Monte-Carlo Pricing under a Hybrid Local Volatility model

Sébastien Gurrieri

Mizuho International plc

GPU Technology Conference
San Jose, 14-17 May 2012

Key Interests in Finance

- Pricing of exotic derivatives
- Monte-Carlo simulations
- Local Volatility model for Foreign Exchange Rates (FX)
- Hybrid with Interest Rate models (IR)

Introduction

Key Interests in Finance

- Pricing of exotic derivatives
- Monte-Carlo simulations
- Local Volatility model for Foreign Exchange Rates (FX)
- Hybrid with Interest Rate models (IR)

Key Interests in CUDA

- High-dimensional Monte-Carlo simulations
- Texture memory (layered)

Plan of the talk

- Description of the problem and motivation for parallel programming and textures

Plan of the talk

- Description of the problem and motivation for parallel programming and textures
- Outline of implementation in CUDA

Plan of the talk

- Description of the problem and motivation for parallel programming and textures
- Outline of implementation in CUDA
- Numerical tests
  - Call/Put options in Local Volatility (LV) model
  - Exotic swaps in LV model
  - Exotic swaps in Hybrid LV model

Plan of the talk

- Description of the problem and motivation for parallel programming and textures
- Outline of implementation in CUDA
- Numerical tests
  - Call/Put options in Local Volatility (LV) model
  - Exotic swaps in LV model
  - Exotic swaps in Hybrid LV model
- Conclusion on performance and use in industry

Description of the problem

The product: Power-Reverse Dual Coupon Swap (PRDC)

Description of the problem

The product: Power-Reverse Dual Coupon Swap (PRDC)

- Underlying swap: for a series of dates $0 \leq T_i \leq 30$ years

Description of the problem

The product: Power-Reverse Dual Coupon Swap (PRDC)

- Underlying swap: for a series of dates $0 \leq T_i \leq 30$ years
  - receive option on FX_i with strike K_i:

$$+max(FX_i - K_i, 0)$$

Description of the problem

The product: Power-Reverse Dual Coupon Swap (PRDC)

- Underlying swap: for a series of dates $0 \leq T_i \leq 30$ years
  - receive option on FX_i with strike K_i:

$$+max(FX_i - K_i, 0)$$

  - pay option on IR_i with strike Q_i:

$$-max(IR_i - Q_i, 0)$$

Description of the problem

The product: Exotic exercise

- Target Redemption Note (TARN) with target A

Description of the problem

The product: Exotic exercise

- Target Redemption Note (TARN) with target A
  - Monitor coupon sum

$$C_i = \sum_{k=1}^{i} max(FX_k - K_k, 0)$$

Description of the problem

The product: Exotic exercise

- Target Redemption Note (TARN) with target A
  - Monitor coupon sum

$$C_i = \sum_{k=1}^{i} max(FX_k - K_k, 0)$$

  - if $C_i > A$, cancel all remaining cash-flows

Description of the problem

The product: Main features

- Sensitive to FX smile

  $\rightarrow$ modelling of smile

Description of the problem

The product: Main features

- Sensitive to FX smile
  $\rightarrow$ modelling of smile
- Sensitive to FX-IR correlation, IR volatility
  $\rightarrow$ modelling of IR stochasticity
  $\rightarrow$ multi-factor FX-IR hybrid

Description of the problem

The product: Main features

- Sensitive to FX smile
  $\rightarrow$ modelling of smile
- Sensitive to FX-IR correlation, IR volatility
  $\rightarrow$ modelling of IR stochasticity
  $\rightarrow$ multi-factor FX-IR hybrid
- Path-dependent due to exotic exercise
  $\rightarrow$ mainly Monte-Carlo

Description of the problem

The model: Dupire's Local Volatility [1]

- Diffusion with volatility $\sigma(t, FX)$

$$\frac{dFX}{FX} = (r_d - r_f)dt + \sigma(t, FX)dW$$

  - r_d is the domestic interest rate
  - r_f is the foreign interest rate
  - dW is a Brownian motion

Description of the problem

The model: Calibration to the market of FX options

Description of the problem

The model: Calibration to the market of FX options

- Market characterized by implied volatility $\theta(t, FX)$
  - $\rightarrow$ once differentiable in t, twice in FX (ideally)
  - $\rightarrow$ satisfies non-arbitrage conditions (ideally)

Description of the problem

The model: Calibration to the market of FX options

- Market characterized by implied volatility $\theta(t, FX)$
  - $\rightarrow$ once differentiable in t, twice in FX (ideally)
  - $\rightarrow$ satisfies non-arbitrage conditions (ideally)

- Model fits the market *exactly* for Dupire's condition

$$\sigma^2(t, FX) = f\left(\frac{\partial\theta}{\partial t}, \frac{\partial\theta}{\partial FX}, \frac{\partial^2\theta}{\partial FX^2}\right)$$

Description of the problem

The model: Sampling the volatility

- LV matrix defined as

$$\sigma_{ni} = \sigma(t_n, FX_i)$$

Description of the problem

The model: Sampling the volatility

- LV matrix defined as

$$\sigma_{ni} = \sigma(t_n, FX_i)$$

- Typical size $\sim 200 \times 200 = 40,000$ entries

Description of the problem

The model: Sampling the volatility

- LV matrix defined as

$$\sigma_{ni} = \sigma(t_n, FX_i)$$

- Typical size $\sim 200 \times 200 = 40,000$ entries
- Bi-linear interpolation in t and FX
  - $\rightarrow$ texture memory [2]
  - $\rightarrow$ simple but lacks flexibility

Description of the problem

The model: Sampling the volatility

- LV matrix defined as

$$\sigma_{ni} = \sigma(t_n, FX_i)$$

- Typical size $\sim 200 \times 200 = 40,000$ entries
- Bi-linear interpolation in t and FX

  $\rightarrow$ texture memory [2]

  $\rightarrow$ simple but lacks flexibility

- Linear interpolation in FX at known t

  $\rightarrow$ *layered* textures

  $\rightarrow$ slightly more complicated but more flexible and/or accurate

Description of the problem

Summary

- Multi-factor and path-dependent product
  - $\rightarrow$ Monte-Carlo simulation
  - $\rightarrow$ good speed-up expected with CUDA

Description of the problem

Summary

- Multi-factor and path-dependent product
  - $\rightarrow$ Monte-Carlo simulation
  - $\rightarrow$ good speed-up expected with CUDA
- Model requires interpolation of a matrix
  - $\rightarrow$ benefit from texture memory

Description of the problem

Summary

- Multi-factor and path-dependent product
  - $\rightarrow$ Monte-Carlo simulation
  - $\rightarrow$ good speed-up expected with CUDA
- Model requires interpolation of a matrix
  - $\rightarrow$ benefit from texture memory
- Multiple cash-flows, monitoring, smile-modelling
  - $\rightarrow$ large number of time steps
  - $\rightarrow$ high-dimensional problem
  - $\rightarrow$ inline random number generation

Implementation Outline

Single-thread:

- On each path j, at each time t_n

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number
  2. transform to Gaussian, then Brownian motion increment dW_n^j

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number
  2. transform to Gaussian, then Brownian motion increment dW_n^j
  3. read previous spot FX_n^j from memory

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number
  2. transform to Gaussian, then Brownian motion increment dW_n^j
  3. read previous spot FX_n^j from memory
  4. calculate volatility σ by calling texture at (t_n, FX_n^j)

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number
  2. transform to Gaussian, then Brownian motion increment dW_n^j
  3. read previous spot FX_n^j from memory
  4. calculate volatility σ by calling texture at (t_n, FX_n^j)
  5. calculate new spot

$$FX_{n+1}^j = FX_n^j e^{(r_d - r_f - \frac{1}{2}\sigma^2)(t_{n+1} - t_n) + \sigma dW_n^j}$$

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number
  2. transform to Gaussian, then Brownian motion increment dW_n^j
  3. read previous spot FX_n^j from memory
  4. calculate volatility σ by calling texture at (t_n, FX_n^j)
  5. calculate new spot

$$FX_{n+1}^j = FX_n^j e^{(r_d - r_f - \frac{1}{2}\sigma^2)(t_{n+1} - t_n) + \sigma dW_n^j}$$

  6. calculate product(s)

Implementation Outline

Single-thread:

- On each path j, at each time t_n

1. calculate next uniform random number
2. transform to Gaussian, then Brownian motion increment dW_n^j
3. read previous spot FX_n^j from memory
4. calculate volatility σ by calling texture at (t_n, FX_n^j)
5. calculate new spot

$$FX_{n+1}^j = FX_n^j e^{(r_d - r_f - \frac{1}{2}\sigma^2)(t_{n+1} - t_n) + \sigma dW_n^j}$$

6. calculate product(s)
7. write new spot in memory

Implementation Outline

Single-thread:

- On each path j, at each time t_n
  1. calculate next uniform random number
  2. transform to Gaussian, then Brownian motion increment dW_n^j
  3. read previous spot FX_n^j from memory
  4. calculate volatility σ by calling texture at (t_n, FX_n^j)
  5. calculate new spot

$$FX_{n+1}^j = FX_n^j e^{(r_d - r_f - \frac{1}{2}\sigma^2)(t_{n+1} - t_n) + \sigma dW_n^j}$$

  6. calculate product(s)
  7. write new spot in memory
- Loop on path, then time.

Implementation Outline

Multi-thread:

- Sequential in time, parallel on paths

Implementation Outline

Multi-thread:

- Sequential in time, parallel on paths
- Grid configuration
  - 1-dimensional grid of N_{blocks} blocks
  - 1-dimensional blocks of $N_{threads}$ threads
  - $s = N_{blocks} \times N_{threads}$ = number of concurrent threads

Implementation Outline

Multi-thread:

- Sequential in time, parallel on paths
- Grid configuration
  - 1-dimensional grid of N_{blocks} blocks
  - 1-dimensional blocks of $N_{threads}$ threads
  - $s = N_{blocks} \times N_{threads}$ = number of concurrent threads
- Thread j calculates paths $j, j + s, j + 2s, etc...$

Implementation Outline

Multi-thread:

- Sequential in time, parallel on paths
- Grid configuration
  - 1-dimensional grid of N_{blocks} blocks
  - 1-dimensional blocks of $N_{threads}$ threads
  - $s = N_{blocks} \times N_{threads}$ = number of concurrent threads
- Thread j calculates paths $j, j+s, j+2s, etc...$
- Thread j must remember previous spot values for paths $j, j+s, j+2s, etc...$

  $\rightarrow$ too much for shared memory

  $\rightarrow$ store previous spot values in global memory

Implementation Outline

Multi-thread:

- Thread j:
  - calculates products on paths $j, j+s, j+2s, etc...$
  - sums them in local variable
  - writes sums in shared memory

Implementation Outline

Multi-thread:

- Thread j:
  - calculates products on paths $j, j+s, j+2s, etc...$
  - sums them in local variable
  - writes sums in shared memory
- Synchronize

Implementation Outline

Multi-thread:

- Thread j:
  - calculates products on paths $j, j+s, j+2s, etc...$
  - sums them in local variable
  - writes sums in shared memory
- Synchronize
- In each block:
  - one thread is attributed to each product
  - accumulates in a local variable all thread sums for this product
  - writes "block-partial" sum in global memory

Implementation Outline

Multi-thread:

- Global memory contains "block-partial" sums for each product, each block, at each time

Implementation Outline

Multi-thread:

- Global memory contains "block-partial" sums for each product, each block, at each time
- Transfer to host

Implementation Outline

Multi-thread:

- Global memory contains "block-partial" sums for each product, each block, at each time
- Transfer to host
- On host, sum results of all blocks.

Implementation Outline

Multi-thread: remark on random number generation

- typical number of times: 500

Implementation Outline

Multi-thread: remark on random number generation

- typical number of times: 500
- typical number of factors: 2, but easily going to 3 and more

Implementation Outline

Multi-thread: remark on random number generation

- typical number of times: 500
- typical number of factors: 2, but easily going to 3 and more
- typical number of simulations: 100K, but may want more

Implementation Outline

Multi-thread: remark on random number generation

- typical number of times: 500
- typical number of factors: 2, but easily going to 3 and more
- typical number of simulations: 100K, but may want more
- global generation requires minimum global memory

$$500 \times 2 \times 100,000 \times 4 = 400MB$$

Implementation Outline

Multi-thread: remark on random number generation

- typical number of times: 500
- typical number of factors: 2, but easily going to 3 and more
- typical number of simulations: 100K, but may want more
- global generation requires minimum global memory

$$500 \times 2 \times 100,000 \times 4 = 400MB$$

- cannot run on all devices, too restrictive for pratical applications

  $\rightarrow$ use inline generation

Texture:

- Desired interpolation

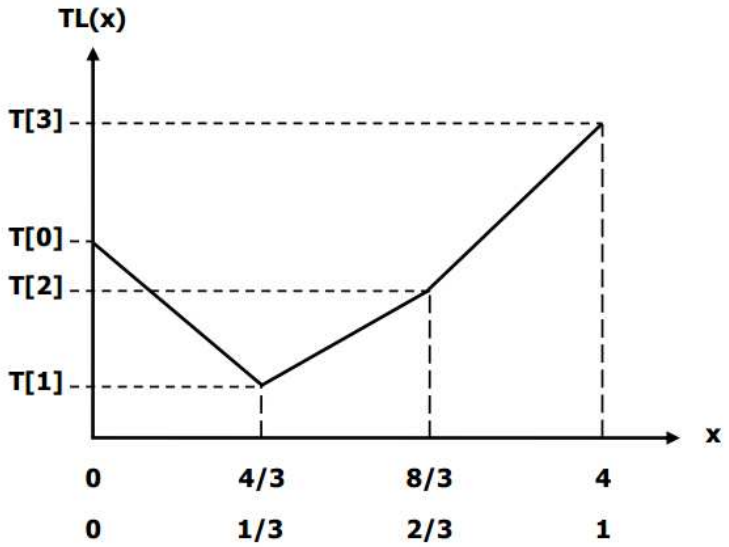


Figure E-3. One-Dimensional Table Lookup Using Linear Filtering

Texture:

- Texture interpolation

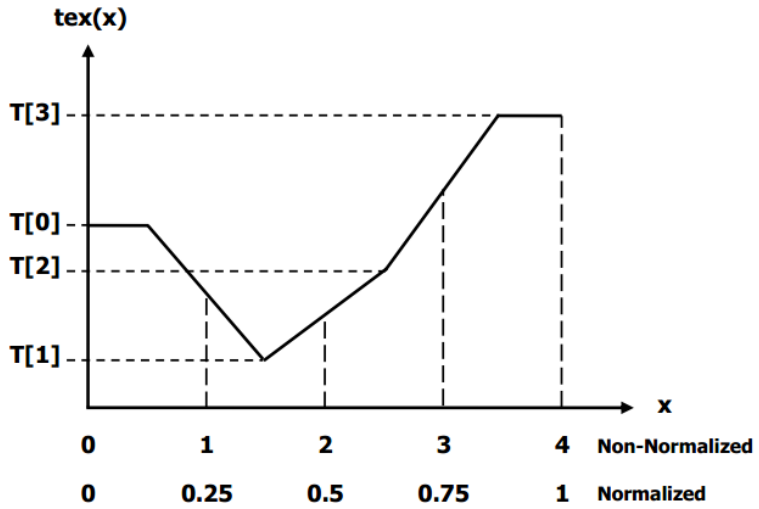


Figure E-2. Linear Filtering of a One-Dimensional Texture of Four Texels in Clamp Addressing Mode

Implementation Outline

Texture:

- Linear rescaling is required

Implementation Outline

Texture:

- Linear rescaling is required
- Given spots $FX_0, FX_1, \cdots FX_{M-1}$, volatilities $\sigma_0, \sigma_1, \cdots \sigma_{M-1}$

Implementation Outline

Texture:

- Linear rescaling is required
- Given spots $FX_0, FX_1, \cdots FX_{M-1}$, volatilities $\sigma_0, \sigma_1, \cdots \sigma_{M-1}$
- The volatility at any spot FX is

$$\sigma(FX) = tex(\alpha FX + \beta)$$

Implementation Outline

Texture:

- Linear rescaling is required
- Given spots $FX_0, FX_1, \cdots FX_{M-1}$, volatilities $\sigma_0, \sigma_1, \cdots \sigma_{M-1}$
- The volatility at any spot FX is

$$\sigma(FX) = tex(\alpha FX + \beta)$$

$$\text{with} \begin{cases} \alpha = \dfrac{M-1}{M(FX_{M-1} - FX_0)} \\ \beta = \dfrac{1}{M}\left(\dfrac{1}{2} - (M-1)\dfrac{FX_0}{FX_{M-1} - FX_0}\right) \end{cases}$$

Implementation Outline

Texture:

- Bi-linear interpolation with standard texture

$$\sigma(t, FX) = \mathrm{tex2D}(\alpha FX + \beta, \gamma t + \delta)$$

Implementation Outline

Texture:

- Bi-linear interpolation with standard texture

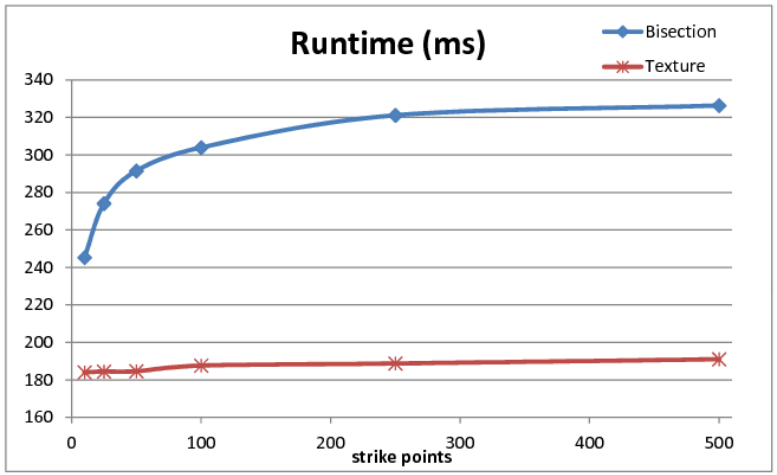
$$\sigma(t, FX) = \mathrm{tex2D}(\alpha FX + \beta, \gamma t + \delta)$$

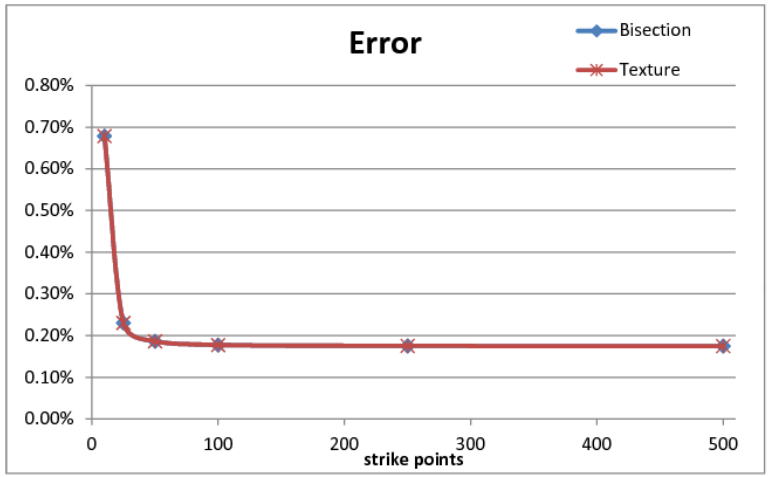
- Linear interpolation with layered texture

$$\sigma(t_n, FX) = \mathrm{tex1DLayered}(\alpha FX + \beta, n)$$

Vanilla Options:

- Performance of the texture (500 time steps, 500K simulations)

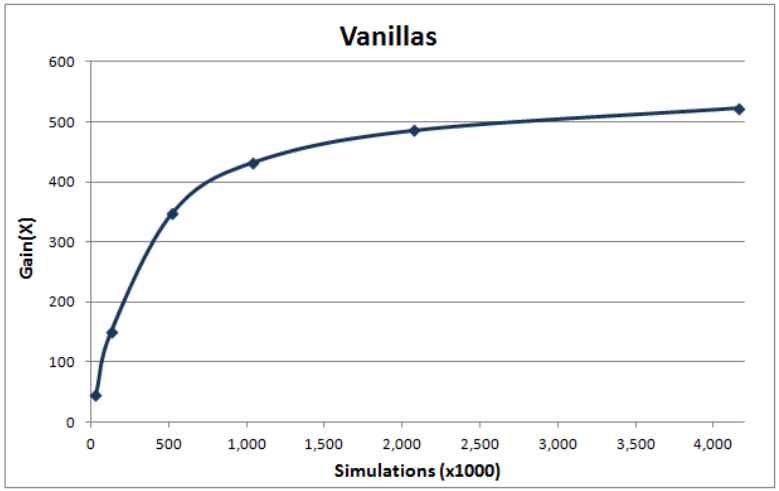




- 50% $\sim$ 70% speed gains with texture
- good accuracy of the texture interpolation
- $\sim$ 100 points sufficient

Vanilla Options:

- Gain (single thread vs. GTX 460)



Exotic Swap (one factor):

- Additional state variable on path j

$$C_i^j = \sum_{k=1}^{i} max(FX_k^j - K_k, 0)$$

$\rightarrow$ one more read/write access from global memory

Exotic Swap (one factor):

- Additional state variable on path j

$$C_i^j = \sum_{k=1}^{i} max(FX_k^j - K_k, 0)$$

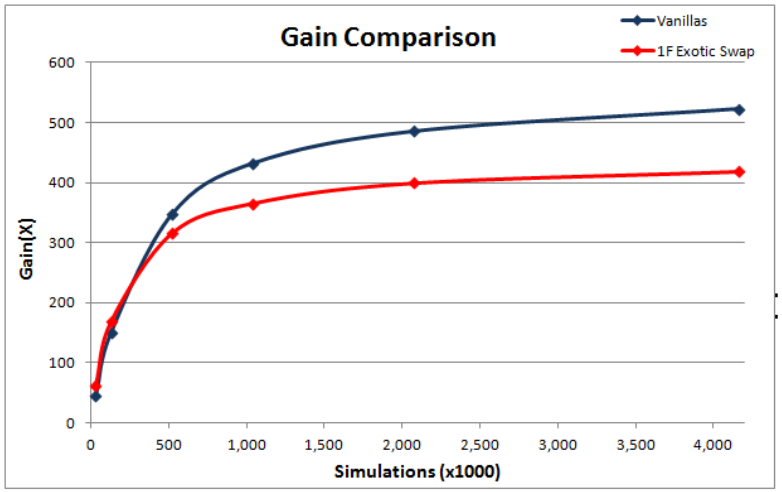
  $\rightarrow$ one more read/write access from global memory

- Product calculated only at cash-flow times (at most 120)

  $\rightarrow$ less operations than for vanillas (500)

Exotic Swap (one factor):

- Gain (single thread vs. GTX 460)



Exotic Swap (hybrid 2F):

- r_d follows Hull-White model

$$dr_d = (\theta - ar_d)dt + \sigma_r dW_r$$

Exotic Swap (hybrid 2F):

- r_d follows Hull-White model

$$dr_d = (\theta - ar_d)dt + \sigma_r dW_r$$

- it has a correlation ρ with FX

$$\begin{aligned} dW_{FX} &= g_1\sqrt{dt} \\ dW_r &= (\rho g_1 + \sqrt{1-\rho^2}g_2)\sqrt{dt} \end{aligned}$$

Exotic Swap (hybrid 2F):

- r_d follows Hull-White model

$$dr_d = (\theta - ar_d)dt + \sigma_r dW_r$$

- it has a correlation ρ with FX

$$\begin{aligned} dW_{FX} &= g_1\sqrt{dt} \\ dW_r &= (\rho g_1 + \sqrt{1-\rho^2}g_2)\sqrt{dt} \end{aligned}$$

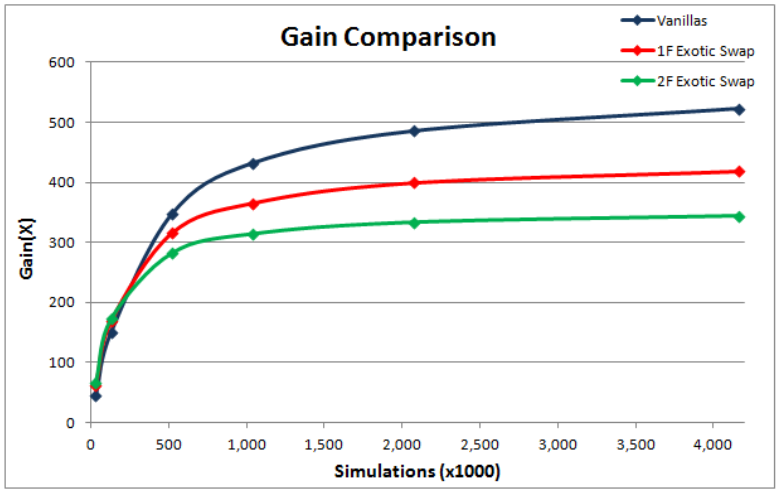
- 2 additional state variable on path j

$$r^j, \quad e^{\int_0^T r^j dt} \text{ (numeraire)}$$

$\rightarrow$ two more read/write accesses to global memory

Exotic Swap (hybrid 2F):

- Gain (single thread vs. GTX 460)



Conclusion

- Extention to 3F, barriers

  $\rightarrow$ very similar to 2F, TARN

  $\rightarrow$ should not be a problem

Conclusion

- Extention to 3F, barriers

  $\rightarrow$ very similar to 2F, TARN

  $\rightarrow$ should not be a problem

- Extention to callables

  $\rightarrow$ Longstaff-Schwartz not entirely parallel

  $\rightarrow$ should not be a problem but gains may be lower

  $\rightarrow$ use Malliavin calculus? (Abbas-Turki, GTC 2010)

Conclusion

- Large gains on GTX 460 and for realistic products and pricing configurations

Conclusion

- Large gains on GTX 460 and for realistic products and pricing configurations
- Possibility to run more simulations

  $\rightarrow$ more accurate Greeks

  $\rightarrow$ more efficient risk management

Conclusion

- Large gains on GTX 460 and for realistic products and pricing configurations
- Possibility to run more simulations

  $\rightarrow$ more accurate Greeks

  $\rightarrow$ more efficient risk management
- Value-at-Risk and Potential Exposure calculations possible without approximations

Conclusion

- Large gains on GTX 460 and for realistic products and pricing configurations
- Possibility to run more simulations

  $\rightarrow$ more accurate Greeks

  $\rightarrow$ more efficient risk management
- Value-at-Risk and Potential Exposure calculations possible without approximations
- Large number of scenario testing possible on exotic portfolios

Disclaimer

This publication has been prepared by Sebastien Gurrieri of Mizuho International plc solely for the purpose of presentation at this conference. The opinions expressed in this presentation are those of the author and do not necessarily reflect the view of Mizuho International plc, which is not responsible for any use which may be made of its contents.

It is not, and should not be construed as, an offer or solicitation to buy, or sell, any security, or any interest in a security or enter into any transaction. This publication may include details of instruments that have not been issued. There is no guarantee that such instruments will be issued in the future.

This publication has been prepared solely from publicly available information. Information contained herein and the data underlying it have been obtained from, or based upon, sources believed by the author to be reliable. However, no assurance can be given that the information, data or any computations based thereon, is accurate or complete. Opinions stated in this report are subject to change without notice.
There are risks associated with the financial instruments and transactions described in this publication. Investors should consult their own financial, legal, accounting and tax advisors about the risks, the appropriate tools to analyse an investment and the suitability of an investment in their particular circumstances. Mizuho International plc is not responsible for assessing the suitability of any investment. Investment decisions and responsibility for any investments is the sole responsibility of the investor. Neither the author, Mizuho International plc nor any affiliate accepts any liability whatsoever with respect to the use of this report or its contents.

References

1. B. Dupire, "Pricing with a smile," Risk 7, pp. 18-20, Jan. 1994.
2. A. Bernemann, R. Schreyer and K. Spanderen, "Accelerating Exotic Option Pricing and Model Calibration Using GPUs", Working Paper, Feb. 2011.
3. http://sebgur.fr/sgdev.html