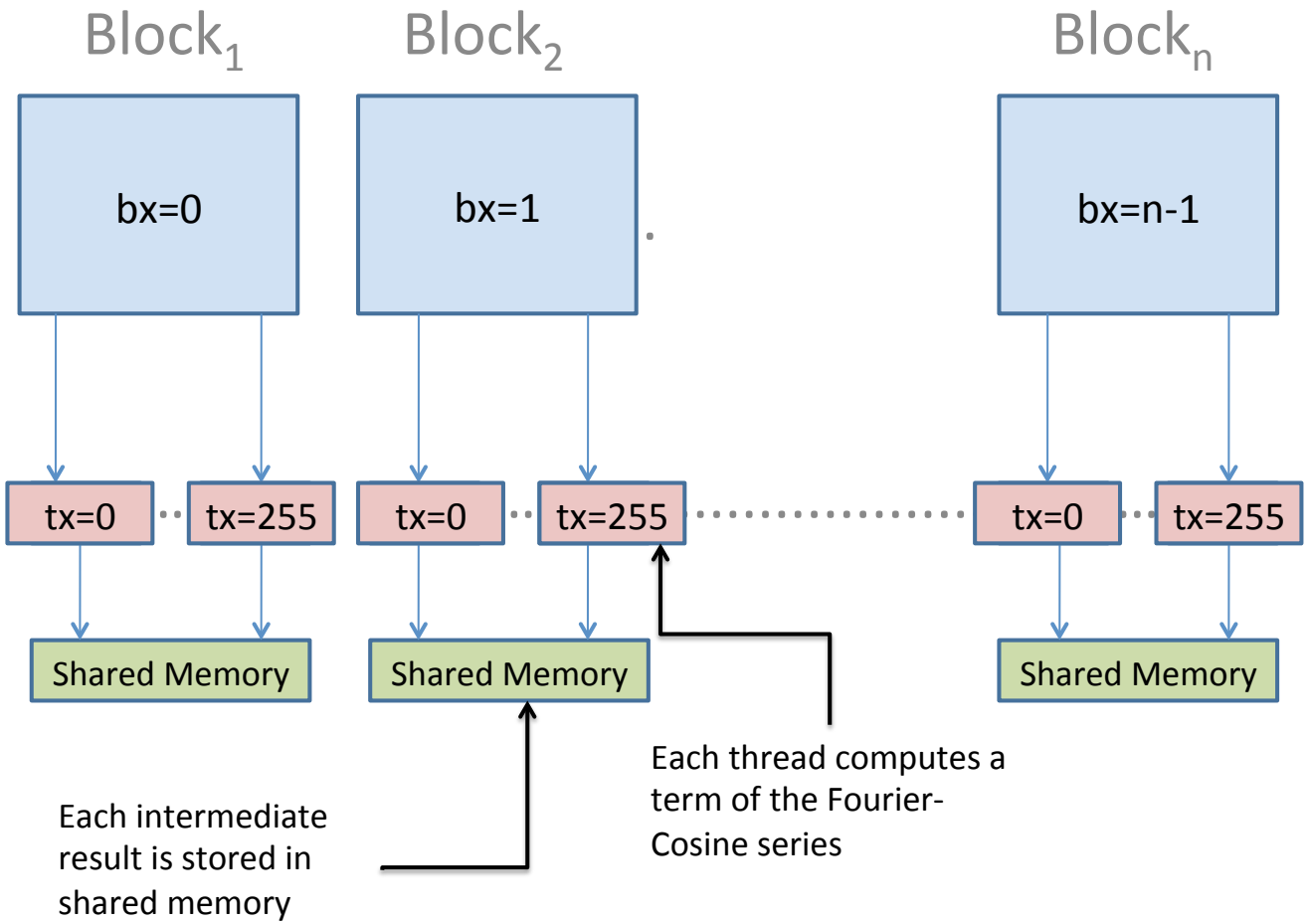
Table : Performance results for the C code in seconds. Based on Intel Core i5 processor.

GPU Implementation



GPU Implementation

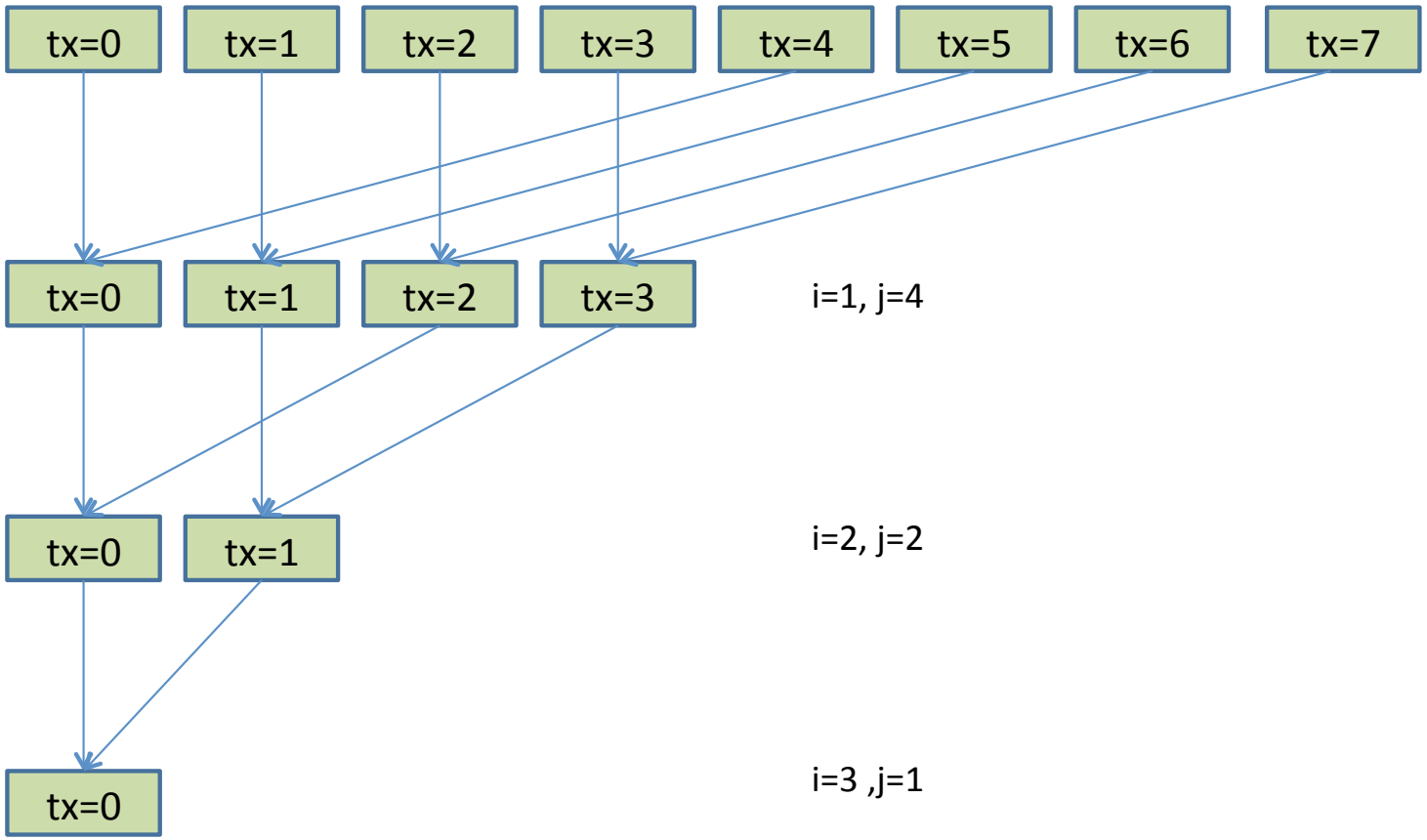
Each option in the chain is mapped to a thread block



Parallel-Fourier-Cosine(p)

1: *shared memory smem*[]
2: $tx \leftarrow threadIdx.x$
3: $bx \leftarrow blockIdx.x$
4: $bd \leftarrow blockDim.x$
5: $j \leftarrow bd$
6: $smem[tx] \leftarrow$ CHARACTERISTICFUNCTION($T[bx], p$)
7: **for** $i = 1$ to $log2(bd)$ **do**
8: $\quad j \leftarrow j/2$
9: $\quad$ **if** $tx < j$ **then**
$\quad\quad smem[tx] \leftarrow smem[tx] + smem[tx + j]$
10: $\quad$ **end if**
11: **end for**
12: **if** $tx = 0$ **then**
13: $\quad V[bx] \leftarrow K[bx] \times exp(-r0 \times T[bx] \times smem[0])$
14: **end if**
15:
16: **return** $V[bx]$

Parallel reduction using shared memory



Overview of the *gpusvcalibration* library

- The `gpusvcalibration` 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 user
- The package is designed for use with existing 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(p){
        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

	AAPL	AMZN	BP	CSCO	GOOG	MSFT
DEoptim	0.31	0.12	0.08	0.074	0.29	0.08
nloptr	0.16	0.063	0.044	0.041	0.15	0.044
ErrorFunction (ms)	1.41	0.56	0.36	0.33	1.25	0.36
Total ErrorFunction	0.42	0.17	0.11	0.1	0.38	0.11
Total Time	0.53	0.23	0.17	0.16	0.44	0.18
% ErrorFunction	79.2%	73.9%	64.7%	62.5%	86.3%	61.1%

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

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

Table : Performance results for the R code in seconds. Each column represents a different option chain.

Comparative Performance of the GPU implementation

	AAPL	AMZN	BP	CSCO	GOOG	MSFT
R/RGPU	1042x	992x	971x	933x	1024x	961x
C/RGPU	313x	297x	288x	278x	303x	297x
CGPU/RGPU	1x	1x	1x	1x	1x	1x

Table : Performance comparison of the offloaded *ErrorFunction* to a serial *R* implementation, a serial $C/C++$ implementation and a *CGPU* implementation. 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 non-convex optimization routines
- Our *gpusvcalibration* R package accelerates SV model calibration by offloading the error function on the GPUs
- Demonstrated a factor of up to 760x performance improvement on a NVIDIA Tesla K20c (Kepler architecture)

Accompanying resources

- Alpha version of R package:

```
library(devtools)
install_github("gpusvcalibration",username = "mfrdixon")
```

- Papers:
  - M.F. Dixon, S. Khan and M. Zubair, gpusvcalibration: A R Package for Fast Stochastic Volatility Model Calibration using GPUs, R/Finance, Chicago, 2014
  - M.F. Dixon, S. Khan and M. Zubair, Accelerating Option Risk Analytics in R using GPUs, Proceedings of HPC'14, Tampa, 2014.
  - M.F. Dixon and M. Zubair, Calibration of Stochastic Volatility Models on a Multi-Core CPU Cluster, In Proceedings of the Workshop on High Performance Computational Finance, SC13, November, 2013.

Summary of the core functions in the *gpusvcalibration* library

Function	Description
`Copy_Data`	Copy the chain object on to the GPU device memory
`Dealloc_Data`	Delete memory allocated on the GPU device and the host for data structures
`Error_Function`	Off-loads the weighted root mean square error calculation on to the GPU device
`Load_Chain`	Default file parser for populating a chain object
`Set_Block_Size`	Set the number of terms in the Fourier-Cosine series approximation
`Set_Model`	Set the stochastic volatility model type
`Test_Error_Function`	Calculates the weighted root mean square error and prices in R for testing purposes

Table : This table provides a summary of the core functions and interface provided for testing in the *gpusvcalibration* library.

Definition (Bates Model)

$$\frac{dS_t}{S_t} = \mu dt + \sqrt{V_t} dW_t^1 + (Y-1) S_t dN_t, \quad (1)$$

$$\frac{dV_t}{V_t} = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^2, \quad (2)$$

- *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 $\sigma_j \geq 0$.*

Definition (European Option Pricing Model)

- *Call price of a vanilla European option is*

$$C(S_0, K, \tau; \mathbf{p}) = S_0 P_1 - K \exp\{-(r-q)\tau\} P_2,$$

- *P_1 and P_2 can be expressed as:*

$$P_j = \frac{1}{2} + \frac{1}{\pi} \int_o^\infty Re \left[\frac{\exp\{-iu lnK\} \phi_j(S_0, \tau, u; \mathbf{p})}{iu} \right] du, j = 1, 2.$$

- *where the parameter set $\mathbf{p} := [\theta, \sigma, \kappa, \rho, v_0, \mu_j, \sigma_j, \lambda]$.*

Definition (Fourier-Cosine Call Price Approximation[2])

$$C(S_0, K, \tau; \mathbf{p}) \approx Ke^{-r\tau} \cdot Re\{\hat{\sum}_{k=0}^{N-1} \phi\left(\frac{k\pi}{b-a}; \mathbf{p}\right) e^{ik\pi\frac{x-a}{b-a}} U_k\},$$

- $x := ln(S_0/K)$ *is the log of moneyness*
- $\phi(w; \mathbf{p})$ *denotes the Bates characteristic function of the log-asset price,*
- U_k *are the payoff series coefficients and*
- N *denotes the number of terms in the cosine series expansion (typically 128 will suffice).*

[2] F. Fang and C. W. Oosterlee, A Novel Pricing Method for European Options based on Fourier-Cosine Series Expansions, SIAM Journal on Scientific Computing, 31(2), 2008.