F ountainhead

GPUs in Finance

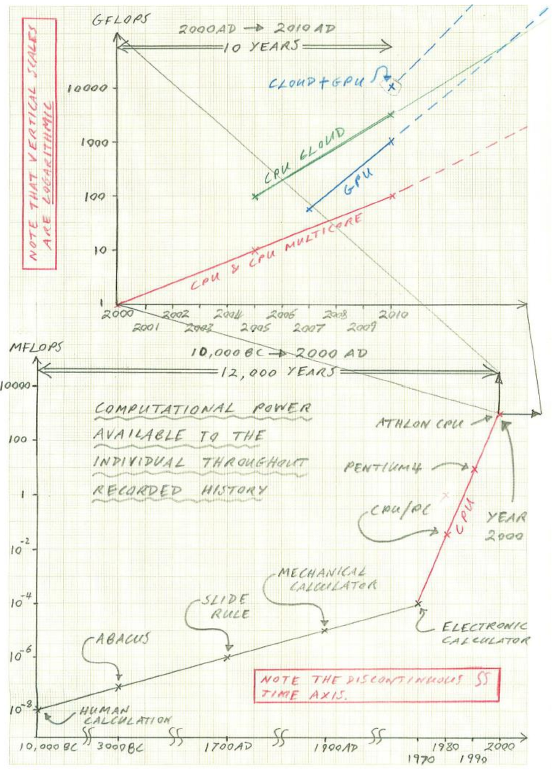
~ An Overview (with some Monte-Carlo thrown in!) ~

Andrew Sheppard & Enzo Alda

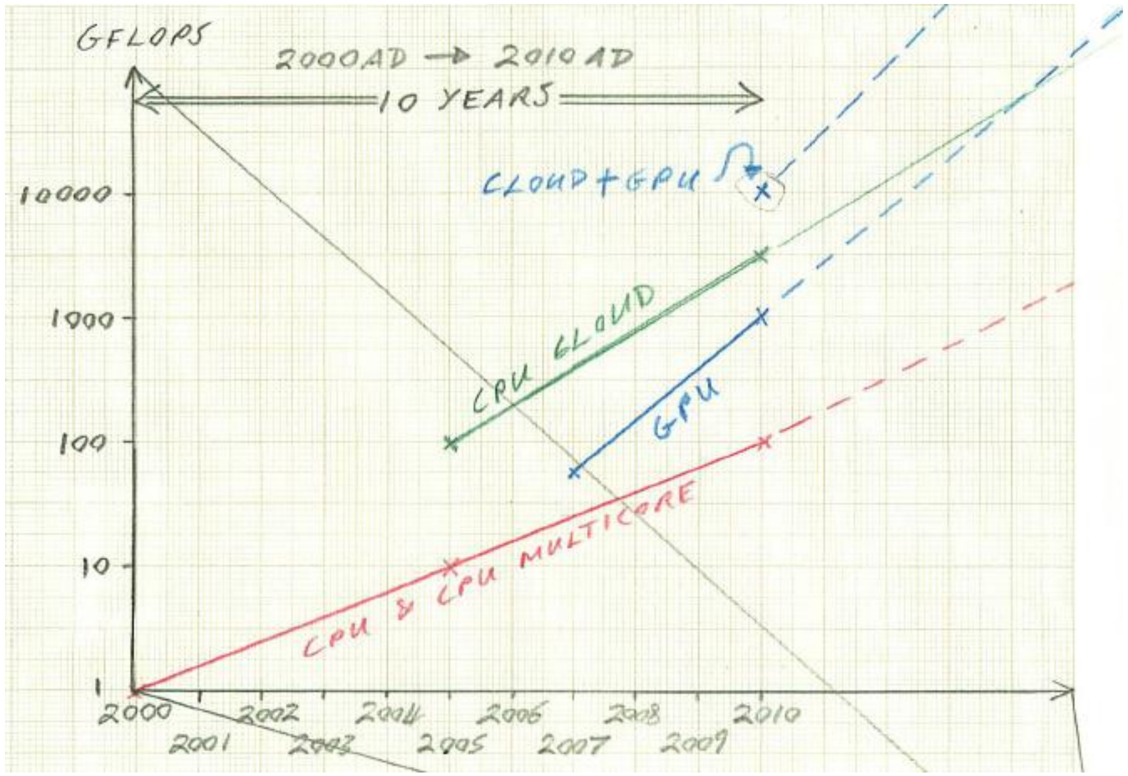
QCon, New York City, 19th June 2012

Fountainhead

Computational power throughout history



Computational power ~ let' s zoom in to the last 10 years



F ountainhead

Outline for the Talk

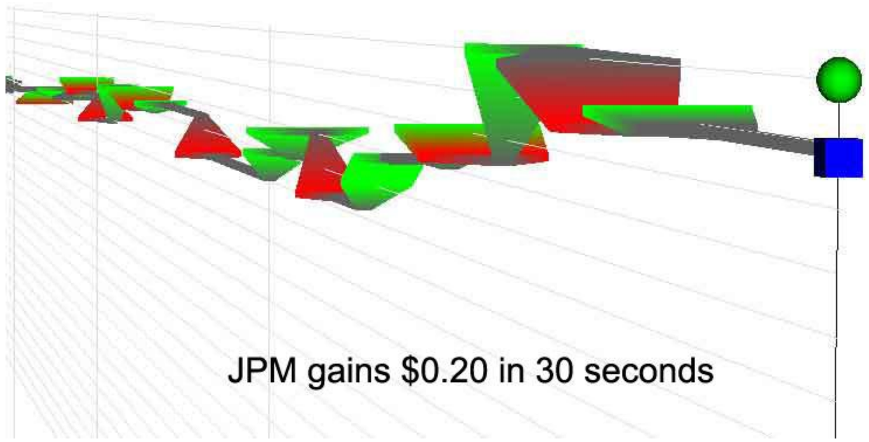
The talk is in two parts:

A. What's driving GPU adoption in finance?
   - Data.
   - Analysis.
   - Speed.
   - Application areas.

B. Overview of Monte-Carlo on GPUs
   - CUDA Thrust.
   - Towards full Monte-Carlo in a single line (well, almost!).

F ountainhead

~ A. GPUs for Finance ~



F ountainhead

What’s driving GPU adoption?

Very much like what drives all of finance and markets …

Greed and Fear!

(PROFIT and LOSS)
(OPPORTUNITY and RISK)

F ountainhead

Greed

Greed basically comes down to two things:

1) Doing things better, faster, and cheaper than the next guy.

2) Aggressively pursuing profitable opportunities.

GPUs can help with both.

Fear

Financial institutions fear two things above all else:

1) Losing money! Which means measuring and controlling risks (market risk, operational risk, etc.).

2) Complying with regulations, because failing to do so can put you out of business. And there are new and wide-ranging regulatory mandates.

GPUs can help with both.

F ountainhead

Speed, Speed & Speed

- In our business, time really is money!
- Three most important things about trading: speed x 3.
- Nothing so impresses traders as ***SPEED.***
- Imagine pricing & structuring deals quicker than others.
- Imagine risk going from overnight to real-time.
- Low latency development (iterate quicker, faster, better).
- 24hrs @
  - x10 speedup --> 2.4 hours
  - x100 speedup --> 15 minutes
  - x1000 speedup --> 90 seconds

Data, Data & Data

• Storage is (almost) free.
• Bandwidth is (almost) free (i.e. moving data around is free; but can be costly in time; co-locate compute with data).
• Data growth in the enterprise and in finance increasing.
• Electronic & high-frequency trading is exploding.
• Making sense of the data -- how to convert data into actionable knowledge?
• Number crunching and complex event processing.
• Visualization.

F ountainhead

Real-time, Real-time & Real-time

• Everything is moving to real-time.
• Everything is moving towards continuous processing.
• Everything is moving towards mobility (anywhere, anytime).
• Data & processing power must be brought together.
• GPUs co-locate data with number crunching.

F ountainhead

10x, 10x, 10x, 10x, 10x

- Rule of 10 (10x better or/and 10x cheaper)
- 10x speedup.
- 10x cheaper.
- 10x less space.
- 10x less power.
- 10x less cooling (related to power).

GPUs Getting Faster

- GPUs are getting faster (512 cores).
- CPUs are getting faster (2, 4, 8 & 12 cores).
- Memory transfer: ~25GB/s CPU vs. 150GB/s (RAM).
- GPUs have the advantage in many areas.
- Gap between GPUs and CPUs is widening for pure number crunching.
- But GPU is not a replacement for CPU. Rather, the GPU is a co-processor that augments the CPU with massive amounts of parallel number crunching capability.

F ountainhead

Applications in Finance

- Pricing, especially of complex assets.
- Risk analysis. (Overnight to real-time risk.)
- Algorithmic trading. (Pre/post trade analysis.)
- High-frequency trading. (Complex event processing).
- Tick data. (Added-value data feeds in real-time).
- Data mining. (Machine learning.)
- Trading strategy prospecting. (Backtesting too.)
- Data visualization. (Making sense of it all.)
- … anything that needs to crunch numbers, or process vast amounts of data … fast!

F ountainhead

Applications - Visualization

Data visualization is in its infancy in finance:

- Data sets are so large, and the velocity of data is so great these days, how are we to make sense of it all.
- Human vision linked to the human brain is still the best information processor and pattern recognition system we have!

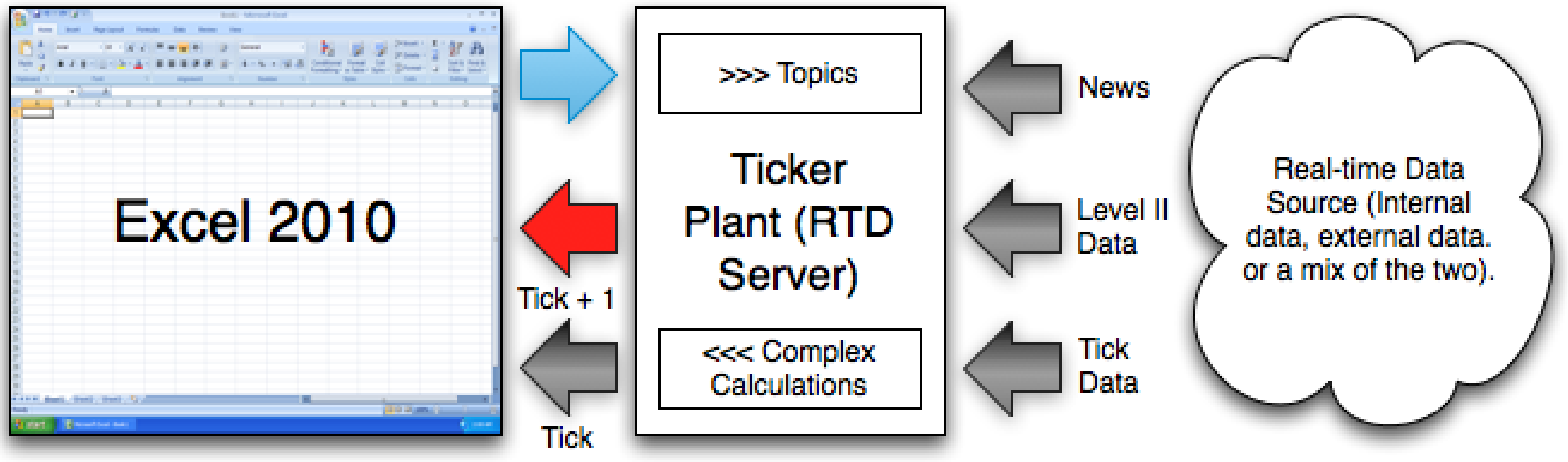
Applications - Added-Value Data

Lots of raw data of little value:

• Finance has lots of data.
• Problem is, how to turn it into actionable knowledge?
• There are tools and techniques, but all require vast amounts of computational power, and that's where GPUs can help.
• Example: Look-ahead ticker plant.

F ountainhead

Applications - Look-ahead Ticker Plant


Excel 2010
>>> Topics
Ticker Plant (RTD Server)
<<< Complex Calculations
Tick + 1
Tick
News
Level II Data
Tick Data
Real-time Data Source (Internal data, external data. or a mix of the two).

"Look-ahead" price ticks try to predict the next tick movement (seconds or minutes ahead)

F ountainhead

A Cautionary Warning

Apophenia

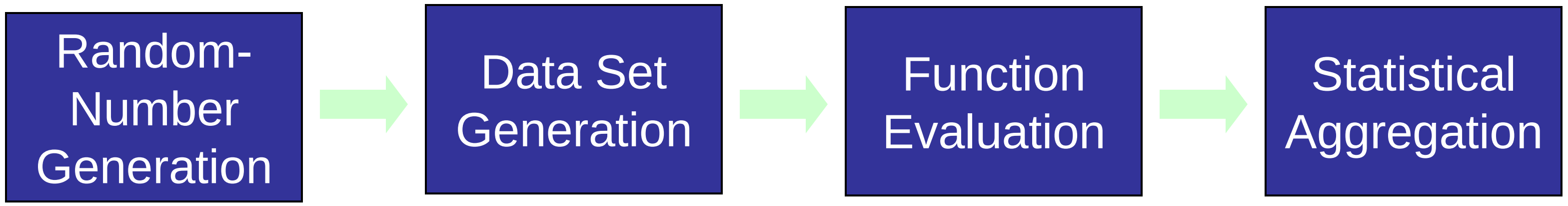
Apophenia is the experience of seeing meaningful patterns or connections in random or meaningless data. The term was coined in 1958 by Klaus Conrad, who defined it as the "unmotivated seeing of connections" accompanied by a "specific experience of an abnormal meaningfulness".

F ountainhead

~ B. Monte-Carlo on GPUs ~

F ountainhead

Elements



Typical Monte-Carlo simulation steps (simplified):

1. Generate random numbers.
2. Data set generation.
3. Function evaluation.
4. Aggregation.

F ountainhead

Guiding Principles for CUDA Monte-Carlo

General guiding principles:

- Understand the different types of GPU memory and use them well.
- Launch sufficient threads to fully utilize GPU cores and hide latency.
- Branching has a big performance impact; modify code or restructure problem to avoid branching.

Guiding Principles for CUDA Monte-Carlo (cont.)

- Find out where computation time is spent and focus on performance gains accordingly; from experience, oftentimes execution time is evenly split across the first three stages (before aggregation).
- Speed up function evaluation by being pragmatic about precision, using approximations and lookup tables, and by using GPU-optimized libraries.

F ountainhead

Guiding Principles for CUDA Monte-Carlo (cont.)

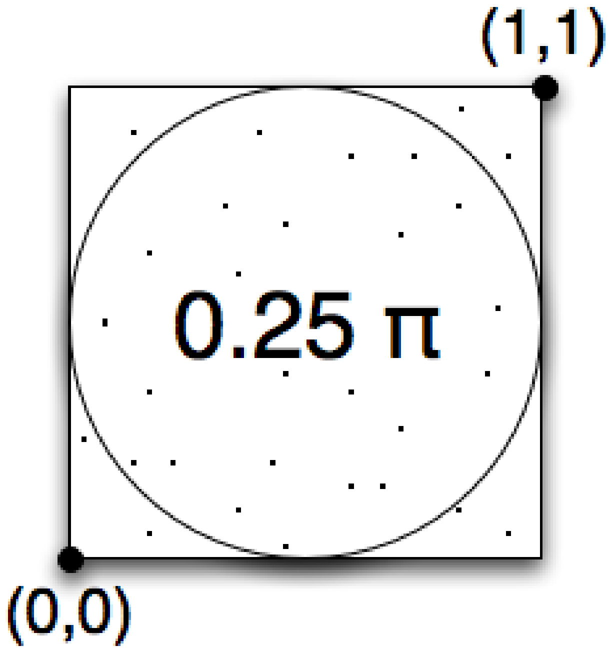
- Statistical aggregation should use parallel constructs (e.g., parallel sum-reduction, parallel sorts).
- Use GPU-efficient code: GPU Gems 3, Ch. 39; CUDA SDK reduction; MonteCarloCURAND; CUDA SDK radixSort.
- And, as always, parallelize pragmatically and wisely!

Example: Monte-Carlo using CUDA Thrust

Let’s consider a simple example of how Monte-Carlo can
Be mapped onto GPUs using CUDA Thrust.

CUDA Thrust is a C++ template library that is part of the CUDA toolkit and has containers, iterators and algorithms; and is particularly handy for doing Monte-Carlo on GPUs.

Example: Monte-Carlo using CUDA Thrust (cont.)



This is a very simple example that estimates the value of the constant **`PI`** while illustrating the key points when doing Monte-Carlo on GPUs.

(As an aside, it also demonstrates the power of CUDA Thrust.)

F ountainhead

Example: Monte-Carlo using CUDA Thrust (cont.)

```
int main() {
    size_t N = 10000000; // Number of Monte-Carlo simulations.
    // DEVICE: Generate random points within a unit square.
    thrust::device_vector<float2> d_random(N);
    thrust::generate(d_random.begin(), d_random.end(), random_point());
    // DEVICE: Flags to mark points as lying inside or outside the circle.
    thrust::device_vector<unsigned int> d_inside(N);
    // DEVICE: Function evaluation. Mark points as inside or outside.
    thrust::transform(d_random.begin(), d_random.end(),
                      d_inside.begin(), inside_circle());
    // DEVICE: Aggregation.
    size_t total = thrust::count(d_inside.begin(), d_inside.end(), 1);
    // HOST: Print estimate of PI.
    std::cout << "PI: " << 4.0*(float)total/(float)N << std::endl;
    return 0;
}
```

F ountainhead

Example: Monte-Carlo using CUDA Thrust (cont.)

```
struct random_point {
    __device__
    float2 operator()(int index) {
        default_random_engine rng;
        // Skip past numbers used in previous threads.
        rng.discard(2*index);
        return make_float2(
                    (float)rng() / default_random_engine::max,
                    (float)rng() / default_random_engine::max);
    }
};
```

Example: Monte-Carlo using CUDA Thrust (cont.)

```
struct inside_circle {
    __device__
    float operator()(float2 p) const {
    return (((p.x-0.5)*(p.x-0.5)+(p.y-0.5)*(p.y-0.5))<0.25) ? 1 : 0;
    }
};
```

F ountainhead

Example: Monte-Carlo using CUDA Thrust (cont.)

Let’s look at the code and how it relates to the steps (elements) of Monte-Carlo.

F ountainhead

Example: Monte-Carlo using CUDA Thrust (cont.)

```
// DEVICE: Generate random points within a unit square.
thrust::device_vector<float2> d_random(N);
thrust::generate(d_random.begin(), d_random.end(), random_point());
```

STEP 1: Random number generation. Key points:

- Random numbers are generated in parallel on the GPU.
- Data is stored on the GPU directly, so co-locating the data with the processing power in later steps.

F ountainhead

Example: Monte-Carlo using CUDA Thrust (cont.)

STEP 2: Generate simulation data. Key points:

- In this example, the random numbers are used directly and do not need to be transformed into something else.
- If higher level simulation data is needed, then the same principles apply: ideally, generate it on the GPU, store the data on the device, and operate on it in-situ.

Example: Monte-Carlo using CUDA Thrust (cont.)

```
// DEVICE: Flags to mark points as lying inside or outside the circle.
thrust::device_vector<unsigned int> d_inside(N);
// DEVICE: Function evaluation. Mark points as inside or outside.
thrust::transform(d_random.begin(), d_random.end(),
                  d_inside.begin(), inside_circle());
```

STEP 3: Function evaluation. Key points:

- Function evaluation is done on the GPU in parallel.
- Work can be done on the simulation data in-situ because it was generated & stored on the GPU directly.

Example: Monte-Carlo using CUDA Thrust (cont.)

```
     // DEVICE: Aggregation.
     size_t total = thrust::count(d_inside.begin(), d_inside.end(), 1);
     // HOST: Print estimate of PI.
     std::cout << "PI: " << 4.0*(float)total/(float)N << std::endl;
```

STEP 4: Aggregation. Key points:

- Aggregation is done on the GPU using parallel constructs and highly GPU-optimized algorithms (courtesy of Thrust).
- Data has been kept on the device throughout and only the final result is transferred back to the host.

F ountainhead

Example: Monte-Carlo using CUDA Thrust (cont.)

Key takeaways from this example:

- Use the tools! CUDA Thrust is a very powerful abstraction tool for doing Monte-Carlo on GPUs.
- It’s efficient too, as it generates GPU optimized code.
- Do as much work on the data as possible in-situ, and in parallel. Only bring back to the host the minimum you need to get an answer.

Fountainhead

Parting Thought

Axiom: Developing software for finance and making money using that software isn’t one of those activities – like ice skating – where you get extra points for doing things the hard way. Quite the opposite, in fact.

(Lemma: The financial crisis was a triple-axel back-flip with tuck. You can’t make AAA from CCC ingredients.)

F ountainhead

Questions & Answers

ajtsheppard@gmail.com