QuantAlea

GTC 2012

May 14-17

San Jose, California

Dr. Daniel Egloff
daniel.egloff@quantalea.net
+41 44 520 01 17
+41 79 430 03 61

New GPU Pricing Library

QuantAlea

- Client project for Bank Sarasin
- Highly regarded sustainable Swiss private bank
- Founded 1841
- Core business
  - Asset management
  - Investment advisory
  - Investment funds
  - Structured products
  - Private and institutional clients
- End of 2011, Safra group acquired majority interest in Bank Sarasin
  - Supports Bank Sarasin's future-oriented positioning as an independent leader in private banking

QuantAlea

QuantAlea

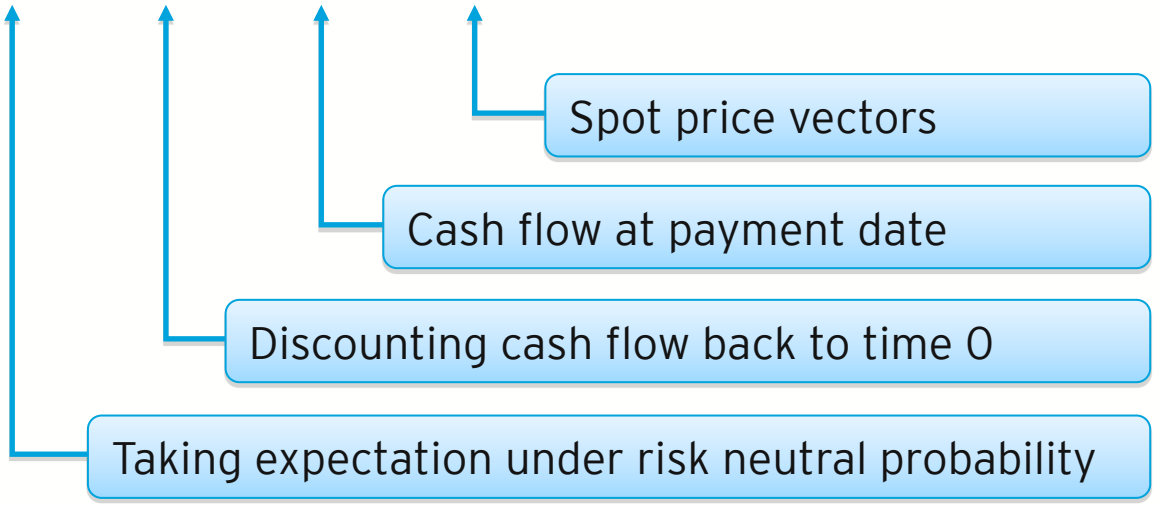
- Consulting and software development for quantitative finance
- Based in Zurich
- Unique blend of experience
  - Financial business side
  - Quant and financial modeling aspects
  - Numerical computing
  - Software engineering
- Early adopters, starting in 2007 to use GPU in finance
- Proven GPU track record
  - Successfully completed various projects in quantitative finance

Derivative Pricing

QuantAlea

- Arbitrage free price of a derivative is an expectation value

$$V = E_Q\left[\sum_i P(0,t_i) f_{t_i}(S_{t_0},\ldots,S_{t_i})\right]$$



- Conceptually simple but .....

Challenges

QuantAlea

Complex products and cash flow structures like baskets and hybrids

Intensive and difficult numerical calculations

Various algorithms such as Monte Carlo, PDE, Fourier methods, ...

Fast changing requirements

Different asset classes difficult to unify

Imperfect and missing market data

Awkward market conventions

Derivative pricing codes are complex work flows

Large development and coding effort for model development and testing

Adding GPU acceleration further complicates the problem

Solution Approch

QuantAlea

Derivative pricing codes are complex work flows

Large development and coding effort for testing and model development

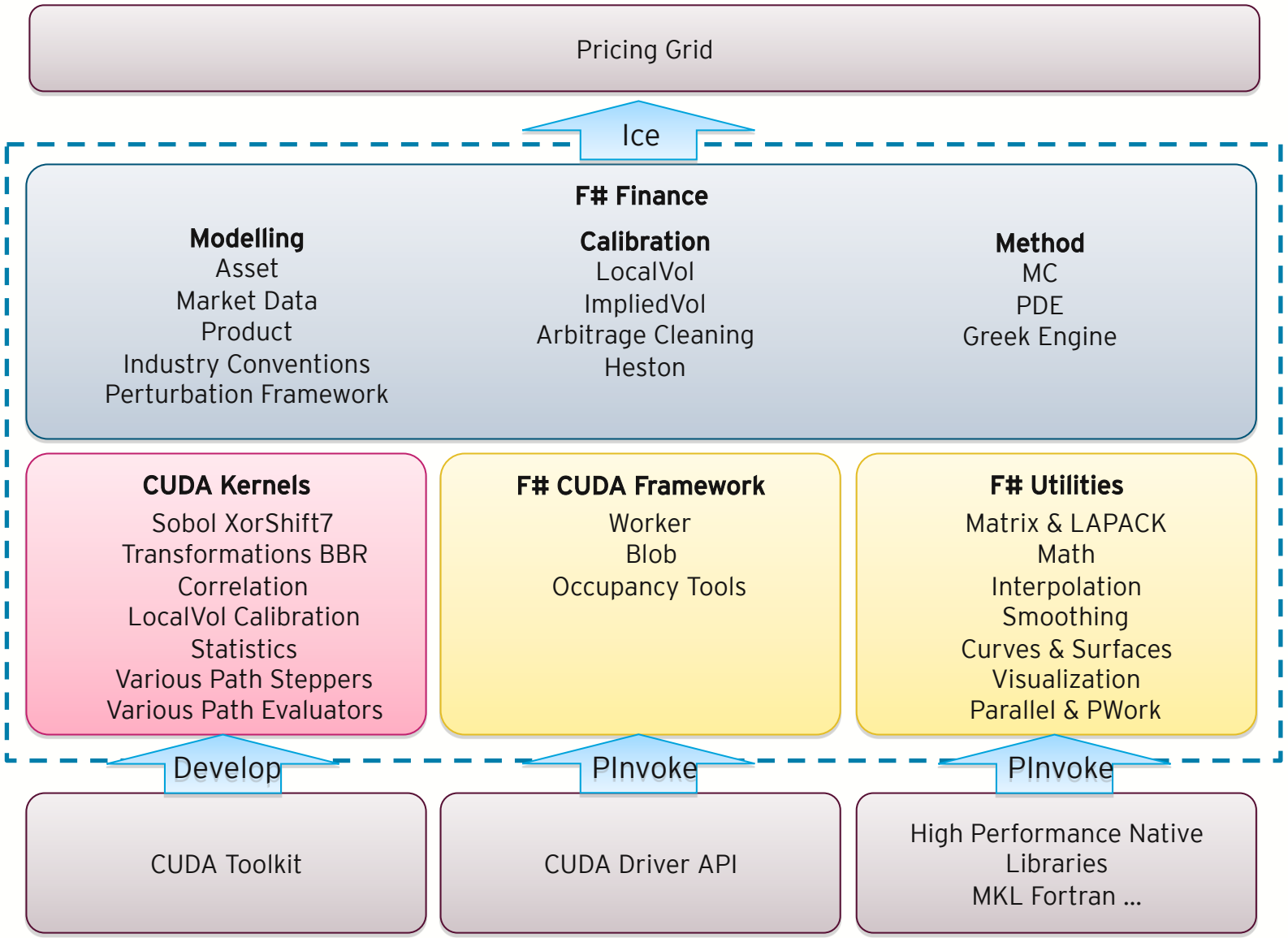
- Use a functional language like F#, Scala, …
  - Functions first class members of language
  - Better suited for numerical problems
  - Immutable data structures
- Use a VM like Microsoft .NET CLR or JVM
  - Garbage collection
  - JIT technology
  - Hotspot compilation
- Introduce proper domain specific abstractions

Adding GPU acceleration further complicates the problem

- Use GPU programming framework
- Check against CPU reference implementation

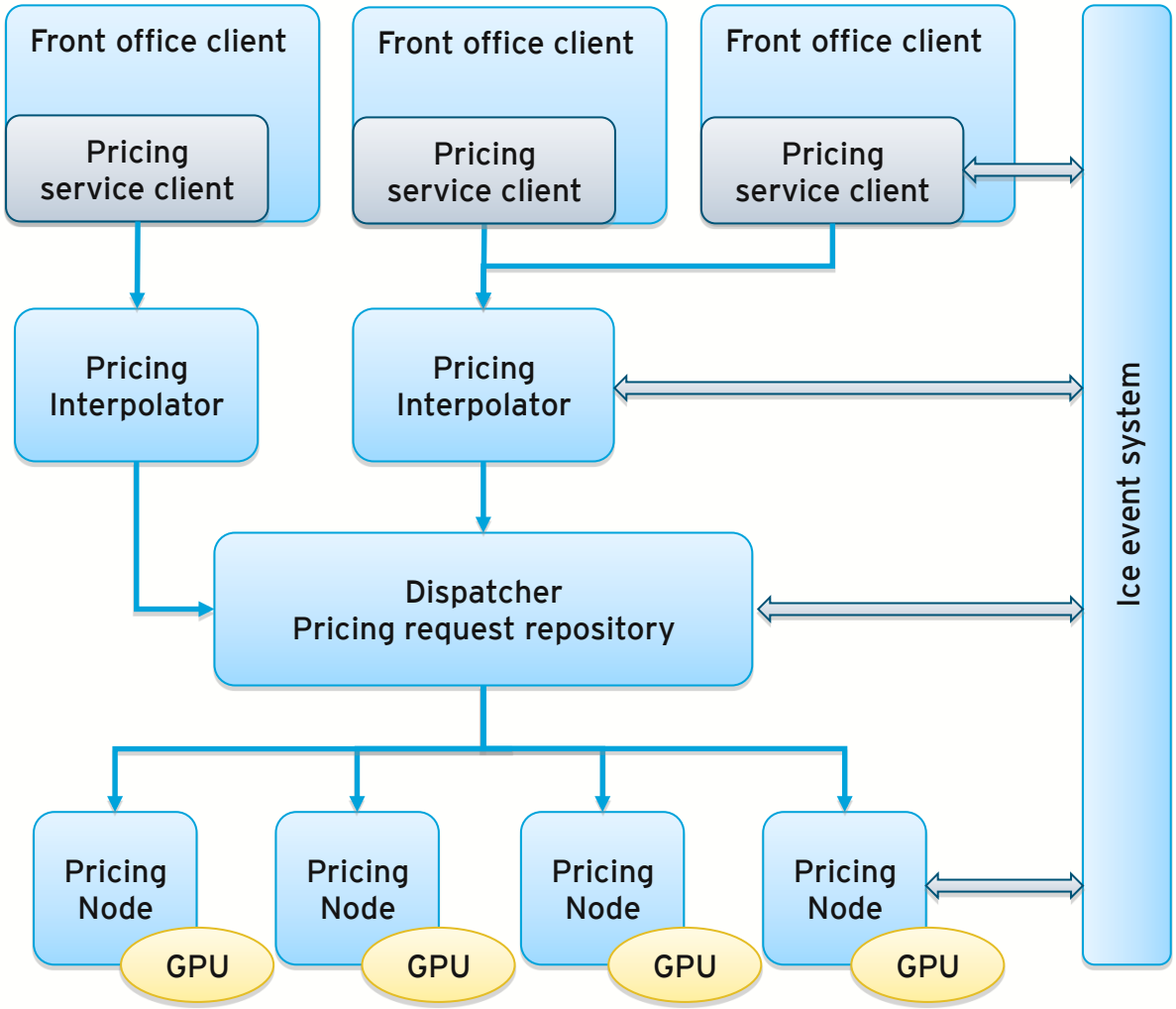
Library Architecture

QuantAlea



Grid Architecture

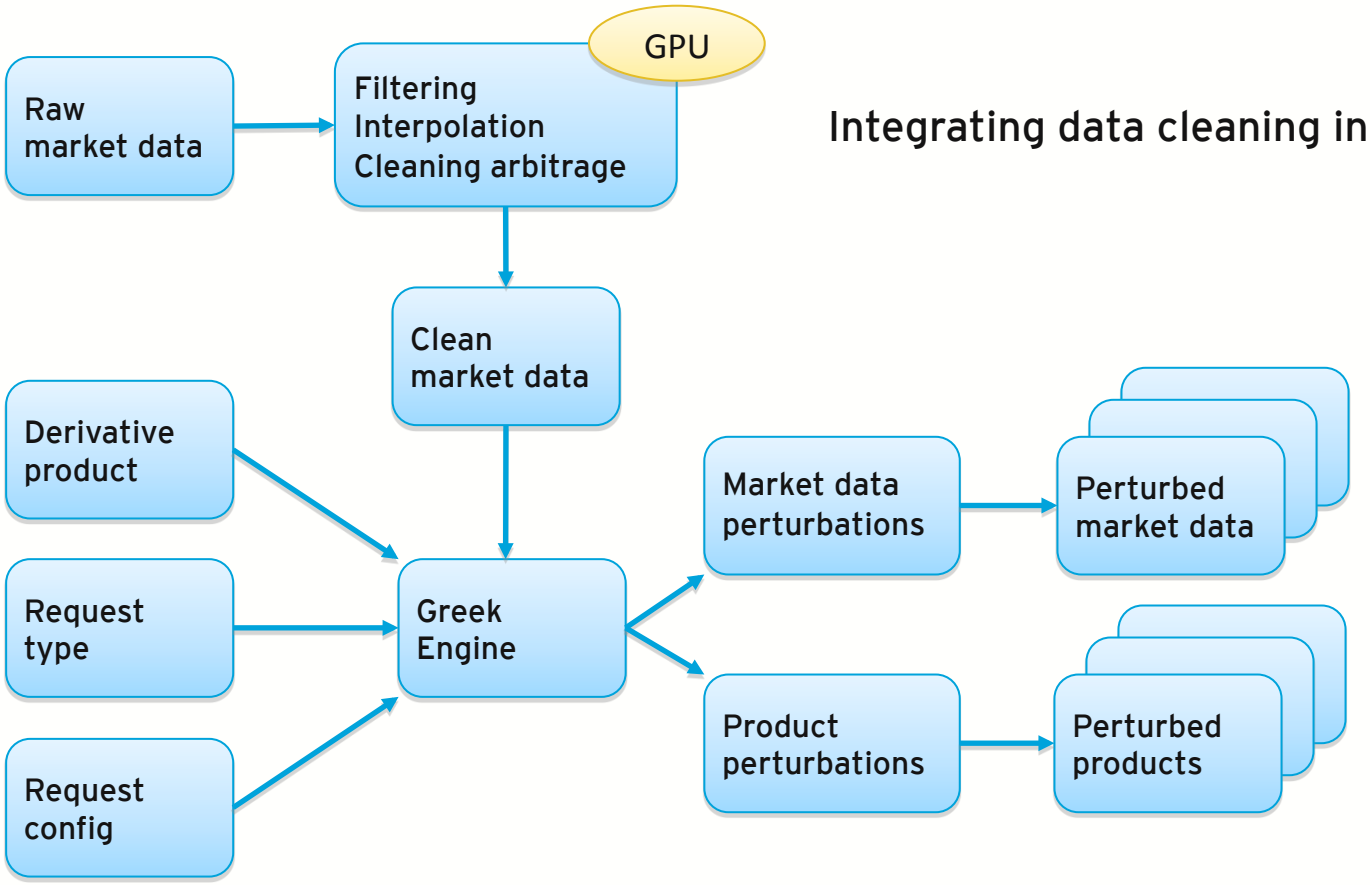
QuantAlea



- Pricing service client sets up pricing request and data transfer via remote objects
- Pricing interpolators give real-time best estimated prices
- Price calculations scheduled on GPUs
- Event system updates client with new pricing results
- Request repository for fault tolerance
- Add compute resources dynamically

Pricing Work Flow I

QuantAlea

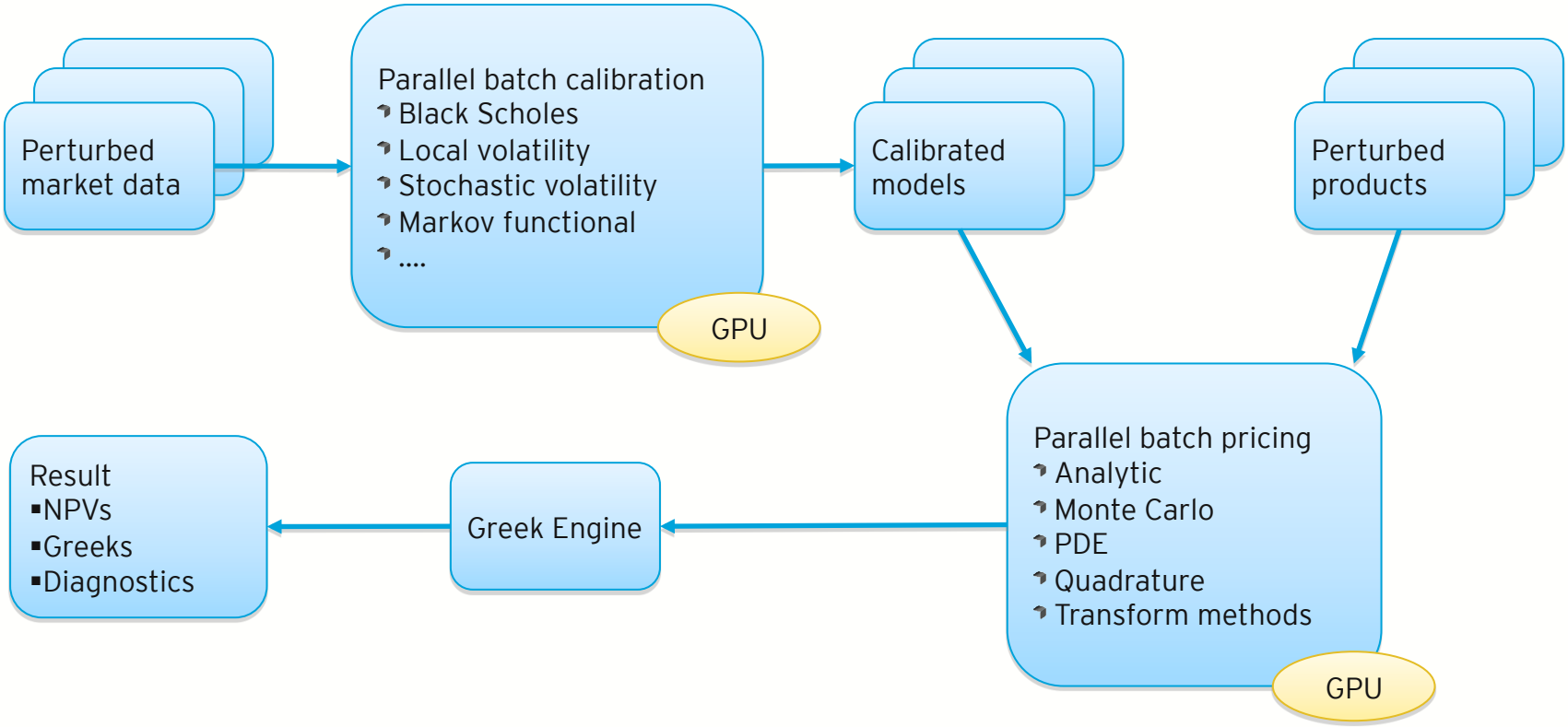


Integrating data cleaning in library

Perturbation pattern across market data and products improves unification

Pricing Work Flow II

QuantAlea



Batching calibration and pricing
Greek engine aggregates calculated NPVs to sensitivities

F# Cuda Framework

QuantAlea

- Abstracts CUDA device and context
- Provides CPU thread
- Bind worker to F# async workflow

Usability in F#

Worker

- Manage variables by name, scalar, 1D, 2D
- Strongly typed
- Automatic texture binding

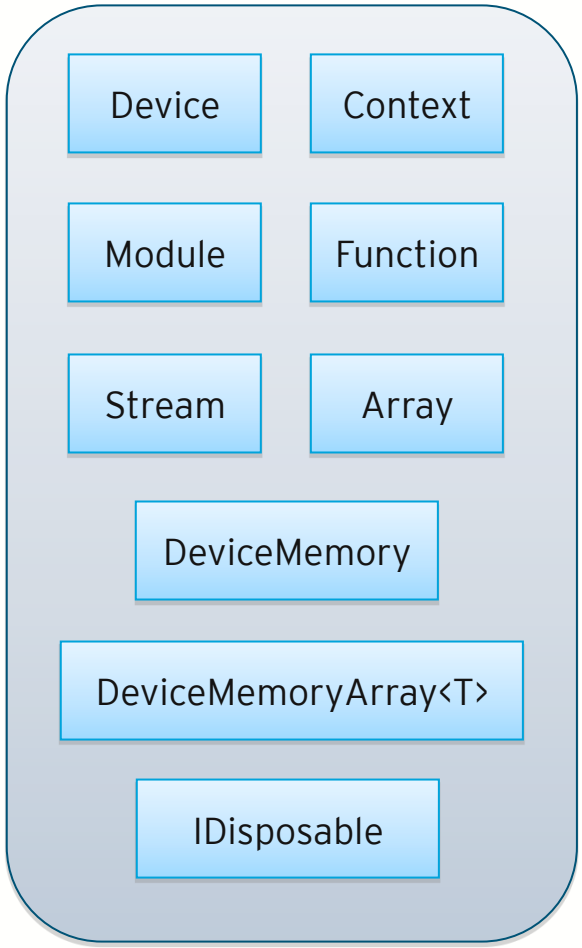
Blob System

- Manage complex data structures
- One host to device copy call
- One device allocation call
- Dispose at once

Performance

Occupancy Tool

- Calculate best thread number to get high occupancy
- Use multiple streams to launch kernels in parallel



F# Cuda Framework I

QuantAlea

1) Write kernel wrapper

Step1: load the ptx file

Step2: calculate kernel launch shape

```
type SobolRng(worker:Worker, nDimensions:int, nVectors:int) =

    let tokenName (name:string) = sprintf "sobol.%s" name
    let ptxModuleName = "Alea.Cuda.Kernel.Math.Random.Sobol.sm_13.ptx"
    let ptxModule = Util.loadModule worker (Assembly.GetExecutingAssembly()) ptxModuleName
    let kernelName = "sobolKernelGenerateFloat64"
    let kernel = ptxModule.[kernelName]
    let nBestThreads = Util.calcBestThreadNumberEx worker kernel (fun x -> 0) [64; 128; 256; 512; 1024]
    let kernel = kernel.SetGridShape(1u, uint32(nDimensions), 1u)
    let kernel = kernel.SetBlockShape(uint32(nBestThreads), 1u, 1u)
    let directions = let rng = SobolGeneratorJumpAhead(nDimensions, None) in rng.directionNumbers

    member this.dynamicTokens =
        seq {
            yield BlobHelper.ArrayParam (tokenName "directions") directions
        }

    member this.update (blob:BlobParams) (dmem:DeviceMemoryArray<float>) (offset:int) (stream:CUstream) =
        seq {
            yield lazy (
                use kpl = new KernelParameterList(5)
                kpl.[1] <- KernelParameterUtil.CreateParameter(nVectors)
                kpl.[2] <- KernelParameterUtil.CreateParameter(nDimensions)
                kpl.[3] <- KernelParameterUtil.CreateParameter(offset)
                kpl.[4] <- KernelParameterUtil.CreateParameter(blob.getDynamicParam<uint32>(tokenName "directions"))
                kpl.[5] <- KernelParameterUtil.CreateParameter(dmem)
                kernel.SetStream(stream).Launch(kpl))
        }
```

Step3: generate blob tokens for data the kernel will use

Step4: generate lazy expression for launching kernel in the CUDA context and streams of the worker

F# Cuda Framework II

QuantAlea

2) Use CUDA kernel wrappers in F# async workflow

```
let workflow (worker:Worker) (nDimensions:int) (nVectors:int) (nInst:int) = async {
    do! Async.SwitchToContext worker.SynchronizationContext

    let n = nDimensions * nVectors
    let rng = SobolRng(worker, nDimensions, nVectors)
    let reducer = Reduce<float>(nInst, worker, n, ReduceMeanAndM2())

    let blobTokens = seq {
        yield! rng.dynamicTokens
        yield! reducer.dynamicTokens
        for i = 0 to nInst - 1 do yield BlobHelper.SizeExtent1D "numbers" i (n * sizeof<float>)
        }

    let blob = blobTokens |> createBlob worker

    let streams = Array.init nInst (fun i -> worker.GetStream(i+1))

    let commands = streams |> Array.mapi (fun i stream ->
        seq {
            let dInput = blob.getDynamicExtent1D<float> "numbers" i
            let offset = i * 100
            yield! rng.update blob dInput offset stream
            yield! reducer.reduce blob dInput i stream
        })

    commands |> launchPipelines

    let result = reducer.result blob
    blob.Dispose()
    return result }
```

Switch to thread context of the worker

Create instance of kernel wrappers

Collect blob tokens from each kernel wrappers and create blob on device

Collect lazy kernel launch expression from each wrappers, and launch them

Gather results from one of the kernel wrappers

3) Launch workflow with some devices

```
let workers = [| 0 .. Api.DeviceCount - 1 |] |> Array.choose (fun deviceId ->
    match Api.GetDeviceComputeCapability(CUdevice(deviceId)) with
    | major, minor when major >= 2 || major = 1 && minor >= 3 -> Some (new Worker(deviceId))
    | _ -> None )

printfn "%A" workers
```

Create workers with devices that support double precision

```
let test (nDimensions:int) (nVectors:int) (nInst:int) =
    workers
    |> Array.map (fun worker -> workflow worker nDimensions nVectors nInst)
    |> Async.Parallel
    |> Async.RunSynchronously

let results = test 8 10000000 2

results |> Array.iteri (fun i result ->...
```

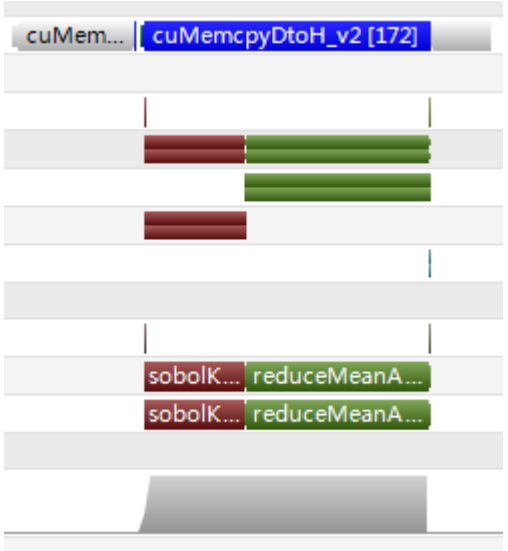
Run workflows asynchronously in parallel and collect results

```
workers |> Array.iter (fun w -> w.Dispose())
```

Release worker resources

Result and Conclusions

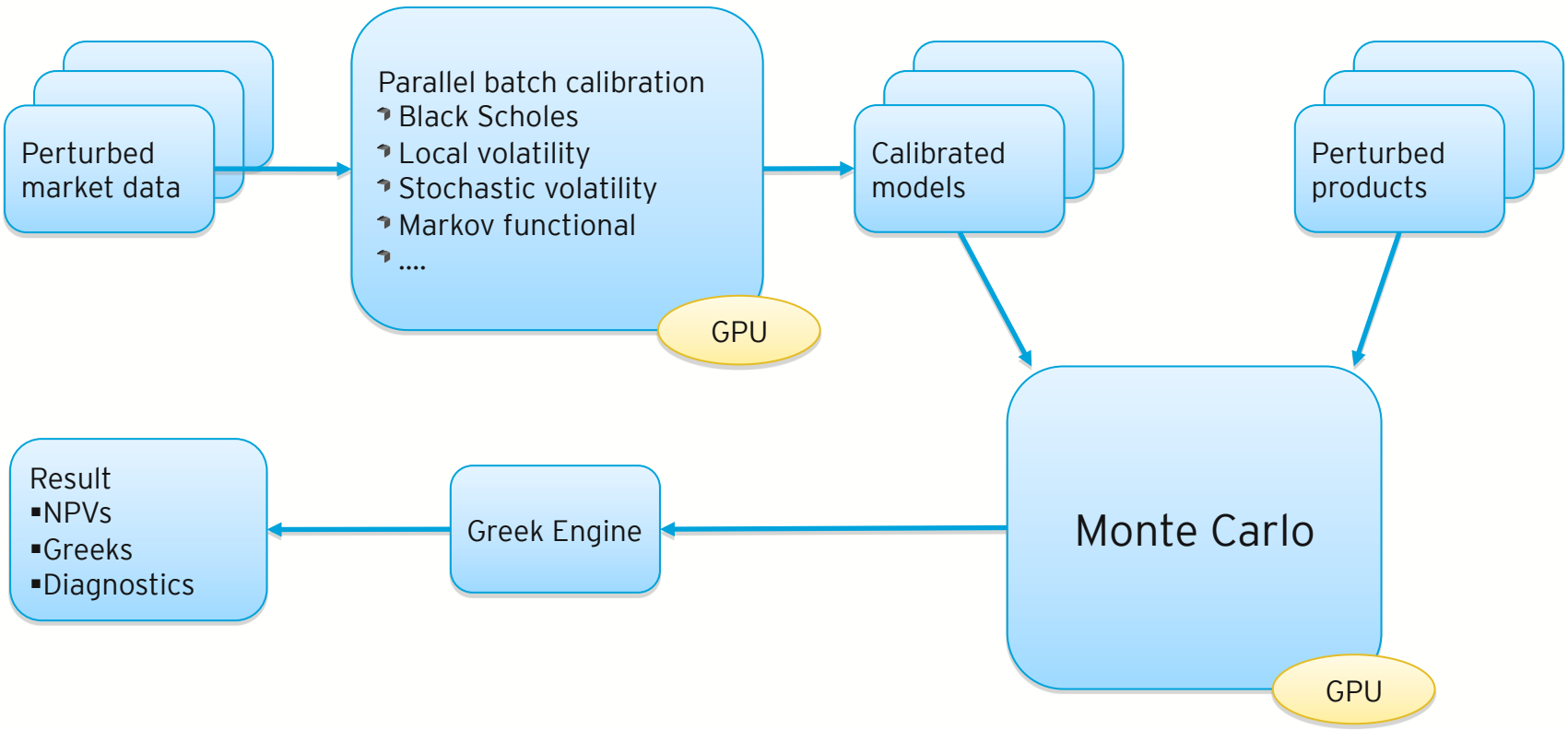
- Create kernel wrapper in F#, hide complex kernel launch logic, such as the reduce algorithm
- Use occupancy tool to calculate a best thread number to make GPU busy
- Use stream tool to make kernel running concurrently
- Use F# async workflow to combine worker, blob, and multiple kernel wrappers
- Blob handles complex data structure and texture binding to minimize host to device copy and multiple memory allocation on device



Kernel	Shared	Grid	Block	Occupancy	Time
sobolKernelGenerateFloat64	0	1x8x1	256x1x1	83%	10.588
sobolKernelGenerateFloat64	0	1x8x1	256x1x1	83%	10.587
reduceMeanAndM2_1_512_Float64	16384	8x1x1	512x1x1	67%	19.507
reduceMeanAndM2_1_512_Float64	16384	8x1x1	512x1x1	67%	19.507
reduceMeanAndM2_2_004_Float64	2048	1x1x1	4x1x1	17%	0.008
reduceMeanAndM2_2_004_Float64	2048	1x1x1	4x1x1	17%	0.007

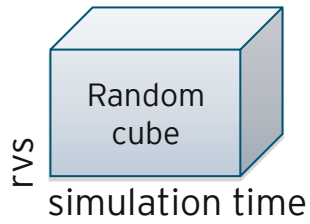
Monte Carlo Method

QuantAlea



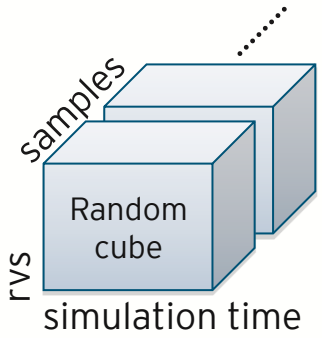
MC Pricing Path Steppers

QuantAlea



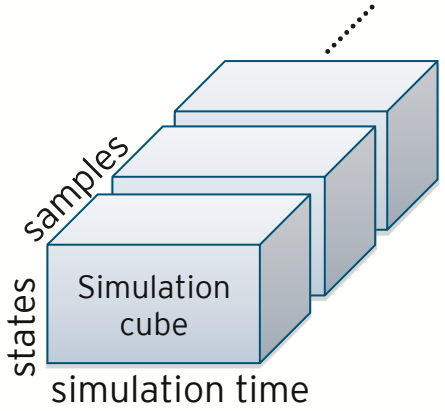
Independent random numbers

- Xorshift7 and Sobol
- Brownian bridge reordering
- Different distributions



Correlated random numbers

- One cube per correlation perturbation



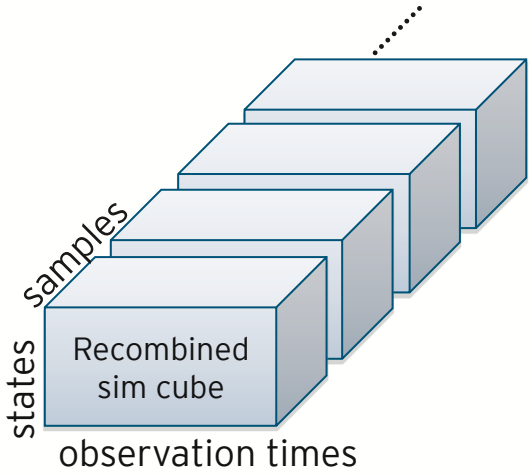
Simulated paths

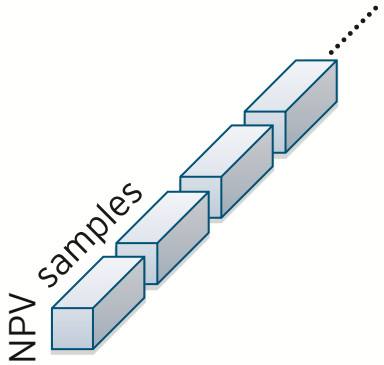
- One cube per aggregated perturbation or basis perturbation
- Additional states for barrier bias reduction

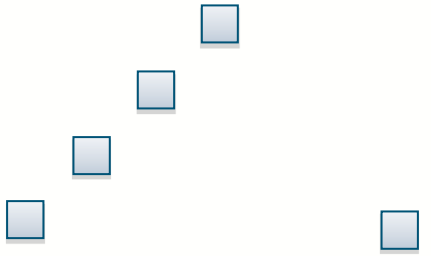
Multiple workers (one per core or GPU) perform multiple iterations
until desired convergence accuracy or number of samples exhausted

MC Pricing Path Evaluators

QuantAlea







Recombined simulation cubes

- Path reuse optimization based on sparsity and graph coloring
- One cube per required perturbation

NPV samples

- Result of path evaluators and payoff generation
- All cash flows converted to payment currency and discounted
- One cube per required perturbation

NPV block statistics

- Block-wise parallel reduction for mean and moments
- Gather from multiple devices to host
- Sequentially aggregated on host
- Update stopping criteria

Multiple workers (one per core or GPU) perform multiple iterations
until desired convergence accuracy or number of samples exhausted

MC Path Reuse

QuantAlea

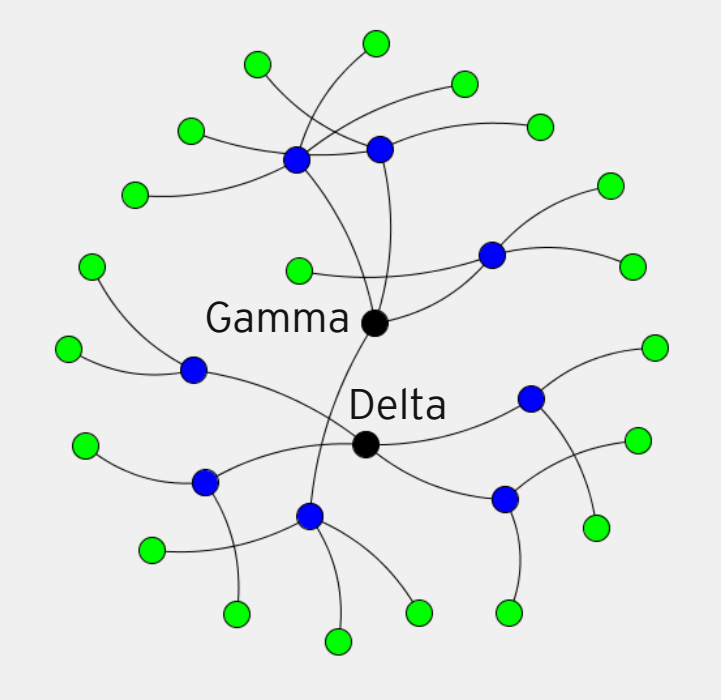
Algorithmic optimization to minimize path simulation effort for basket options

- Compute dependency structure of stochastic differential equation on parameters
- Solve a graph coloring problem to find structurally orthogonal decomposition of dependency structure
- Structurally orthogonal components are independent perturbations which can be grouped to aggregated perturbations
- Find recombination logic to express every perturbation as a linear combination of aggregated perturbations
- Not obvious in the context of so called multi-asset quanto options
- Difficult to implement on GPU because it leads to non-coalescing memory access patterns

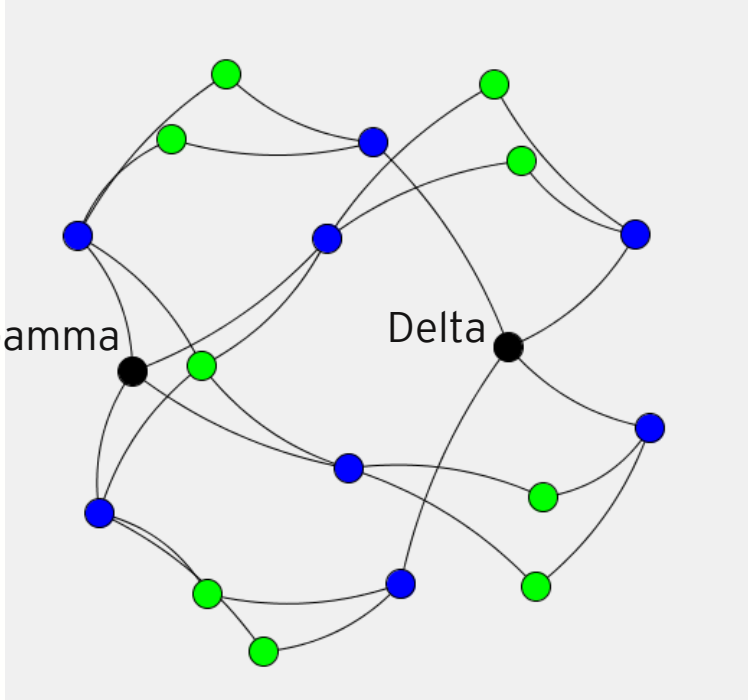
Example - Basket of 4

QuantAlea

Naive



Sharing NPVs

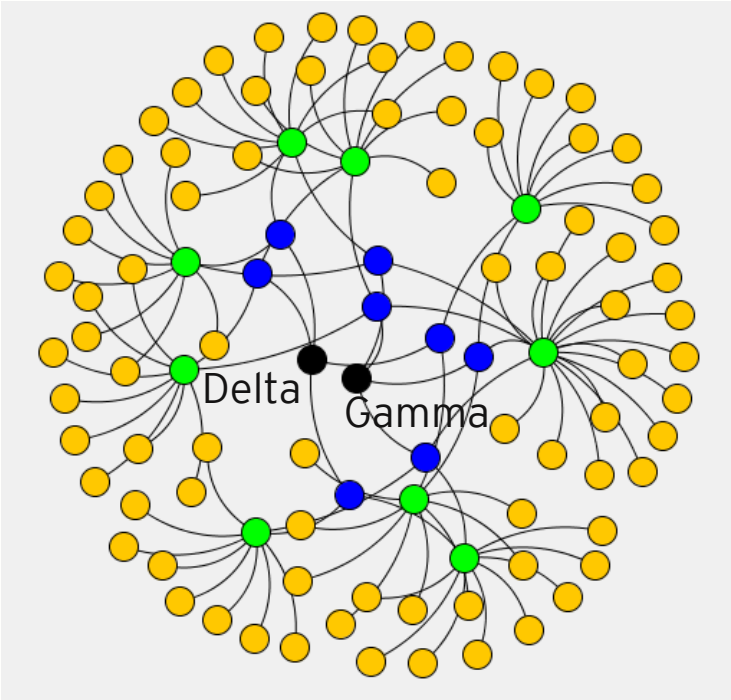


Black	Sensitivity vectors Delta, Gamma
Blue	Sensitivity coordinates for Delta, Gamma
Green	Perturbations

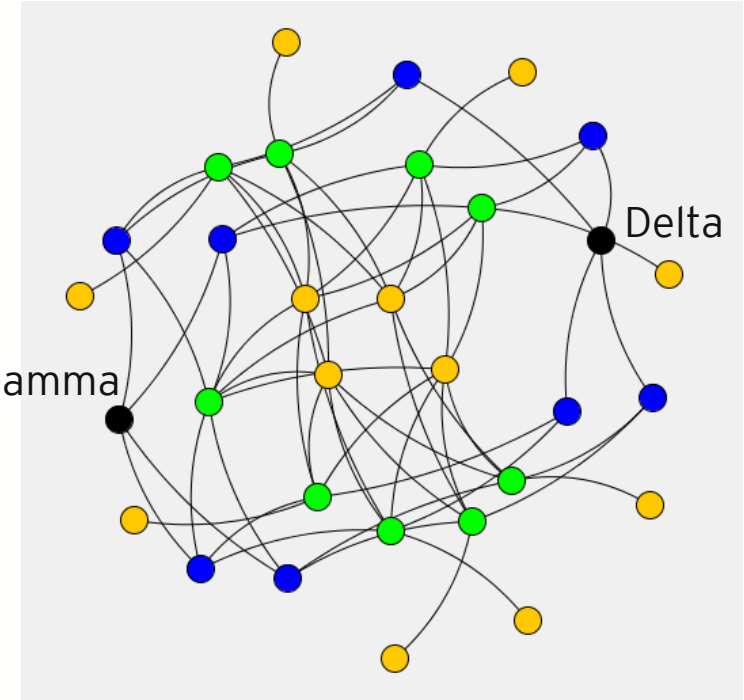
Example - Basket of 4

QuantAlea

Standard



Path reuse optimization



Yellow Simulated states

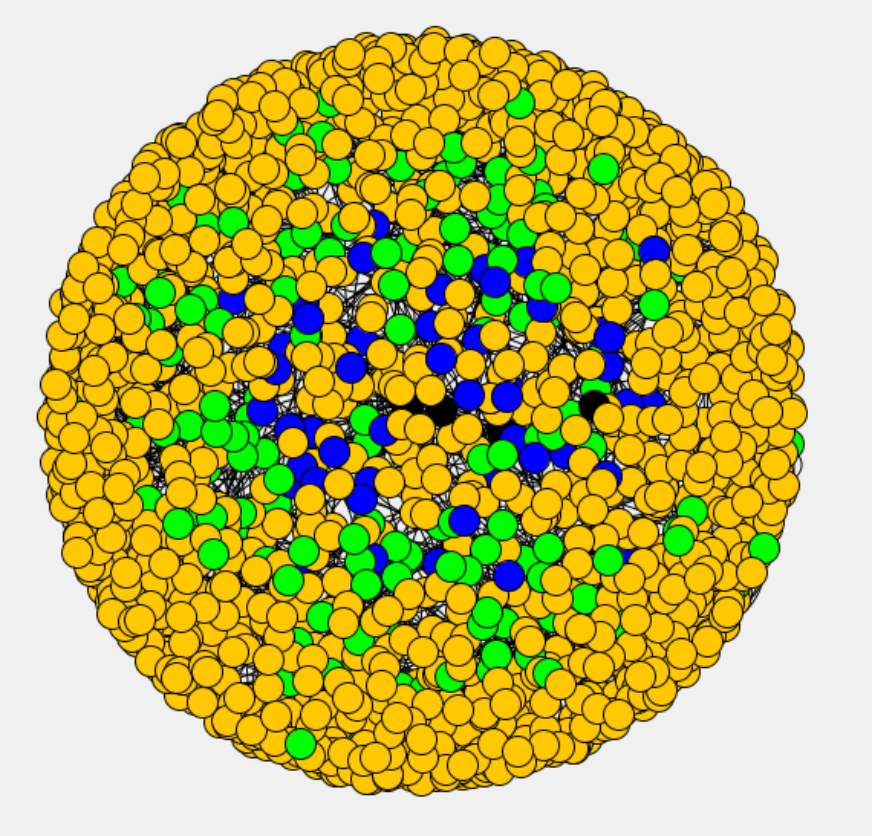
The simulation cost is proportional to the number of yellow nodes

Path reuse reduced cost by a factor of 5!

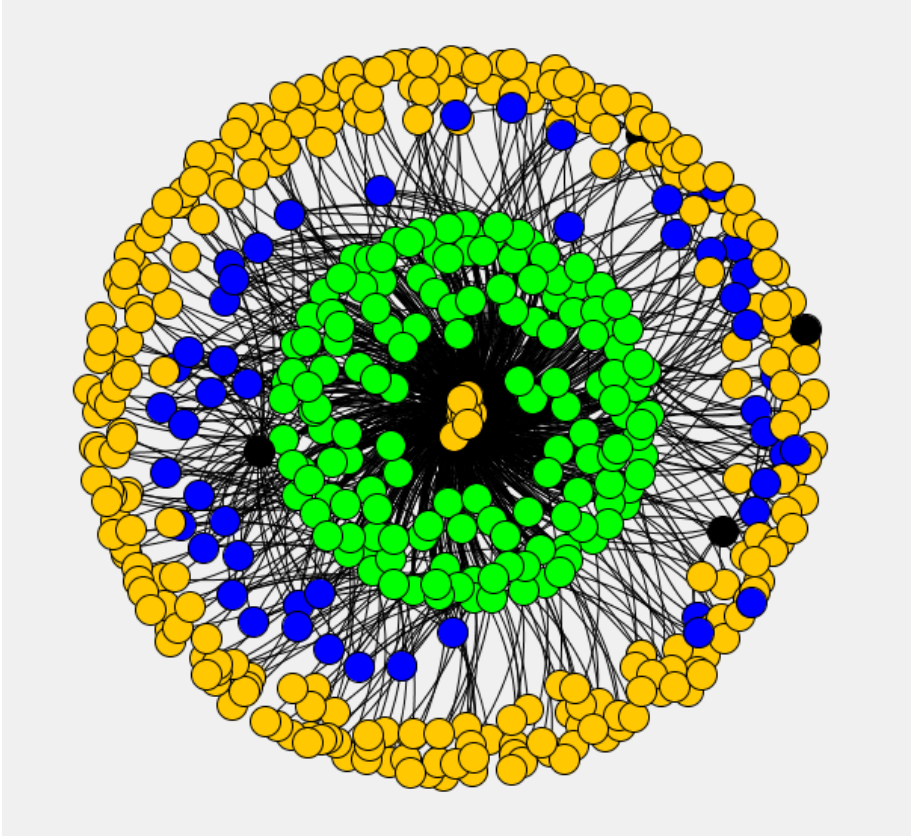
Example - Basket of 8

QuantAlea

Standard



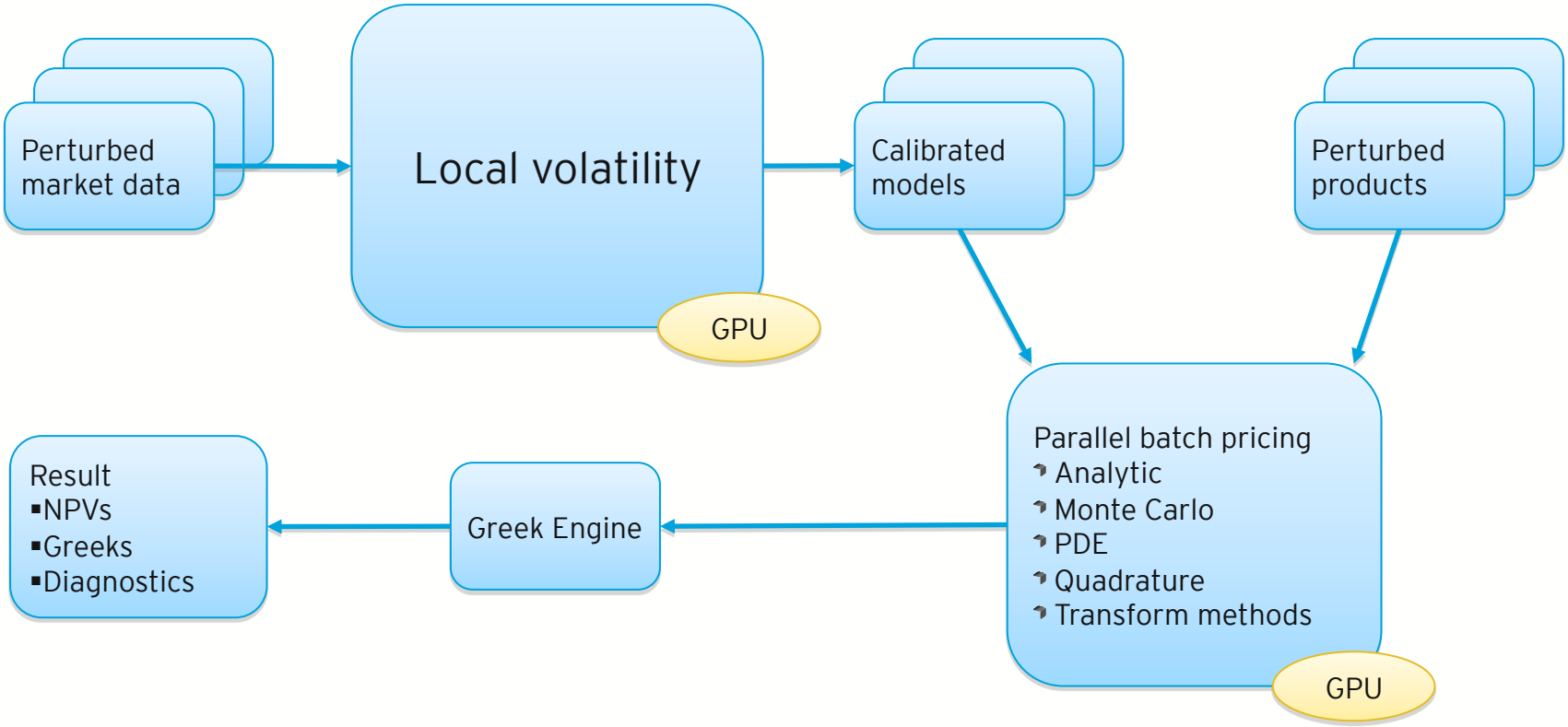
Path reuse optimization



Even more extreme for Delta, Gamma, Cross Gamma and Vega of a basket of 8 assets

Local Volatility Calibration

QuantAlea

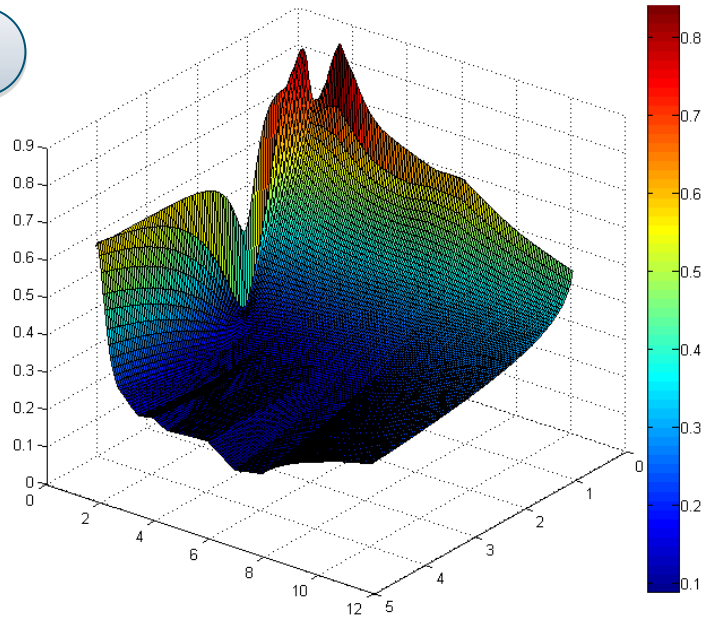


Local Volatility GPU

QuantAlea

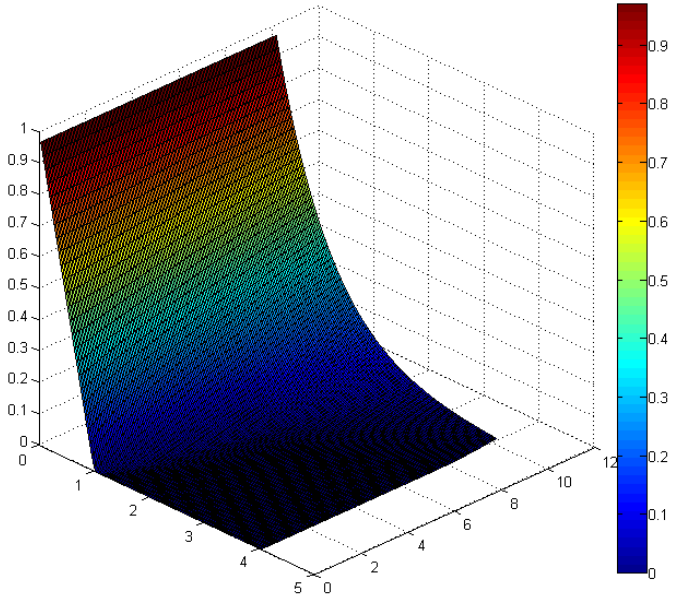
- Local volatility calibration is numerically challenging
  - Standard approach via Dupire's formula may produce instable results
  - PDE based techniques are more stable but more difficult to implement
  - Incorporation of discrete dividends is conceptually and numerically difficult
- PDE based implementation using several kernels

1



Initial implied volatilities from market quotes

2



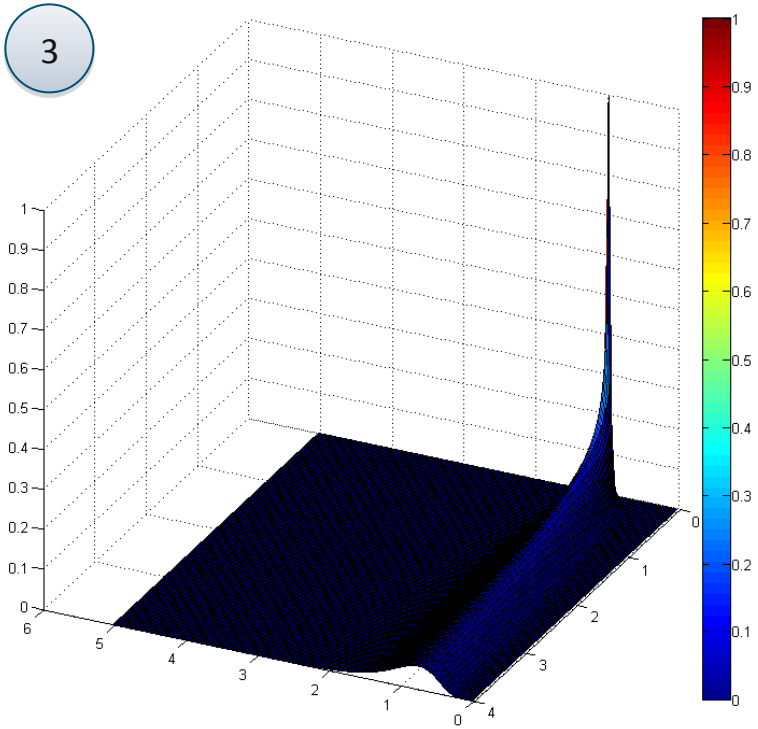
Call price surface

- Properly transformed to strip off dividend singularities
- Independent local calculations

Local Volatility GPU

QuantAlea

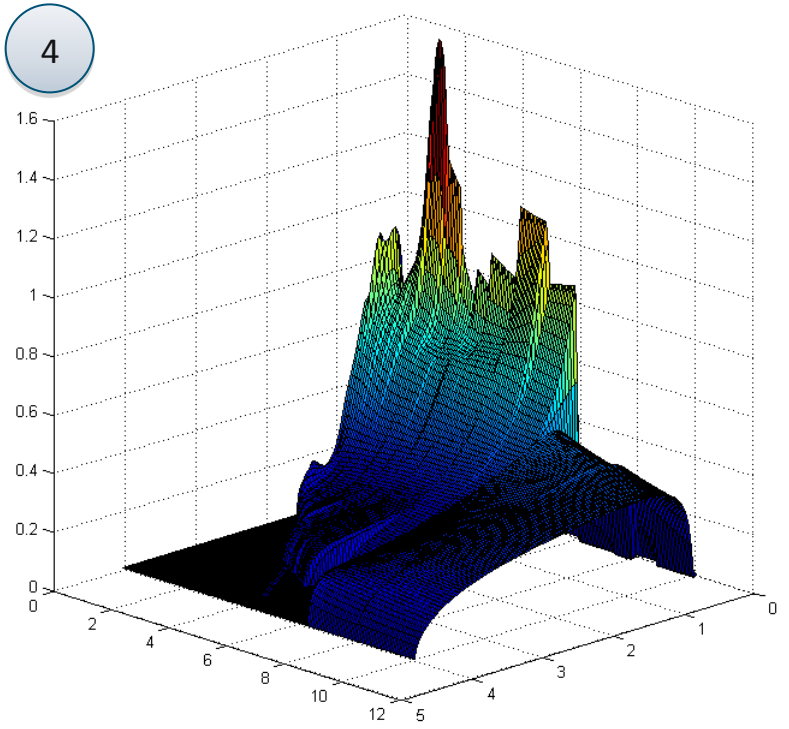
3



Arrow Debreu price density

- Independent local calculations

4



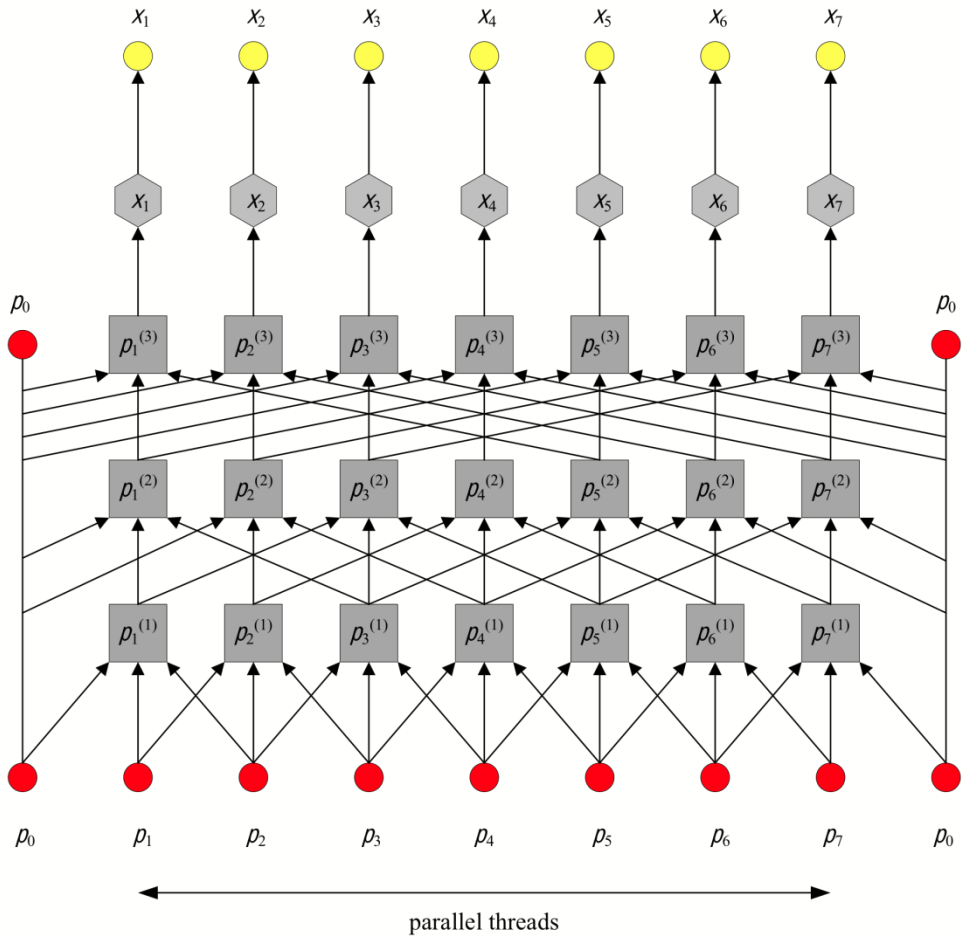
Final local volatilities with dividend singularities

- Calculated inside empirical truncation bounds
- Solving a tri-diagonal system for every time slice
- Transformation to account for discrete dividend singularities

Tri-Diagonal Solver

QuantAlea

- Local volatility calibration and PDE pricing builds on optimized parallel tri-diagonal solver based on parallel cyclic reduction (PCR)



Local Volatility GPU

QuantAlea

```
let cudaWrapper =
    PureVolsToPureCallPrices.create worker pureTimes pureStrikes pureVolsToPureCallPricesParams
    |> PureCallPricesToArrowDebreuPrices.create arrowDebreuConfig beta
    |> EmpiricalTruncationBound.create
    |> AbrLocalVolatilityPure.create abrConfig
    |> ResampleLocalVolForLogSpot.create outputDouble timeGrid.times spotGrid resampleLocalVolParams

{ new LocalVolCalibrationCuda with
    member this.dynamicTokens = cudaWrapper.dynamicTokens
    member this.dDiffusion blob = cudaWrapper.dLocalVols(blob)
    member this.dDrift blob = cudaWrapper.dLocalDrifts(blob)
    member this.create blob = cudaWrapper.create(blob)
}
```

Use 5 kernel wrappers to create a local volatility calibration pipeline

Last kernel of pipeline provides the drift and diffusion coefficient matrices for local volatility model simulation

```
do discretizedDiffusionModels |> Array.Parallel.iter (fun m -> m.driftAndDiffusion.Force() |> ignore)

member this.forGpu (simulationCoord:DiscretizationCoord) (worker:Worker) (timeGrid:TimeGrid) (spotGrid:float[][]) (outputDouble:bool) =
    { new LocalVolCalibrationCuda with
        member this.dynamicTokens = ...

        member this.dDiffusion blob =...

        member this.dDrift blob =...

        member this.create blob = ()
    }
```

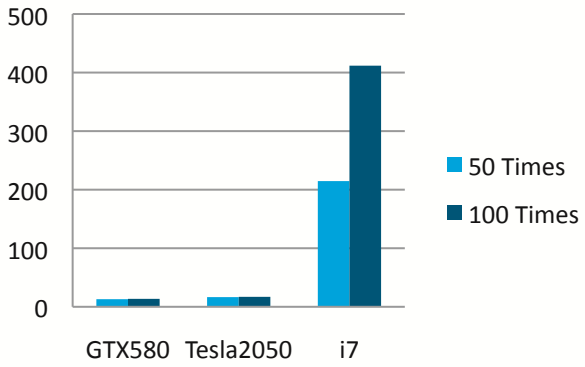
Can be calculated in parallel on multiple CPU cores or on GPU

- Chain 5 kernel wrappers to a complete calibration pipeline
- Final kernel adapts for the desired path stepper either in log spot or pure price coordinates
- Parallel calibration for all combination of basis model and assets as a single batch
- Fallback to CPU if no device with double precision support, use F# lazy evaluation and parallel arrays to implement parallel calibration on multi-core CPU

Local Volatility GPU

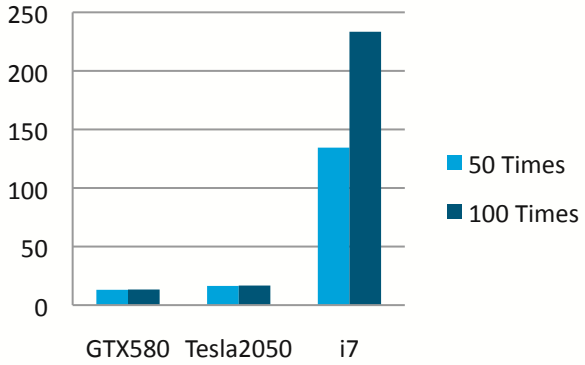
QuantAlea

Local vol calibration
20 surfaces in log spot



Device	Time steps	Log spot	Pure spot
GTX580	50	12.98	13.08
Tesla2050	50	16.53	16.39
i7	50	214.66	134.47
GTX580	100	13.58	13.33
Tesla2050	100	17.01	16.72
i7	100	411.77	233.41

Local vol calibration
20 surfaces in pure spot



- Local volatility calibration up 30 times faster on GPU
- Pure spot version only requires diffusion
- Almost no additional runtime cost for
  - log spot, which requires diffusion and drift
  - more time steps on GPU

MC Timings

QuantAlea

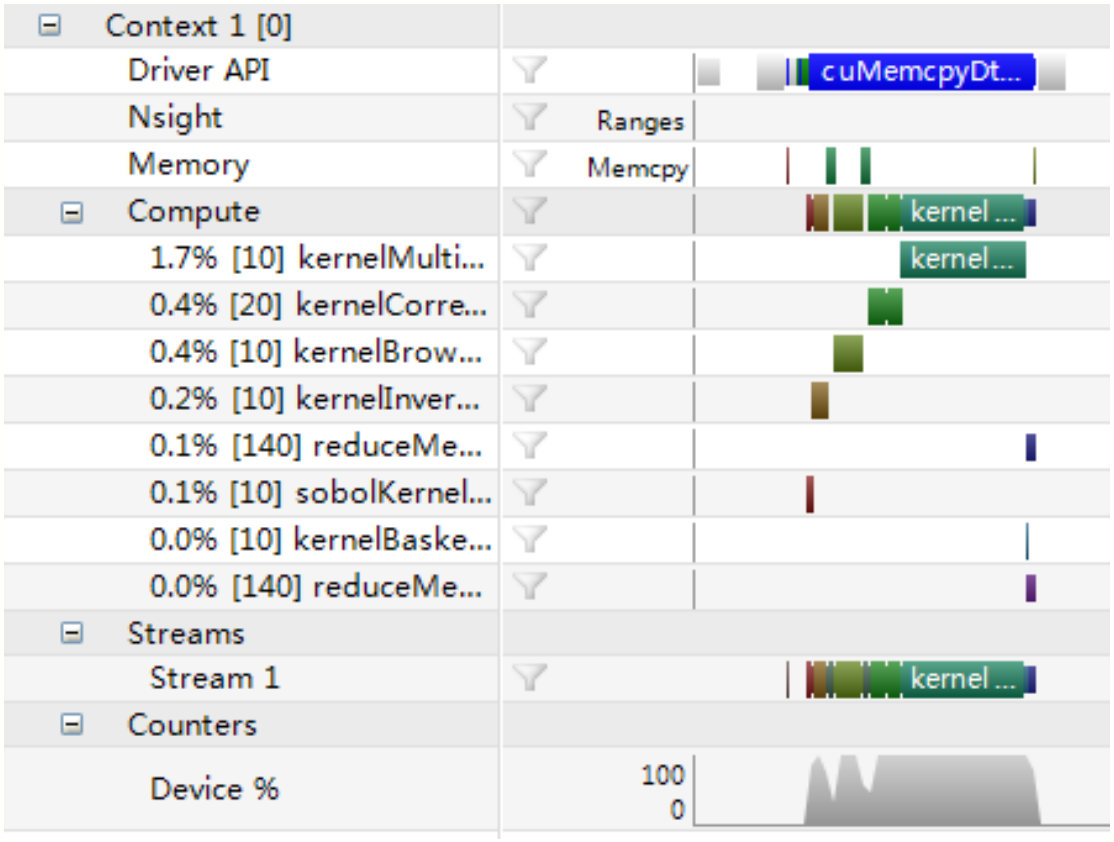
Standard Basket of 4 assets

- Black Scholes Log Spot
- Calculating price and Delta, Gamma, Vega, Correlation Delta
- Results in 5 basis models and a total of 14 market perturbations

Samples	times	devices	n	acc samples	T total (ms)	T scaled	T gpu	T gpu scaled	T prepare	cpu/gpu ratio
100'000	50	GTX580	1	100'000	48.53	48.53	31.20	31.20	17.33	64.29%
100'000	50	Tesla2050	1	100'000	69.33	69.33	46.80	46.80	22.53	67.50%
100'000	50	GTX580	2	100'000	38.13	38.13	15.60	15.60	22.53	40.91%
100'000	100	GTX580	1	104'856	83.20	79.35	62.40	59.51	20.80	75.00%
100'000	100	Tesla2050	1	104'856	123.07	117.37	93.60	89.27	29.47	76.06%
100'000	100	GTX580	2	100'000	57.20	57.20	31.20	31.20	26.00	54.55%
1'000'000	50	GTX580	1	1'048'570	329.33	314.08	312.00	297.55	17.33	94.74%
1'000'000	50	Tesla2050	1	1'048'570	551.20	525.67	530.40	505.83	20.80	96.23%
1'000'000	50	GTX580	2	1'048'570	180.27	171.92	156.00	148.77	24.27	86.54%
1'000'000	100	GTX580	1	1'048'560	622.27	593.45	592.80	565.35	29.47	95.26%
1'000'000	100	Tesla2050	1	1'048'560	1031.34	983.57	998.40	952.16	32.93	96.81%
1'000'000	100	GTX580	2	1'048'560	331.07	315.73	296.40	282.67	34.67	89.53%

MC Workflow

QuantAlea



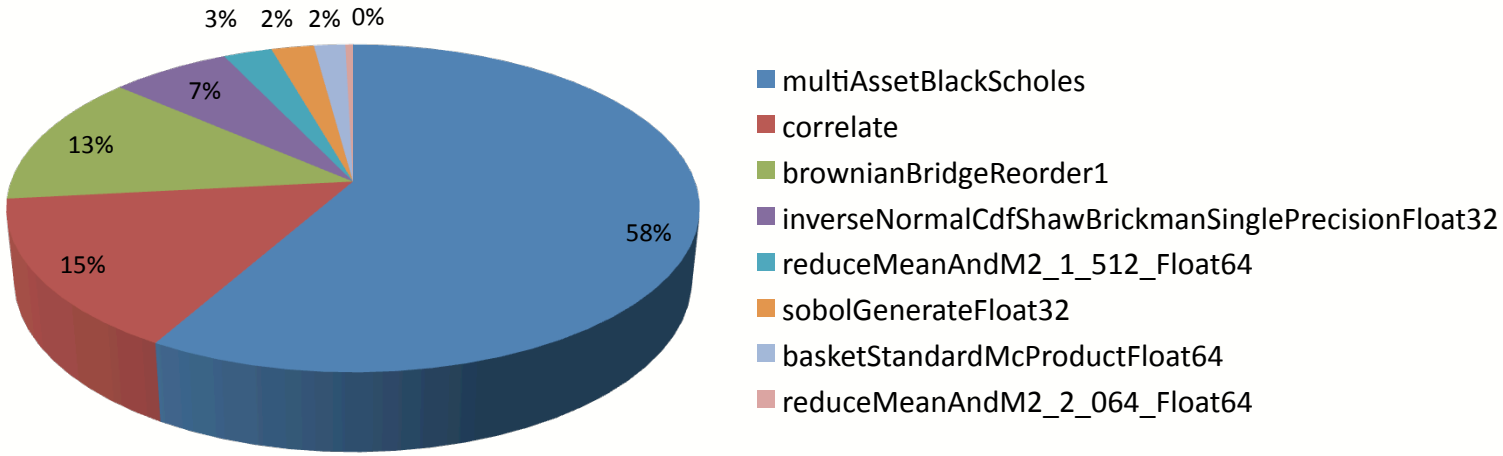
Iteration 1

- Sobol generation
- Inverse Normal
- Brownian bridge reordering
- Correlation twice
- Multiasset Black Scholes path stepper
- Basket standard product evaluator
- Reduce with Mean and M2

Iteration 2

…

Black Scholes - Standard Basket



Simple product with European payoff

- Path generation most significant, even with path reuse optimization
- Correlation and Brownian bridge reordering also important
- Inverse cumulative normal distribution also not negligible
- Payoff generation insignificant

MC Timings

QuantAlea

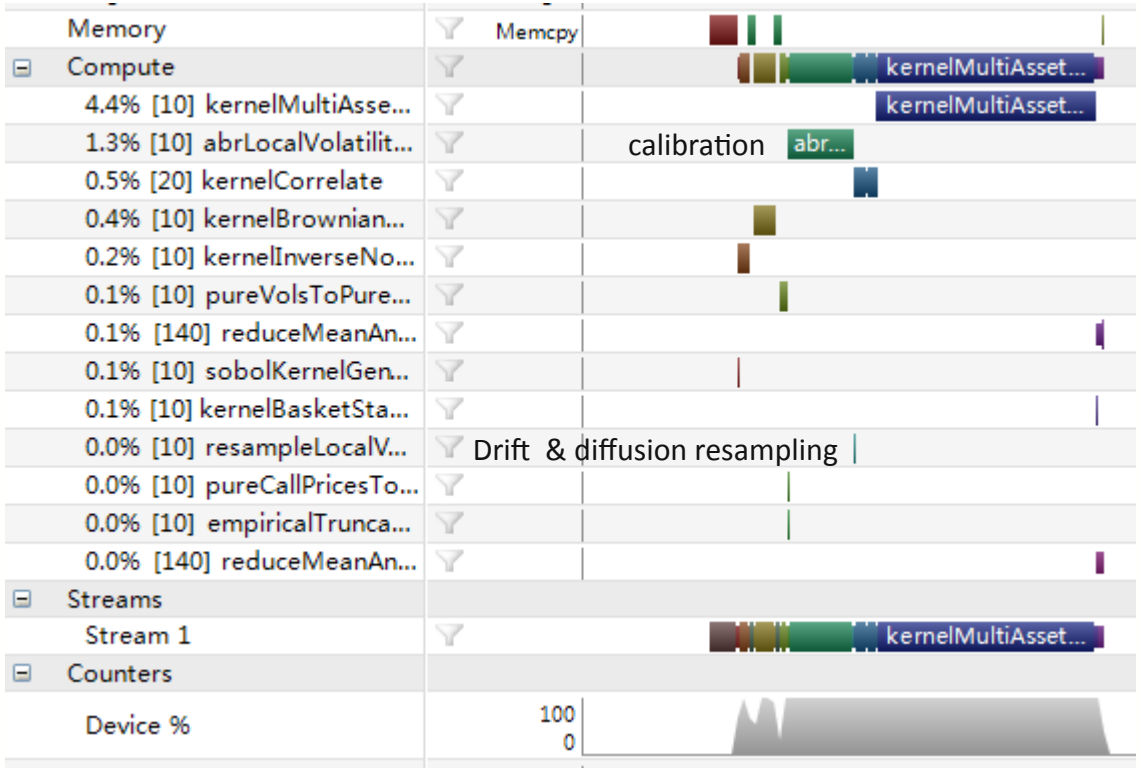
Basket of 4 assets

- Local Vol Log Spot
- Calculating price and Delta, Gamma, Vega, Correlation Delta
- Including calibration of local volatility for all asset and all perturbations
- Results in 4 x 5 = 20 local volatility surface calibrations
- Parallel local volatility calibration on CPU: +150ms
- No path optimization: + 550ms

samples	times	devices	n	acc samples	T (ms)	T scaled	T gpu	T gpu scaled	T prepare	cpu/gpu ratio
100'000	50	GTX580	1	100'000	79.73	79.73	62.40	62.40	17.33	78.26%
100'000	50	Tesla2050	1	100'000	83.20	83.20	62.40	62.40	20.80	75.00%
100'000	50	GTX580	2	100'000	62.40	62.40	31.20	31.20	31.20	50.00%
100'000	100	GTX580	1	104'856	147.33	140.51	109.20	104.14	38.13	74.12%
100'000	100	Tesla2050	1	104'856	188.93	180.18	140.40	133.90	48.53	74.31%
100'000	100	GTX580	2	100'000	102.27	102.27	62.40	62.40	39.87	61.02%
1'000'000	50	GTX580	1	1'048'570	570.55	544.12	546.00	520.71	24.54	95.70%
1'000'000	50	Tesla2050	1	1'048'570	691.88	659.83	655.20	624.85	36.68	94.70%
1'000'000	50	GTX580	2	1'048'570	299.87	285.98	280.80	267.79	19.07	93.64%
1'000'000	100	GTX580	1	1'048'560	1'118.56	1'066.76	1'076.40	1026.55	42.16	96.23%
1'000'000	100	Tesla2050	1	1'048'560	1'479.65	1'411.12	1'435.20	1368.74	44.44	97.00%
1'000'000	100	GTX580	2	1'048'560	592.80	565.35	546.00	520.72	46.80	92.11%

MC Workflow

QuantAlea



Iteration 1

- pureVols -> pureCallPrices
- pureCallPrices -> arrowDebreuPrices
- empiricalTruncationBound
- abrLocalVolatilityPure
- resampleLocalVolForLogSpot
- Sobol generation
- Inverse Normal
- Brownian bridge reordering
- Correlation twice
- Multiasset LocalVolLogSpot stepper
- Basket standard product evaluator
- Reduce with Mean and M2

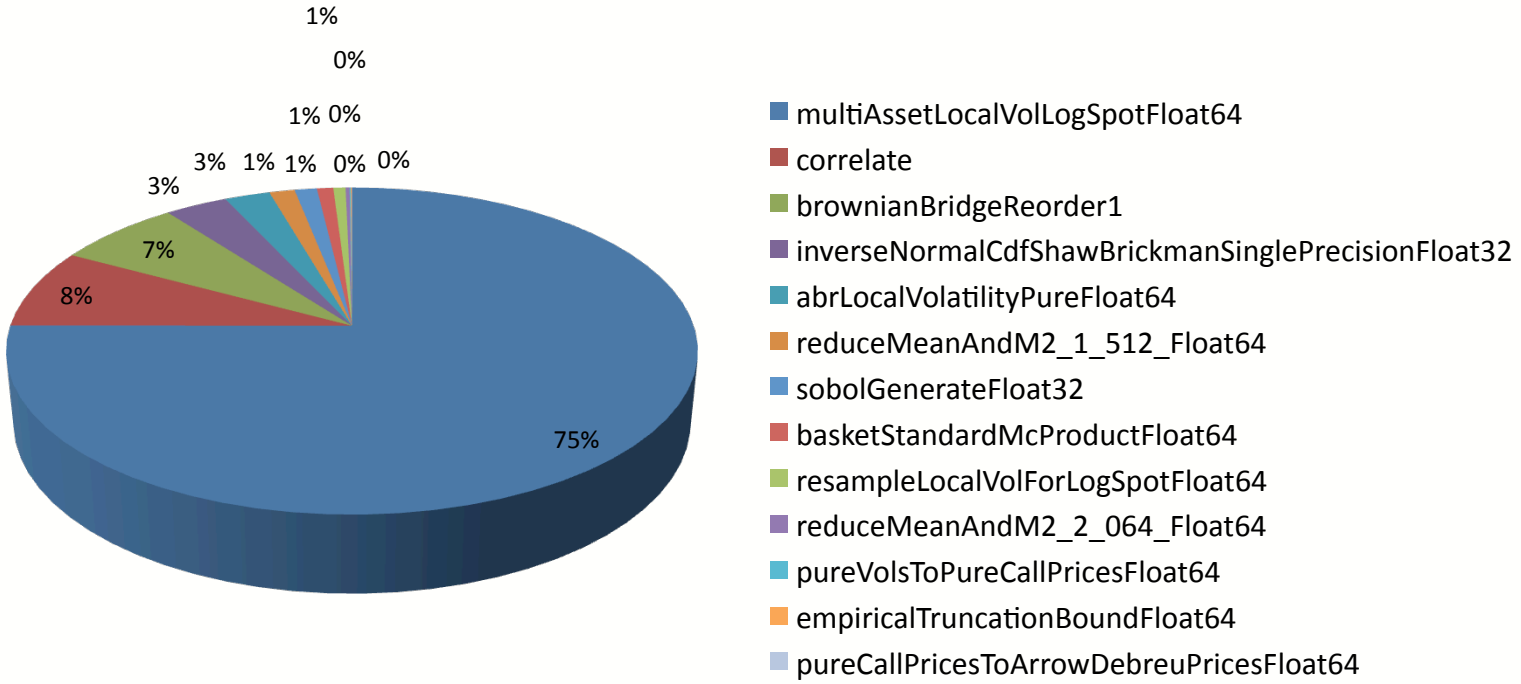
Iteration 2

- Sobol generation
- Inverse Normal
- Brownian bridge reordering
- Correlation twice
- Multiasset LocalVolLogSpot stepper
- Basket standard product evaluator
- Reduce with Mean and M2

Iteration 3

...

Local Vol - Standard Basket



Local volatility model

- Path generation dominant
- Parallel calibration of 20 local volatility surfaces on GPU very fast
- Path reuse optimization significant, also reducing number of LV calibrations
- Payoff generation insignificant

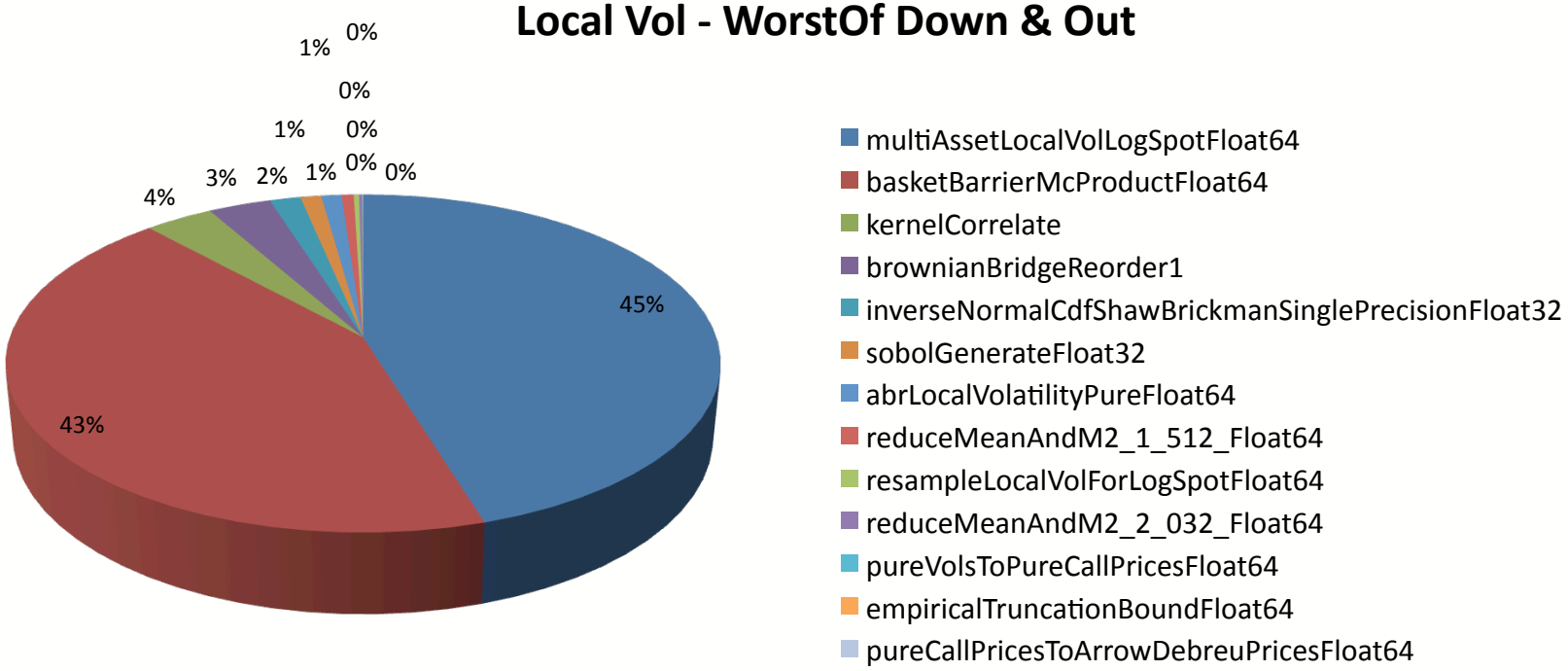
MC Timings

QuantAlea

Worst of down and in basket of 4 assets with 4 continuous barriers

- Local Vol Log Spot
- Calculating price and Delta, Gamma, Vega, Correlation Delta
- Barrier bias reduction leads to 4 additional states
- Timings including calibration of 20 local volatility for all asset and all perturbations

samples	times	devices	n	acc samples	T (ms)	T scaled	T gpu	T gpu scaled	T prepare	cpu/gpu ratio
100'000	50	GTX580	1	104'856	157.73	150.43	124.80	119.02	32.93	79.12%
100'000	50	Tesla2050	1	104'856	228.80	218.20	202.80	193.41	26.00	88.64%
100'000	50	GTX580	2	100'000	97.07	97.07	62.40	62.40	34.67	64.29%
100'000	100	GTX580	1	104'856	289.47	276.06	249.60	238.04	39.87	86.23%
100'000	100	Tesla2050	1	104'856	443.73	423.18	405.60	386.82	38.13	91.41%
100'000	100	GTX580	2	104'856	180.27	171.92	124.80	119.02	55.47	69.23%
1'000'000	50	GTX580	1	1'048'560	1'237.60	1'180.29	1'200.80	1145.19	36.80	97.03%
1'000'000	50	Tesla2050	1	1'048'560	2'003.74	1'910.94	1'965.60	1874.57	38.13	98.10%
1'000'000	50	GTX580	2	1'048'560	643.07	613.29	608.40	580.23	34.67	94.61%
1'000'000	100	GTX580	1	1'022'346	2'414.54	2'361.76	2'371.20	2319.38	43.33	98.21%
1'000'000	100	Tesla2050	1	1'022'346	3'922.54	3'836.80	3'884.41	3799.50	38.13	99.03%
1'000'000	100	GTX580	2	1'048'560	1'267.07	1'208.39	1'216.80	1160.45	50.27	96.03%



Complicated product with continuous barriers

- Path generation and payoff equally significant
- Path reuse optimization still pays off
- All other kernels negligible

MC GPU Implementation

QuantAlea

- Fast due to various algorithmic and implementation optimizations
  - Path reuse
  - Blob technology
  - Optimized GPU kernels
  - Multi GPU support
- Cube concept disentangles random number, path generation and payoff generation
  - Products can be evaluated under different model scenarios
  - Hybrid solutions mixing calculations on CPU and GPU
  - Integration of CPU based scripting into overall framework
- Sophisticated solution
  - Can handle complex data management
  - Can represent complex work flows like local volatility calibration
  - Allows interoperability of multiple kernels within framework
  - Dynamically dispatch to different steppers and evaluators
  - Seamless multi GPU support with async work flows

PDE Pricing

QuantAlea

- General purpose solver for multiple single asset options
  - Single factor problems
    - Single asset local volatility, 1 factor IR, ...
  - Pool many (>500) pricing problem to be processed as a batch in parallel

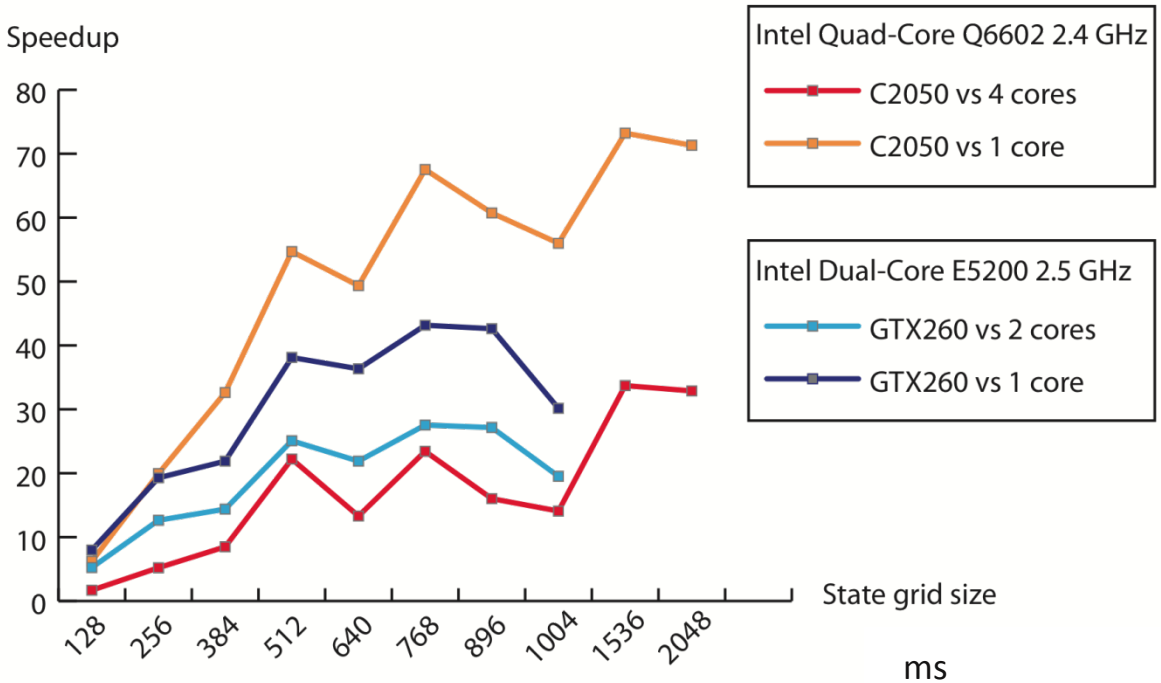
$$\frac{\partial V}{\partial t} + (r_t - q_t)s\frac{\partial V}{\partial s} + \frac{1}{2}\sigma_{\text{loc}}^2(t,s)s^2\frac{\partial^2 V}{\partial s^2} - r_t V = 0,$$

- Specific ADI solvers for two dimensional PDEs
  - Heston stochastic volatility
  - Basket of 2 assets
  - Hybrid equity / stochastic volatility / rates

$$\frac{\partial V}{\partial t} + \frac{1}{2}vs^2\frac{\partial^2 V}{\partial s^2} + \rho\sigma vs\frac{\partial^2 V}{\partial s \partial v} + \frac{1}{2}\sigma^2 v\frac{\partial^2 V}{\partial v^2} + (r_t - q_t)s\frac{\partial V}{\partial s} + (\kappa(\eta - v) - \Lambda(t,s,v))\frac{\partial V}{\partial v} - r_t V = 0$$

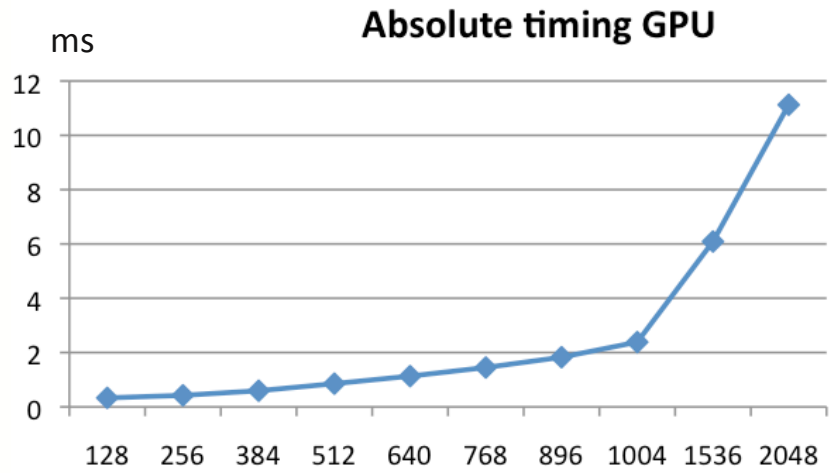
PDE Pricing ADI

QuantAlea



Implementation details:

- Multi-core with Intel TBB library
- GPU in single precision



Hedge Portfolio Search

QuantAlea

- Delta, Gamma, … of an exotic option should be matched
- Use n (~ 2 .. 10) hedge instruments for the hedge portfolio
- Filter rules can remove solutions from further consideration
  - Example {X > 0, Y < 0}, where X and Y are properties of the hedge portfolio
- Different selection criteria defines the order (top/bottom 100) of the hedges
  - Matching quality
  - Price of hedge
  - Liquidity of tradables

H	I	J	K	L	M	N
Equality Constraints			Filters		Selector	
Delta	55000		Theta	>0	Theta	Top 100
Vega	76000		Gamma	>0		
MaxNotional						
100000						

Filters

	A	B	C	D	E	F
1	Name	Premium	Delta	Gamma	Theta	Vega
2	SPOT	100	1	0	0	0
3	OPT100C	10.19688	0.996131	0.002828	1.219164	0.159546
4	OPT101C	0.002471	-0.00219	0.001813	0.681543	0.095516
5	OPT102P	5.264204	0.948538	0.031955	9.086083	1.46424
6	OPT103P	0.13025	-0.07586	0.03789	14.02935	1.981216
7	OPT104C	1.30963	0.531242	0.130866	31.51296	5.518374
8	OPT1C	1.077985	-0.4681	0.135291	30.47439	5.517653
9	OPT105C	0.090667	0.067267	0.04192	10.54397	1.806618
10	OPT2C	5.063376	-0.87191	0.05153	22.18284	2.905306
11	OPT106C	0.004492	0.004147	0.003475	1.123367	0.169788
12	OPT3C	9.809549	-0.99833	0.001698	0.445498	0.074748
13	OPT107C	8.1E-05	9.31E-05	0.000101	0.0355	0.005137
14	OPT4C	14.80809	-0.99993	7.81E-05	0.026515	0.00391
15	OPT108C	11.00304	0.947098	0.014879	4.692075	3.139975
16	OPT5C	0.109941	-0.03988	0.012889	3.439709	2.502228
17	OPT109C	6.922901	0.779492	0.035978	14.65946	8.630411
18	OPT6C	0.670506	-0.17788	0.038552	10.55721	7.581413
19	OPT110C	2.780671	0.568254	0.066773	13.88465	11.44247
20	OPT7C	2.637405	-0.44125	0.0516	18.10649	11.48669

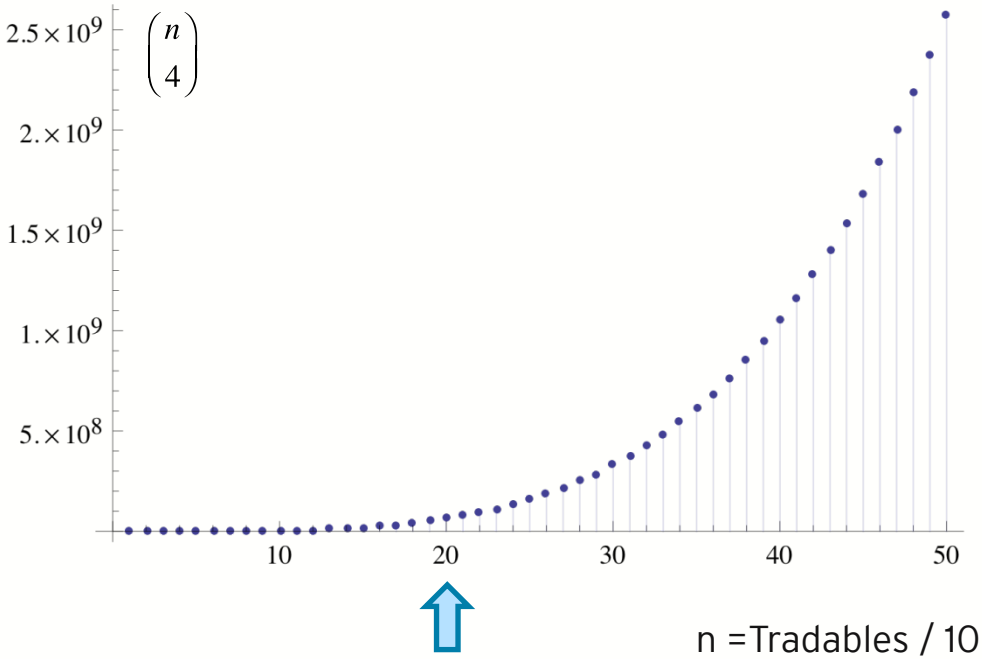
Hedge instruments

Hedge Portfolio Search

QuantAlea

- Solution requires full search
- Matrix A: row holds Greeks of a hedge instrument
- Hedge weights solution of Ax = b, b = Greeks of exotic option
- Solve many linear systems Ax = b for all possible hedge portfolios

Hedge size = 4 Tradables = 200 Combinations ~64.68 mio.	Time (seconds)	Normalized
search (GPU)	7.27	1.0
search_cpu (CPU)	309.94	42.63
search_cpu_mkl (CPU)	257.92	35.35



Thank you