GTC2015 | 3/17/2015

CORPORATE & INVESTMENT BANKING, PRIVATE BANKING,
ASSET MANAGEMENT, SECURITIES SERVICES

GLOBAL BANKING & INVESTOR SOLUTIONS DIVISION

A TRUE STORY: GPU IN PRODUCTION FOR INTRADAY RISK CALCULATIONS

Régis FRICKER

Regis.fricker@sgcib.com

GPU TECHNOLOGY CONFERENCE

BUILDING TOGETHER

TEAM SPIRIT SOCIETE GENERALE

CONTENTS

PROBLEMATIC

A. OUR PROBLEM

B. PARALLELIZATION OF A MONTE CARLO SCHEME

SOLUTION

A. SOLUTION CHALLENGES

B. HOW TO USE GPU IN C# ?

C. HOW TO USE GPU IN A FINANCE COMPUTE FARM ?

IN PRACTICE

A. PROJECT MANAGEMENT

B. RAW PERFORMANCES

C. RISK ENGINE PERFORMANCES

D. BREAKING CHANGE THINKING

PROBLEMATIC

OUR PROBLEM

- What do **traders** need?

Fast prices:
To answer client request rapidly.

Accurate prices:
Require a lot of computation time.

- What do **managers** need?

Reduce costs:
Reduce computation ressources.

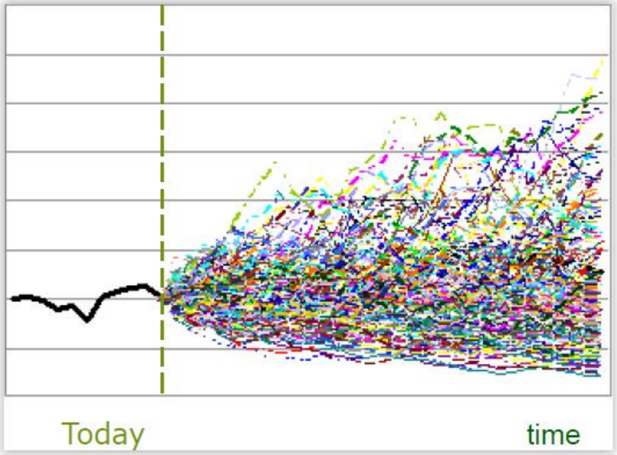
Control risks:
Even more computation time (more and more).

- Most importantly, what do **clients** need ?

Efficient service:
Fast answer to requests

Competitive prices:
Complex model. High computation time.

PARALLELIZATION OF A MONTE-CARLO SCHEME



- Definition

$$\pi = \frac{1}{Path} \sum_{i=0}^{Path} PayOff\left(\left\{S^i_{T_j}\right\}_{j\in[1..N]}\right)$$

$$S_{T_{i+1}} = Transition_i\left(S_{T_i}, W\right)$$

- Simulation $S^j_{T_i}$
  - Transition function doesn’t depend on path.
  - Two nested loops: one with respect to time and one to path.
  - Parallelism on path loop because Path >> N

- PayOff function doesn’t depend on path
  - Parallelism on path loop

SOLUTION

SOLUTION CHALLENGES

- Current pricing ecosystem
  - Risk engine is fully written in C#
  - CPU Compute Farm.

- Objective
  - Use GPU and SIMD instruction in C#.
  - Introduce GPU servers in Compute Farm.
  - Reduce latency by a factor 30.
  - Reduce compute costs of the Farm.
  - Ensure overall profitability (hardware and maintainability over time).

ALTIMESH HYBRIDIZER (1/2)

- External tool provided by Altimesh

- Writing and maintaining one single code in C#.
- Generating readeable source code for:
  - CUDA
  - C++/OMP
  - C++/AVX
- C# inheritance are handled by Hybridizer.
- Hybridizer offers extensibility framework to allow usage of platform-specific features (shared memory, fast math, libraries, etc).
- Easy to call behind C#:
  - DllImport to call native dll.
  - Data marshalling are handled by Hybridizer.

One code, 3 runners (C#, AVX, CUDA), same numerical results

ALTIMESH HYBRIDIZER (2/2)

- Hybridizer is not a magic wand.

- Some C# features are not handled:
  - No allocation inside a kernel.
  - very limited runtime support (no collection)

- Loop parallelization is not automatic.

- Sequential pattern is not automatically changed to parallel pattern.

MC framework must be adapted to satisfy these constraints and map on work distribution concepts.

NEW MONTE-CARLO FRAMEWORK

- Thinking parallel not sequential.

- Back to basics:
  - Memory accesses (coalescence, memory type).
  - Memory allocation.

- Pricing memory footprint is adjustable.

- Model and Payoff implementation are hardware independent.
  - Everyone can add a model or a payoff without Cuda knowledge.

FINANCE DISTRIBUTED CALCULATION SCHEME

- Database for market data, deals information and pricing results.

- CPU compute farm:
  - Each server has 2 bi-CPU (8 cores by CPU).

- Each core of CPU compute farm:
  - Load one deal.
  - Load market data.
  - Price this deal.
  - Upload result.

- IBM Platform Symphony solution is used as grid middleware.

GPU SERVERS

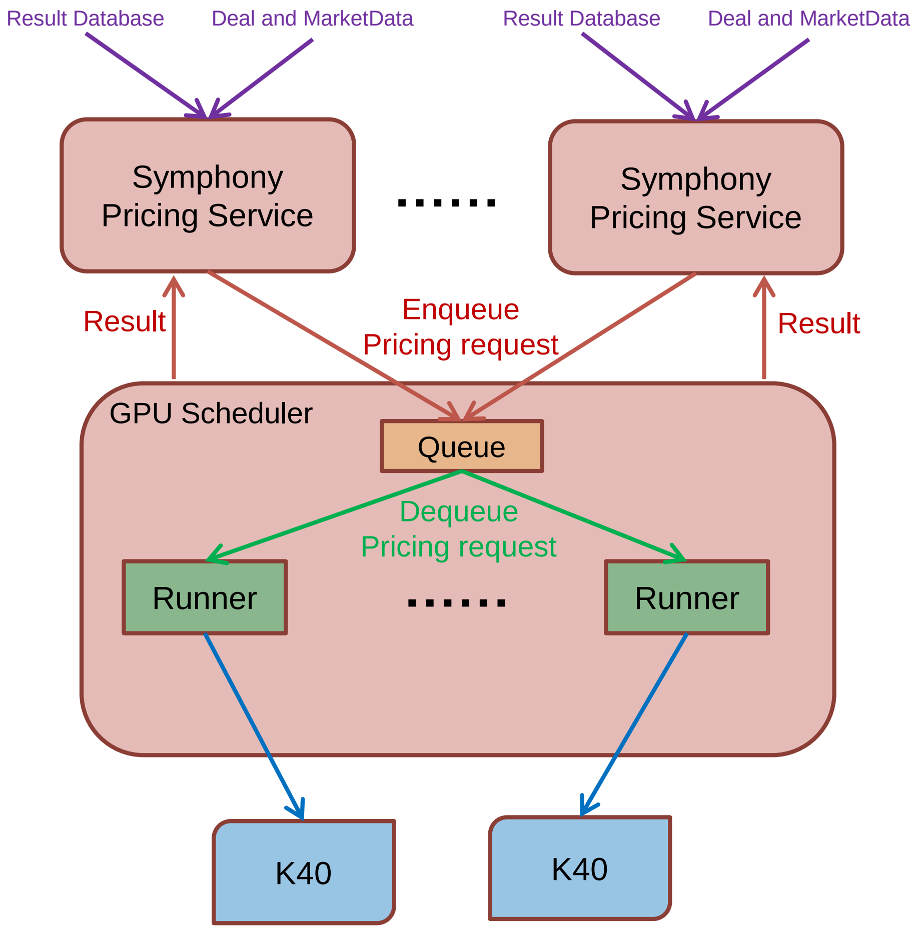
- GPU server contains:
  - 1 bi-CPU (8 cores by CPU).
  - 2 K40.

- GPU server price = 1.5 x CPU server price.
  - **Pricing on GPU must be accelerated by 3 to be profitable.**

- GPU are not handled properly by Symphony

- NVidia limitation in multi process context:
  - Each process have its own Context. Around 80Mo by process and card.
  - Each process are independent. How to manage GPU memory footprint ?

GPU SCHEDULER

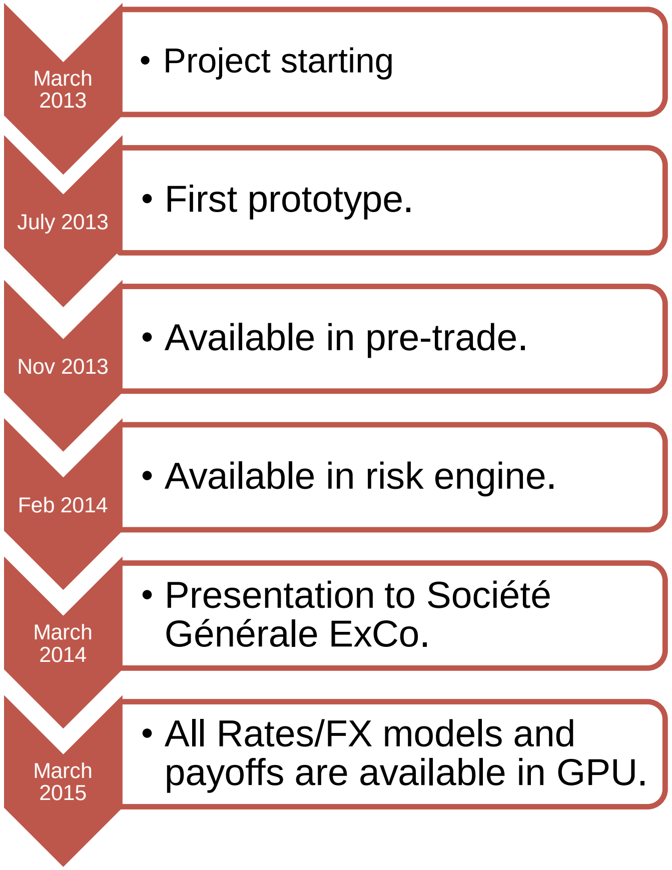


- One GPU scheduler by server.
  - One context by card.
  - Easy to manage GPU memoryfootprint

- Multithreading and Stream.

IN PRACTICE

PROJECT MANAGEMENT



March 2013
- Project starting

July 2013
- First prototype.

Nov 2013
- Available in pre-trade.

Feb 2014
- Available in risk engine.

March 2014
- Presentation to Société Générale ExCo.

March 2015
- All Rates/FX models and payoffs are available in GPU.

- 4 people:
  - 2 on Monte-Carlo framework.
  - 1 on GPU scheduler.
  - 1 on risk engine integration.

RAW PERFORMANCES (1/2)

- Rewritten C# version is twice faster than legacy code.

- Configuration :
  - Intel Xeon E5-1620 @ 3.60Hz (8 cores with hyperthreading)
  - One K40.
- Product:
  - Call on mean price with a 2 factor model.
  - Nb time step: 250.
  - Nb paths: 300 000.
- Single price:

	Single Thread C#	8 threads C#	Single thread AVX	8 thread AVX	GPU
Time	19.908	5.218	8.931	3.65	0.239
Gain	1.0	3.8	2.2	5.5	83.3

RAW PERFORMANCES (2/2)

- Workload test:
  - Launch 8 processes (1 by core).
  - Each process price 10 times the same product.
  - ➢ 80 prices are done.

	C#	AVX	GPU
Time	256	176	15
Gain	1.0	1.5	17.1

- Hardware ressources are saturated during this test.
- GPU usage indicators:
  - GPU utilisation: 99%
  - Power: 150W / 235W.
  - Memory usage peak: 11Go/12Go

RISK ENGINE PERFORMANCES

- Cores to manage a specific Book are divided by 10.

- Pricing time behind the Risk Engine is not only MC time:
  1. Time to load Deal info and Market Data.
  2. Model calibration time.
  3. Monte-Carlo time.
  4. Time to upload result.

- On GPU, Monte-Carlo is not a problem anymore.
  - ➢Other tasks becomes significant and must be optimized.

- In the current setup, GPU are not financially interesting when Monte-Carlo time is less then one third of total time.

BREAKING CHANGE THINKING

- At Société Générale, GPU is now synonymous with performance and efficiency:
  - 2013 : a client request for a very sophisticated product 5 min
  - 2014 : same request 8s

- GPU is not scary anymore
  - no longer reserved to a small expert community

- Think parallel, not sequential.
  - Every new algorithm should be thought in terms of parallel execution

CONCLUSION

- Thank you.
- Questions.