

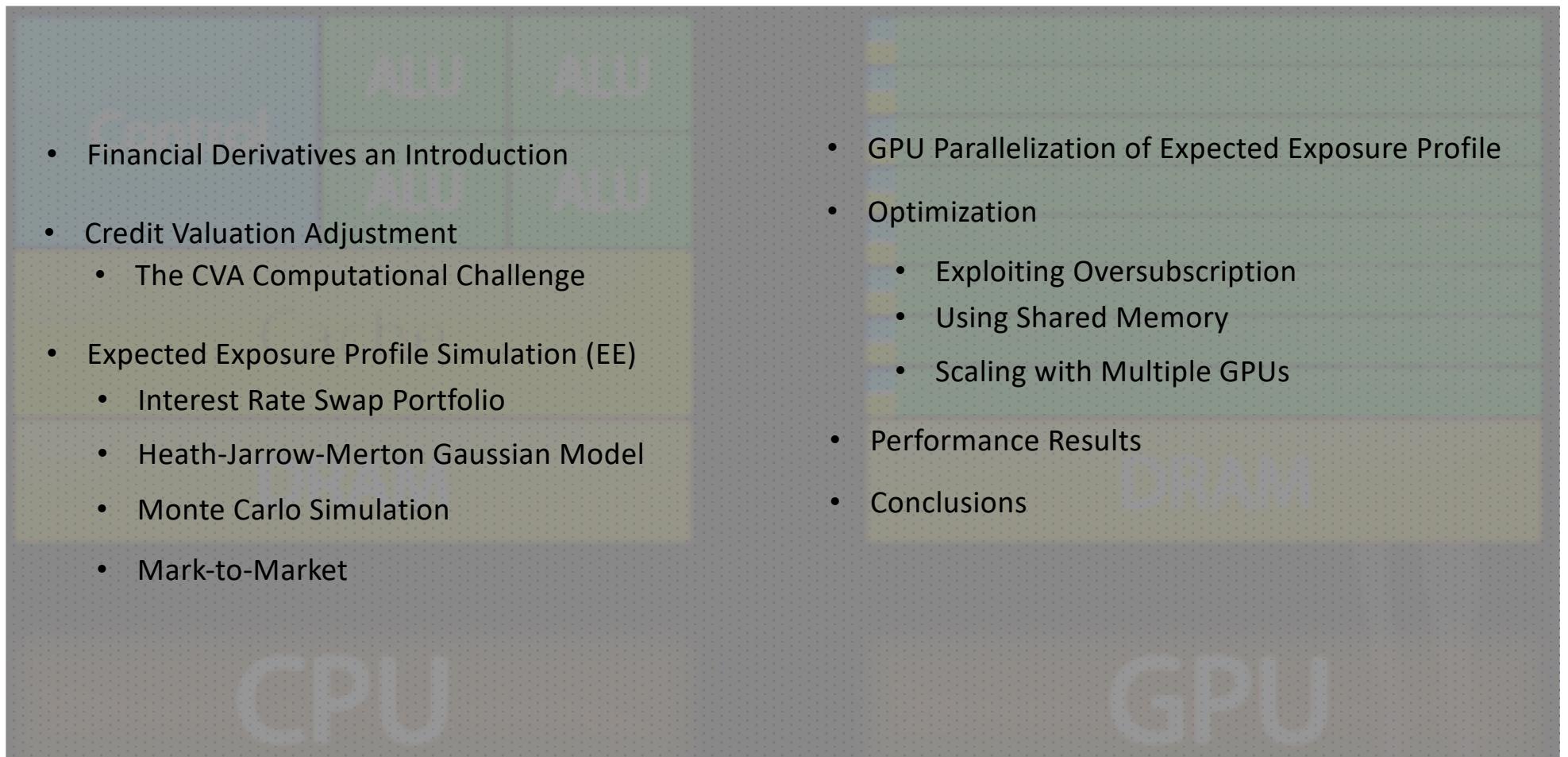


GPU-Accelerated Pricing of Credit Value Adjustment for Interest Rate Swap under a Gaussian HJM Model

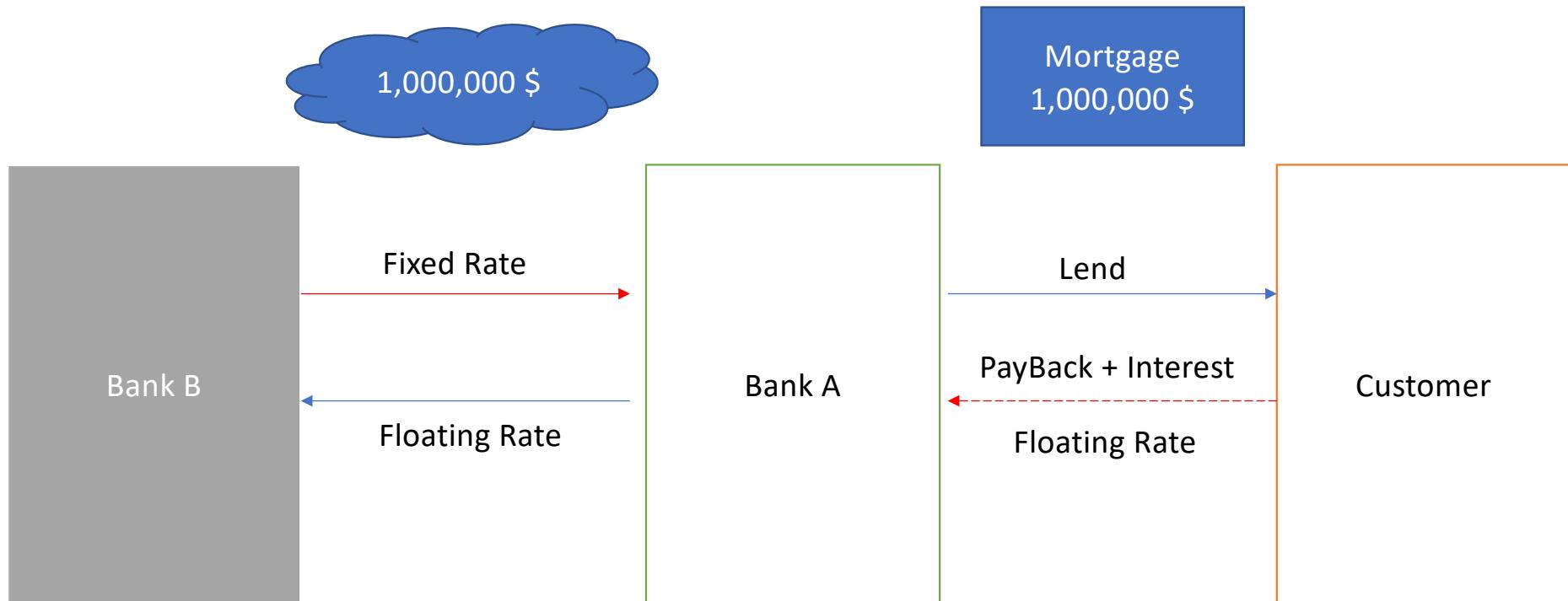
MASTER DISSERTATION

MASTER IN HIGH PERFORMANCE COMPUTING

Presentation Outline



Financial Derivatives an Introduction



Bank A enters into an Interest Rate Swap Contract with Bank B, receiving fix rates payments, to protect against a potential loss of money due to a lower value of interest rates.

Financial Derivatives an Introduction

Financial Derivatives traded in the OTC Market

- Fixed Income (Interest Rate)
- Foreign exchange
- Credit derivatives
- Commodity derivatives

Trading in the interest rate OTC market is done directly with a financial counterparty. Therefore, if any of the two-counterparty involved in the contract default the value of the contract can be considered a loss.

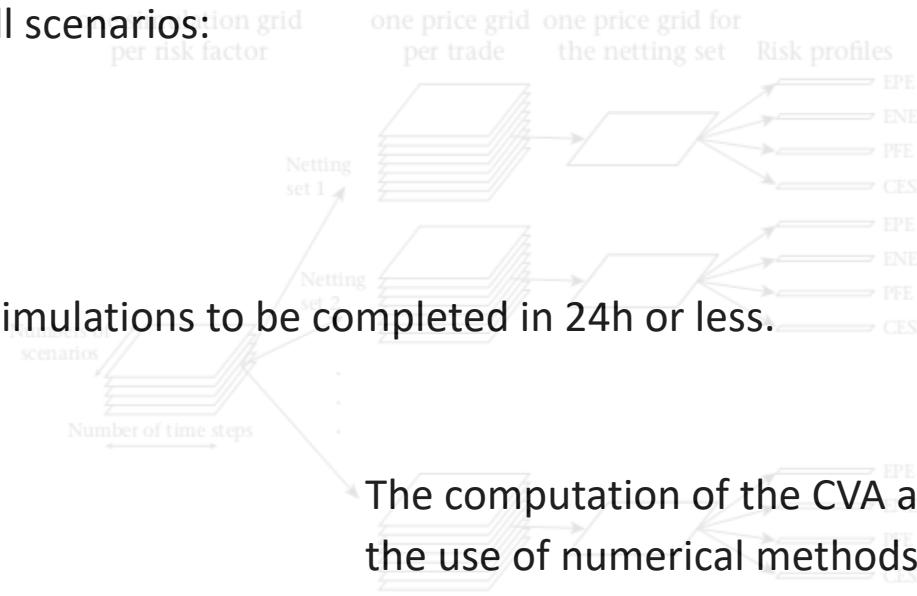
$$P_{risky} = P_{riskfree} - CVA$$

The CVA Computational Challenge

An investment bank generally needs to process for the bigger counterparties and for all scenarios:

- 1,000,000 of trades
- 5000 pricing points
- 400 scenarios

This results in 200TB of simulations to be completed in 24h or less.



The computation of the CVA and other risk profiles requires the use of numerical methods like Monte Carlo simulation, which demands a high number of computational resources and electricity.

Monte Carlo Simulation

Monte Carlo methods estimates the expected values of financial payoff functions based on underlying assets.

Monte Carlo methods suffer from slow convergence with respect to other simulation methods. His algorithmic is suitable to accelerate using heterogenous computing like NVIDIA GPU, achieving high speedup rates.

Credit Valuation Adjustment

$$CVA = LGD (1 - R) \sum_{k=0}^n DF(t_k, t) EE(t_k) PD(t_k, t_k - 1)$$

Where:

- LGD is the loss fraction (of total value), given default
- R is the recovery rate
- DF (t, t) Is the discount factor, discounting the value from time of default back to the valuation time.
- EE (t) is the Expected Exposure at time t
- PD (t_k, t_{k-1}) is the probability of default of the counterparty at time t.

Credit Valuation Adjustment

Computing CVA on a portfolio of interest rate swap can be separated on several steps:

- Calibration of underlying models to simple market instruments
- Generation of forward rate, discount factors and portfolio scenarios
- Computation of the exposure profile
- Discounting exposures to current time and weighting them with the corresponding default probabilities.

Expected Exposure Profile Simulation

Forward Rate Evolution (Risk Factor) evolution by simulating the Gaussian HJM SDE

$$r(t + dt) = r(t) + \left(\sum_{i=1}^k vol_n[t] \int_0^\tau vol_n[s] ds \right) dt + \sum_{n=1}^3 (vol_n[t] phi[n] \sqrt{dt}) + \frac{dF}{dt \tau} dt$$

Model:

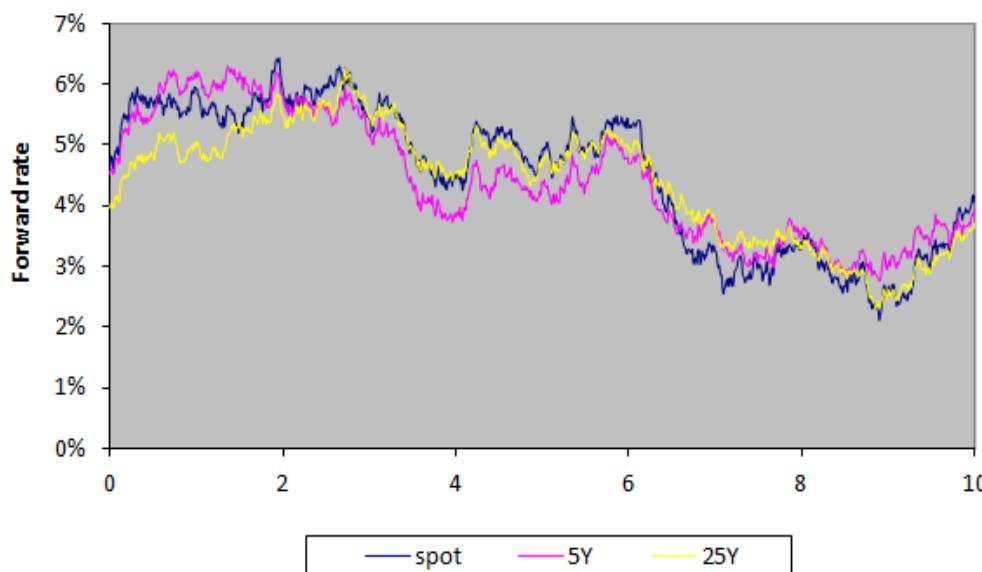
- $r_j(t+dt)$ is the forward rate change over time dt
- t takes values between 0 and T expiry (**25**)
- dt is the size of the simulation step (**0.01**)
- $d\tau$ is the difference between two consecutive tenors
- $vol_n[t]$ the calibrated volatility $i = 0..2$
- $\phi[n]$ is a gaussian variate with $N(0, 1)$
- dF represents the difference between two consecutive rates.

Monte Carlo Simulation:

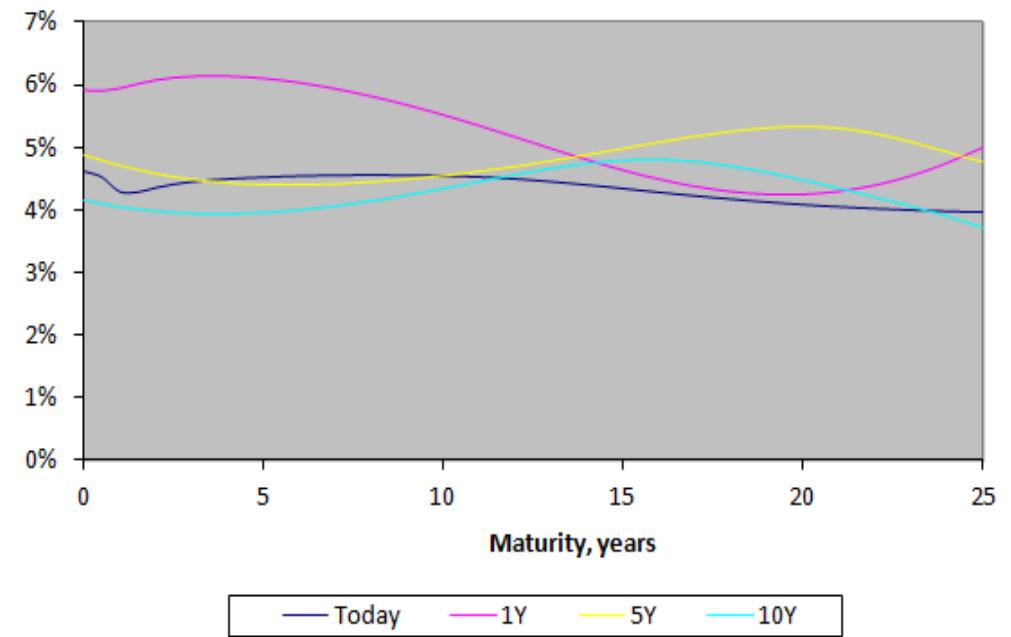
- Path generation using Euler Maruyama Numerical Scheme
- No variance reduction Scheme applied
- CURAND_RNG_PSEUDO_XORWOW pseudo random generator
- Number of simulation steps 2500, across 25 tenors
- Accuracy 0.000001

Expected Exposure Profile Simulation

Forward Rates Evolution



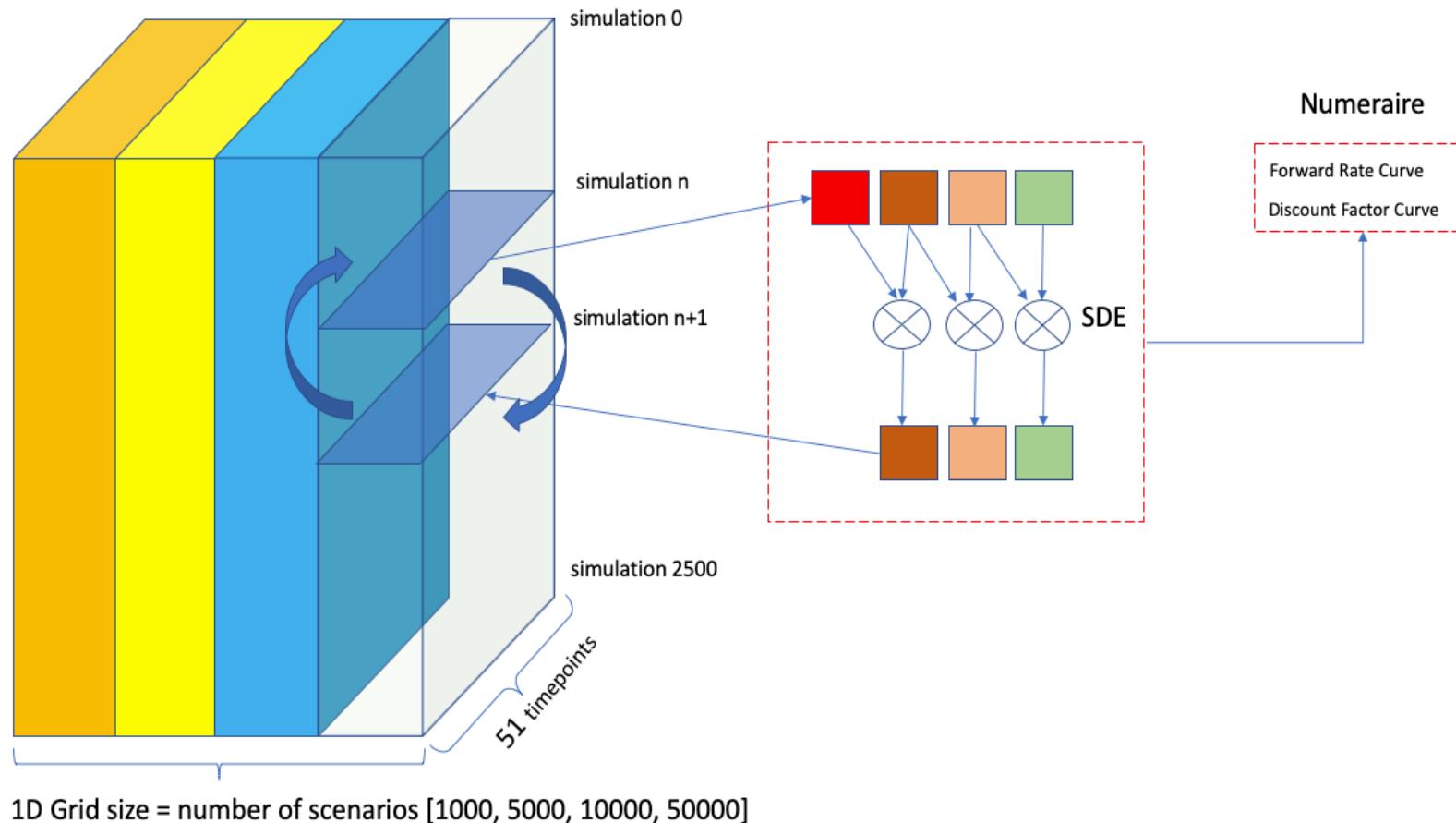
(a) Rate Evolution by Tenors



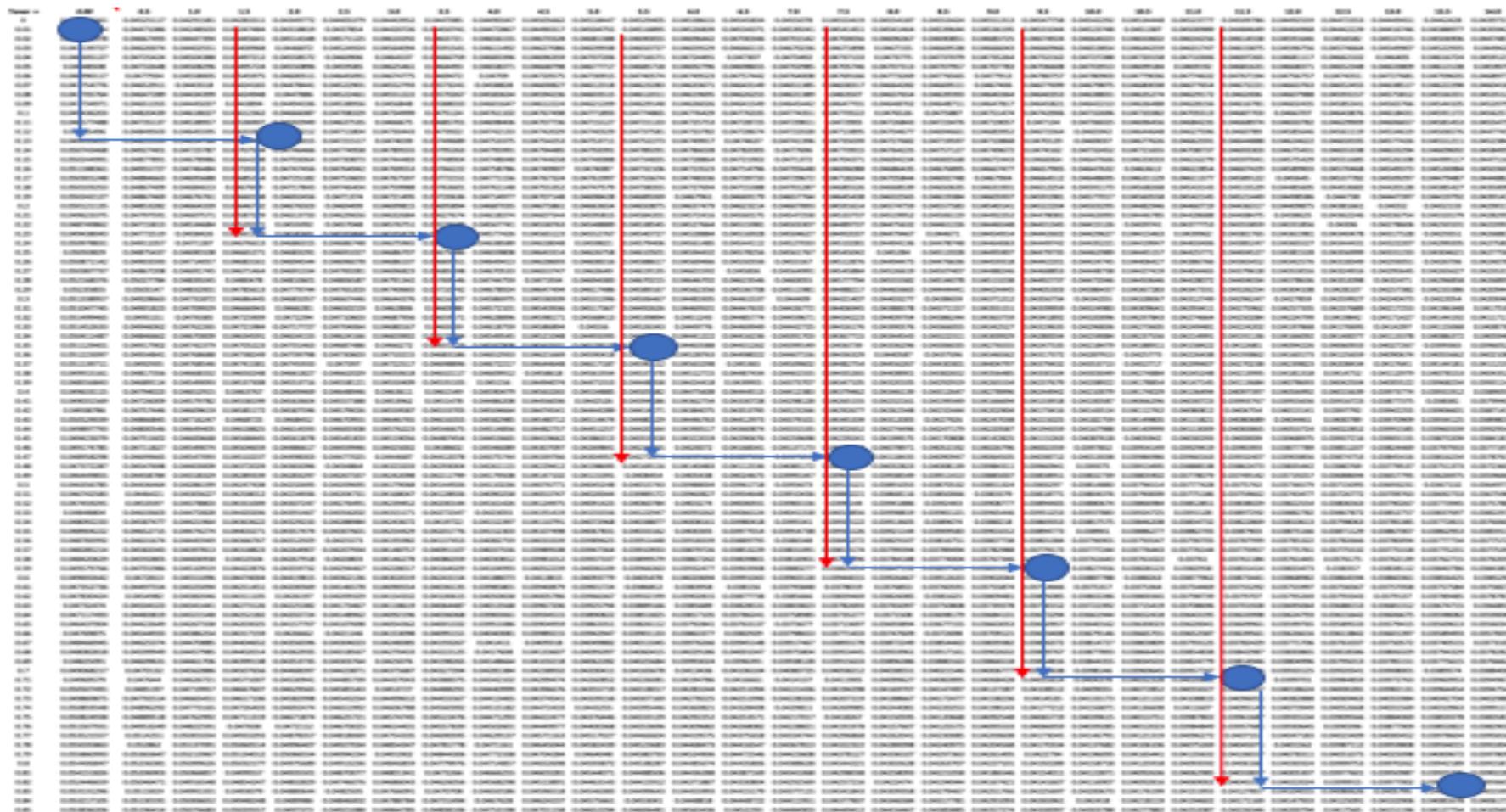
(b) Term Structure – Yield Curve

Expected Exposure Profile Simulation

Risk Factor Simulation Algorithm Implementation $O(N^3)$



Forward Rates Simulation Grid



Expected Exposure Profile Simulation

Discount Factors

$$Z(t, T) = \mathbb{E}^{\mathbb{Q}} \left[\exp \left(- \int_t^T r(s) ds \right) \right] = \exp \left(- \sum r_t \Delta t \right)$$

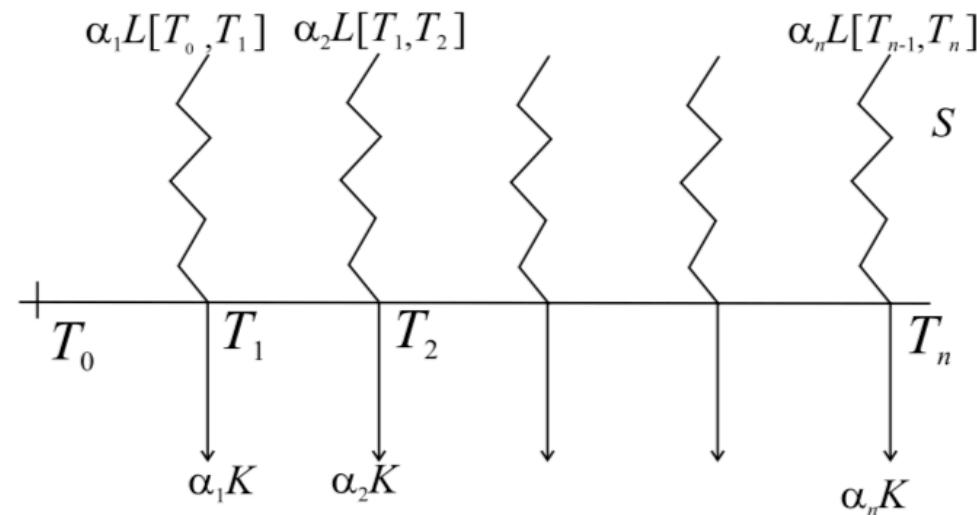
LIBOR Rate Calculation

$$L(t, T, S) = \frac{1}{\tau} * \left(\frac{B(t, T)}{B(t, S)} - 1 \right) = \frac{1}{\tau} * \left(\frac{e^{-\sum_{i=0}^T r(i) \Delta \tau}}{e^{-\sum_{i=0}^S r(i) \Delta \tau}} - 1 \right) \quad (4)$$

$$L(t; T, S) = \frac{1}{\tau} * (e^{-r(t) * \tau} - 1) \text{ where } \tau = T - S \quad (5)$$

Expected Exposure Profile Simulation

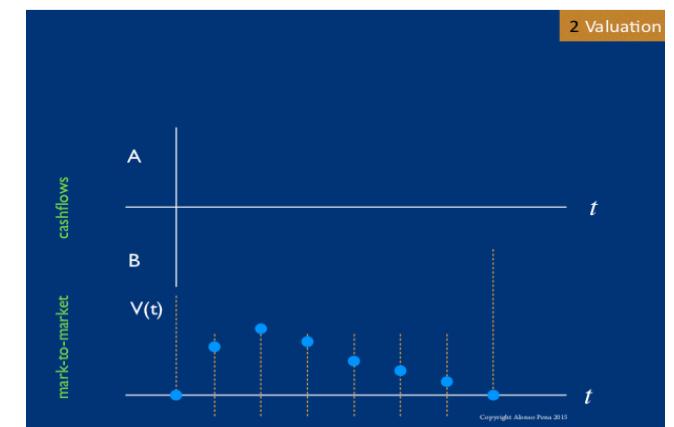
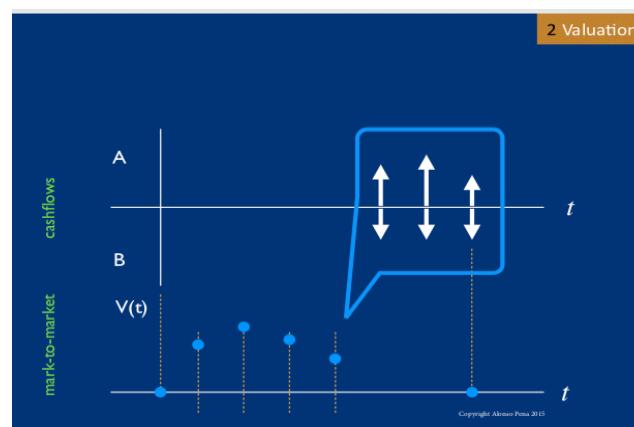
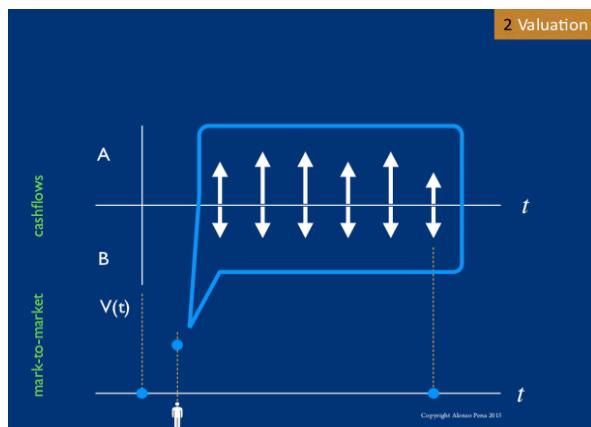
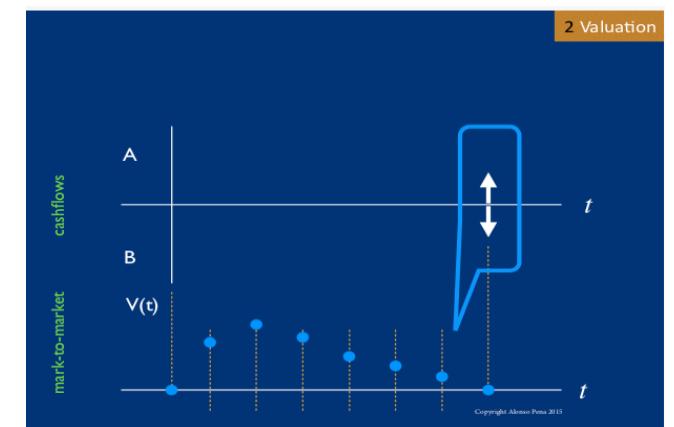
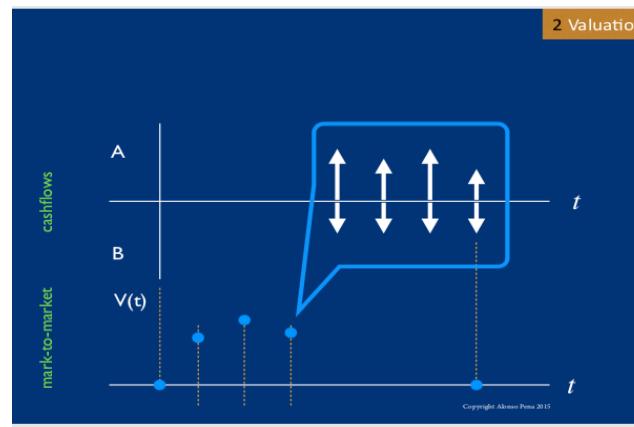
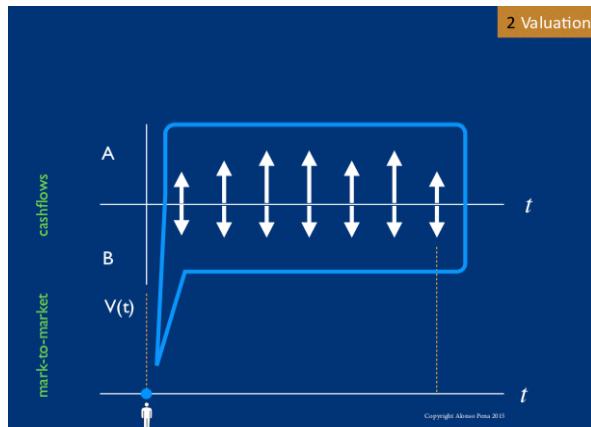
Vanilla Interest Rate Swap Net Present Value NPV(0)



$$PayOff(K; DF; L; T_1, \dots, T_n) = N \sum_{i=1}^n DF_i \delta_{i-1} (L(T_{i-1}, T_i) - K)$$

Expected Exposure Profile Simulation

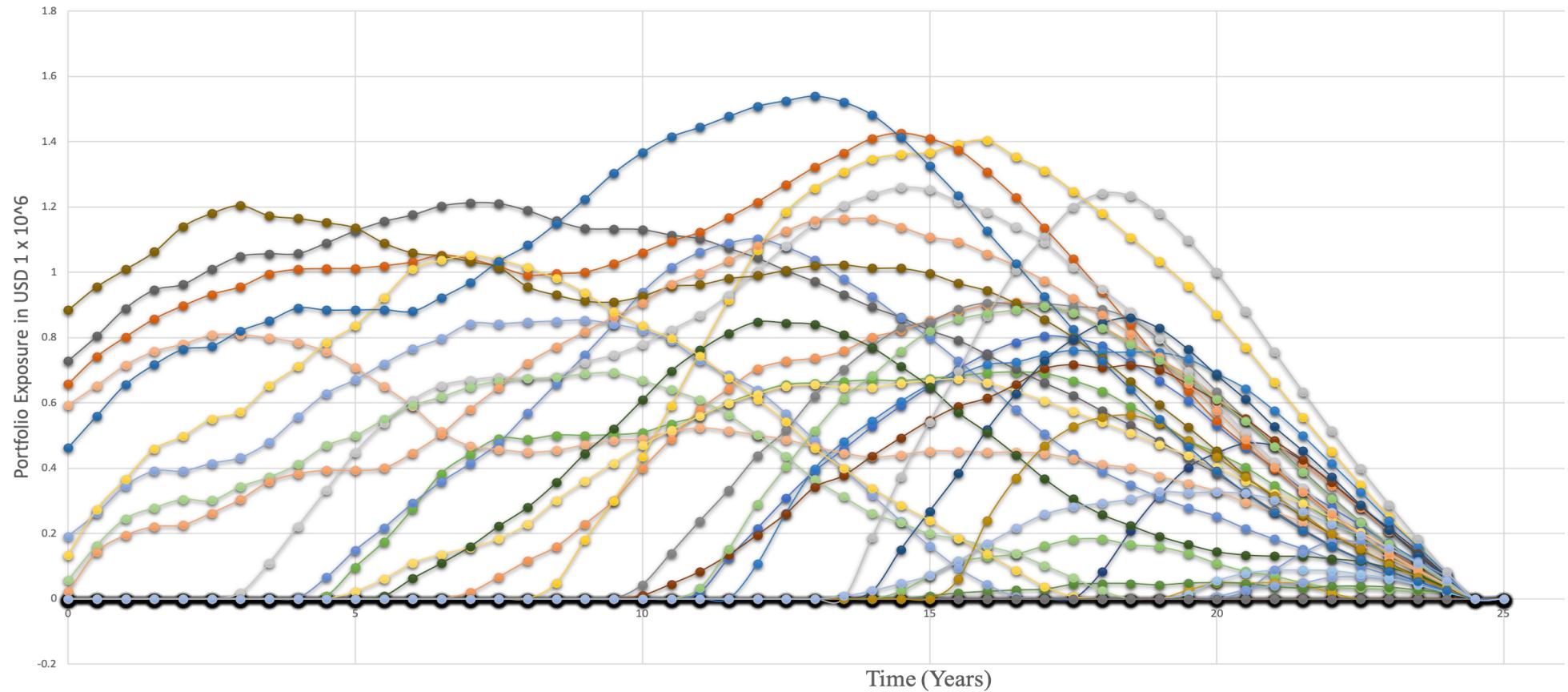
Mark-to-Market $V(t) = \max \{ 0, NPV(t) \}$



Expected Exposure Profile Simulation

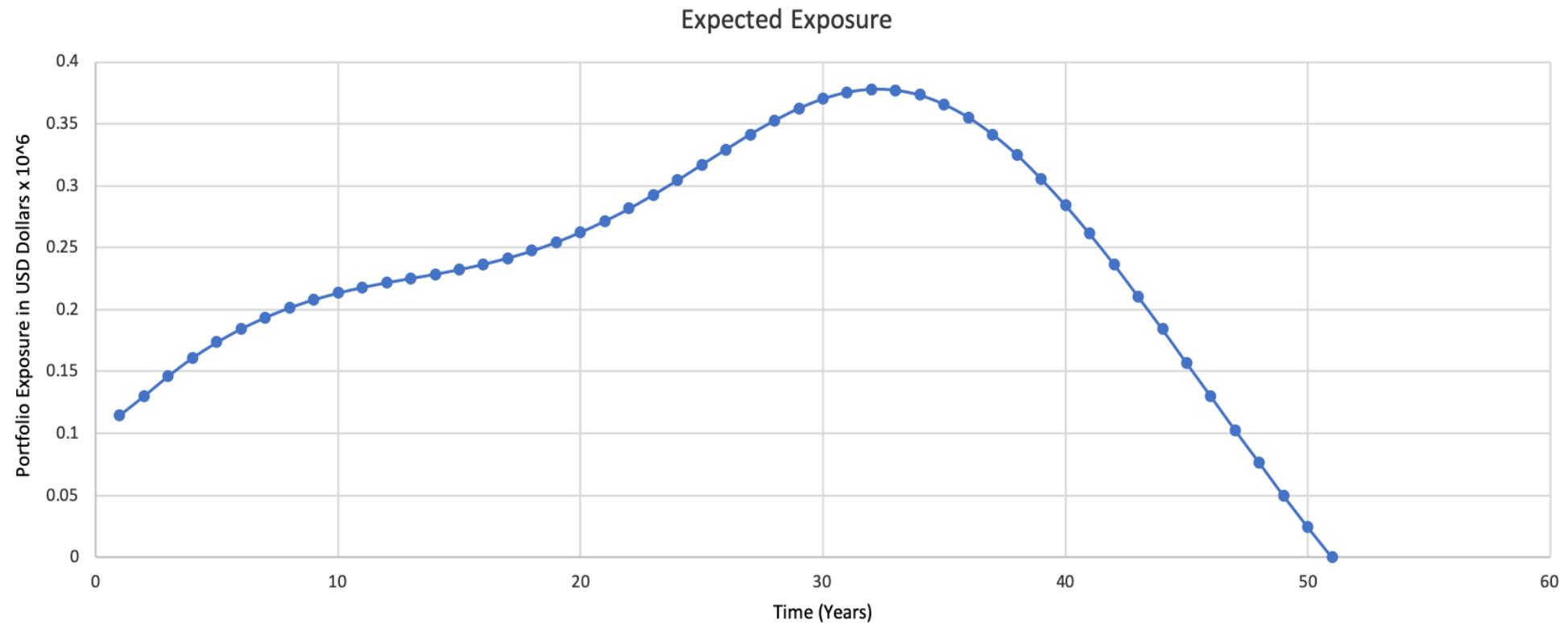
Monte Carlo Simulation of Expected Exposure Profiles (50,000)

Portfolio Exposure for a sample of Scenarios



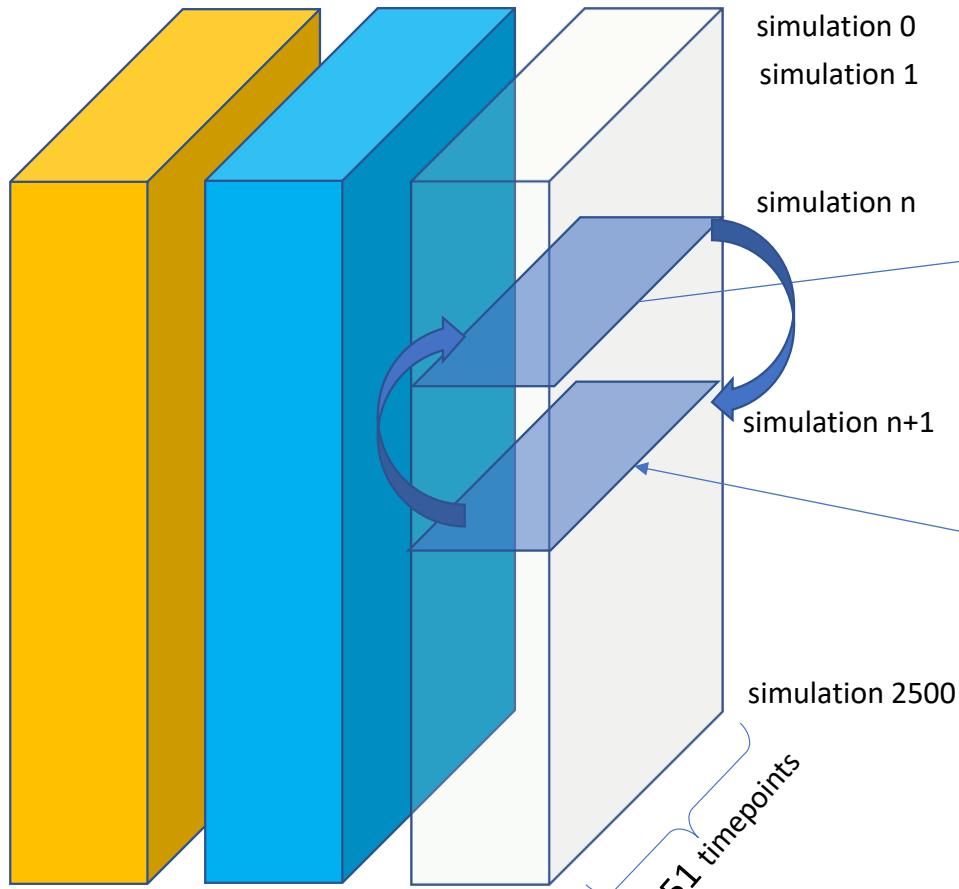
Expected Exposure Profile Simulation

Monte Carlo Simulation of Expected Exposure Profiles



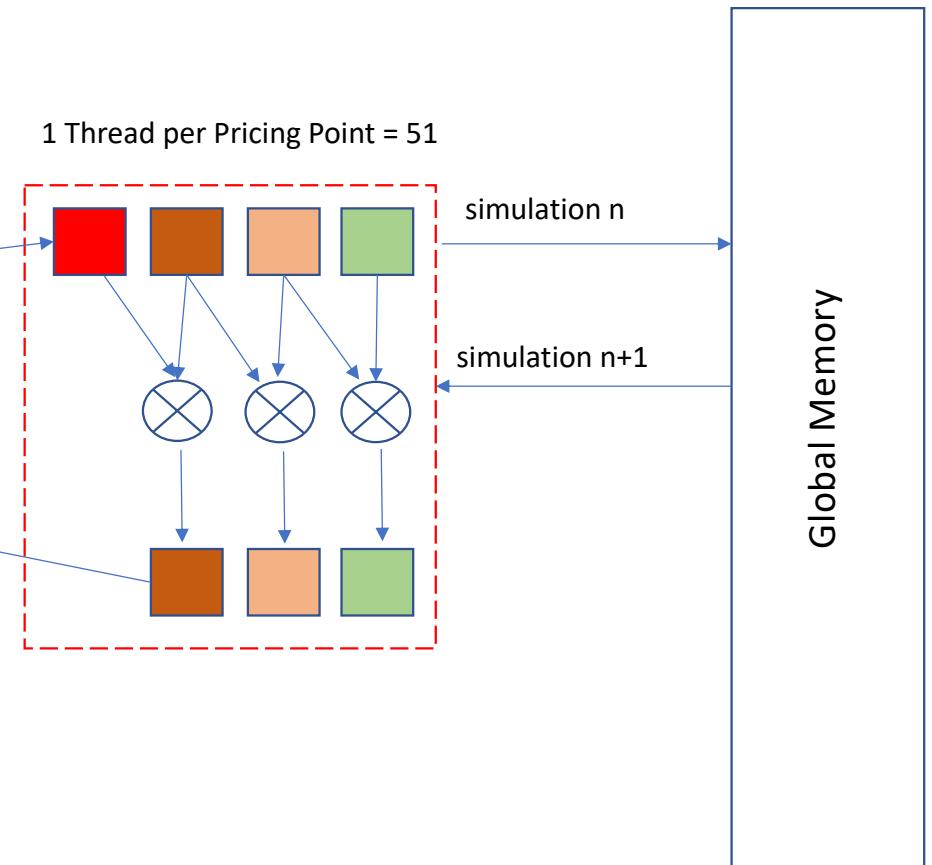
GPU Parallelization of Expected Exposure Profile

Loop Parallelization and Stencil parallel patterns



50,000 scenarios / 1 Thread Group Per Scenario

`__generate_paths_kernel<<number_scenarios, curve_size>>()`

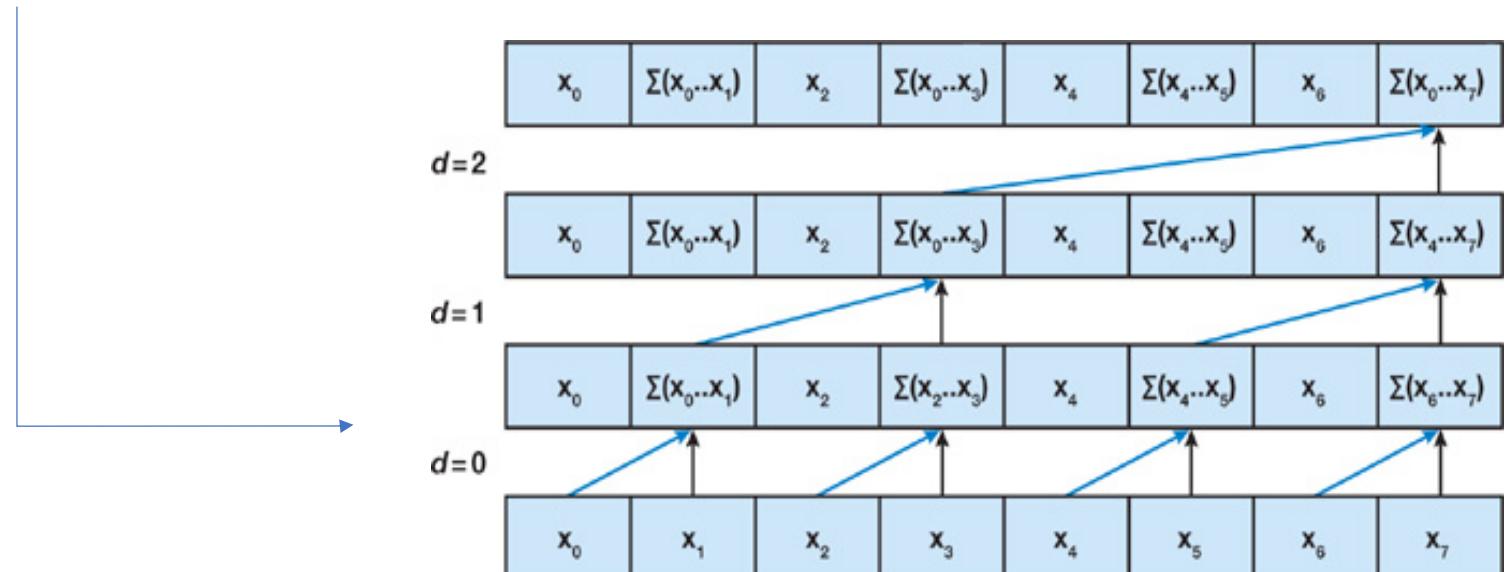


https://github.com/mahsanchez/cva_hjm_cuda

GPU Parallelization of Expected Exposure Profile

Interest Rate Swap Mark To Market using parallel Map and Prefix Sum patterns

```
forall j in parallel do  
    temp[j] = f(in[j]);  
all_prefix_sums(out, temp);
```



https://github.com/mahsanchez/cva_hjm_cuda

GPU Parallelization of Expected Exposure Profile

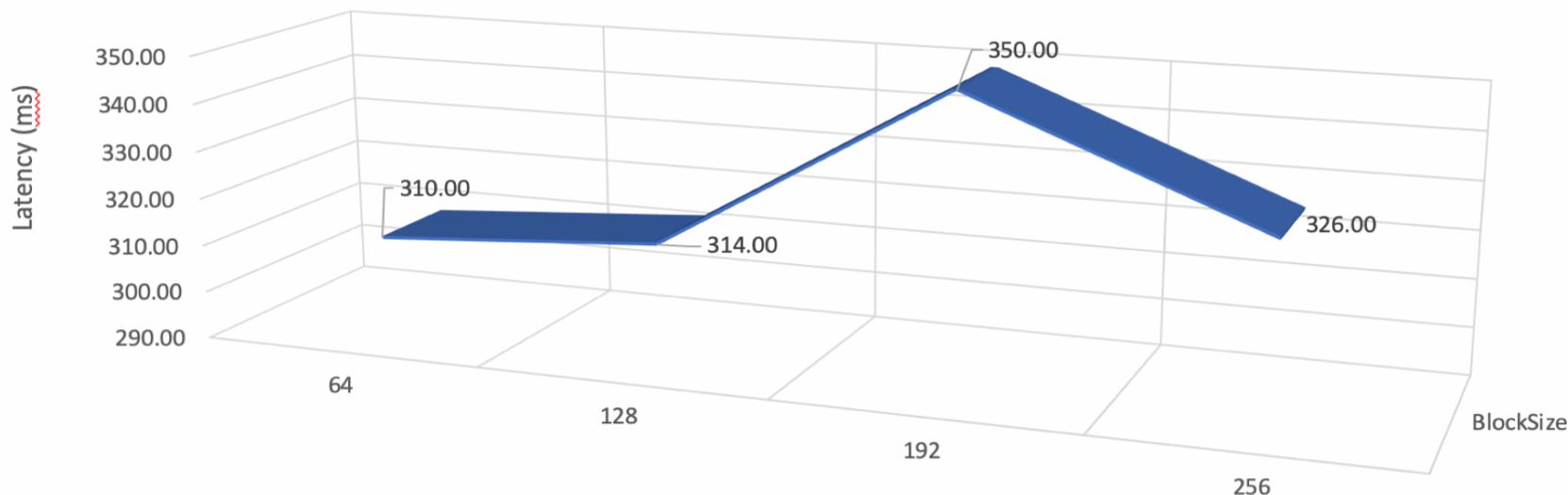
Parallel Column Summation Reduction using cublasDgemv (Matrix Vector Multiplication)

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} (1 \ 1 \ 1 \ \cdots \ 1_m) = (\sum_{k=1}^n a_{k,1} \ \ \sum_{k=1}^n a_{k,2} \ \ \sum_{k=1}^n a_{k,3} \ \ \cdots \ \ \sum_{k=1}^n a_{k,m})$$

[Accelerating Reduction and Scan Using Tensor Core Units](#)

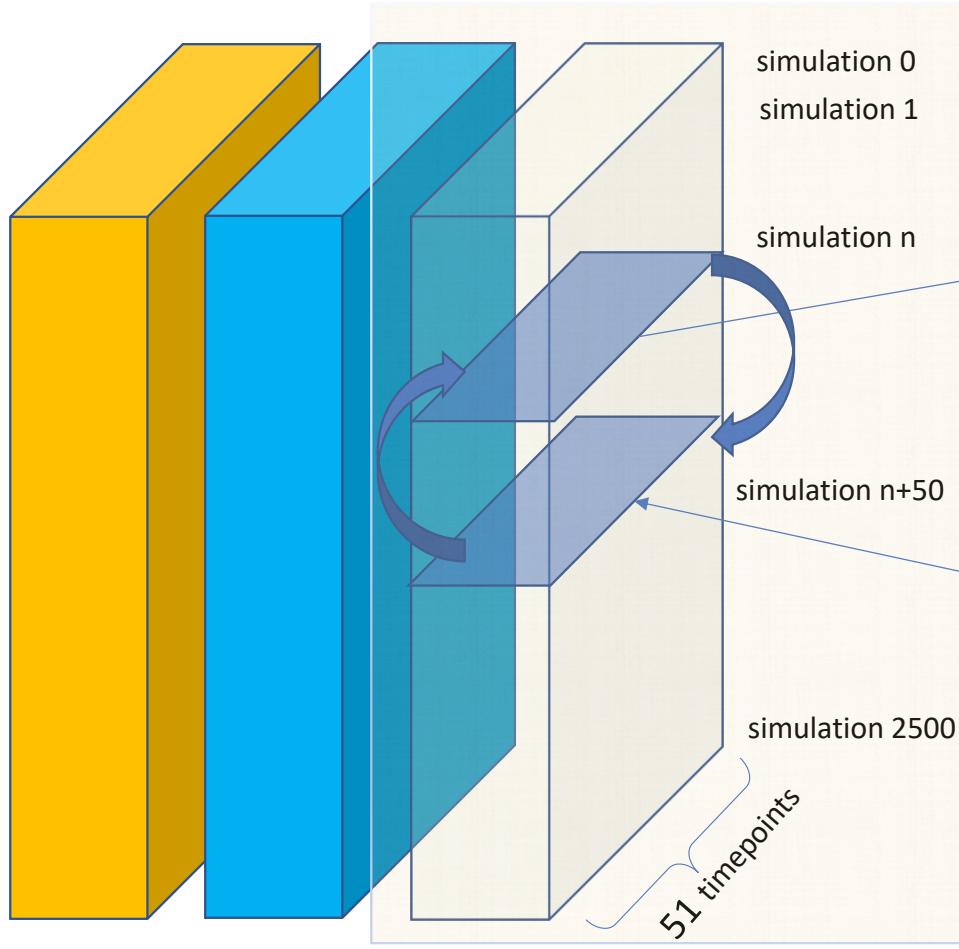
Optimization – Oversubscription

Latency Analysis per Thread Block Size



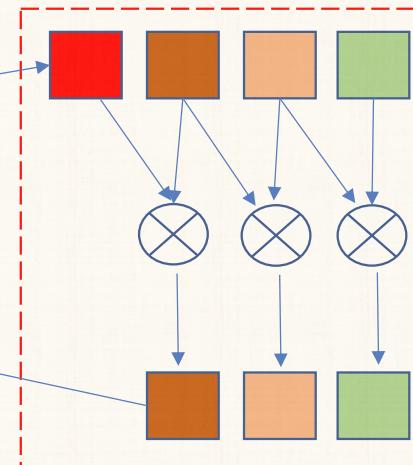
Optimization Shared Memory

Loop Parallelization and Stencil parallel patterns



`_generate_paths_kernel4<<number_scenarios, curve_size>>()`

1 Thread per Pricing Point = 51



simulation n

simulation n+1
simulation n+2
simulation n+3
...
simulation n+49

Shared
Memory

simulation n+50

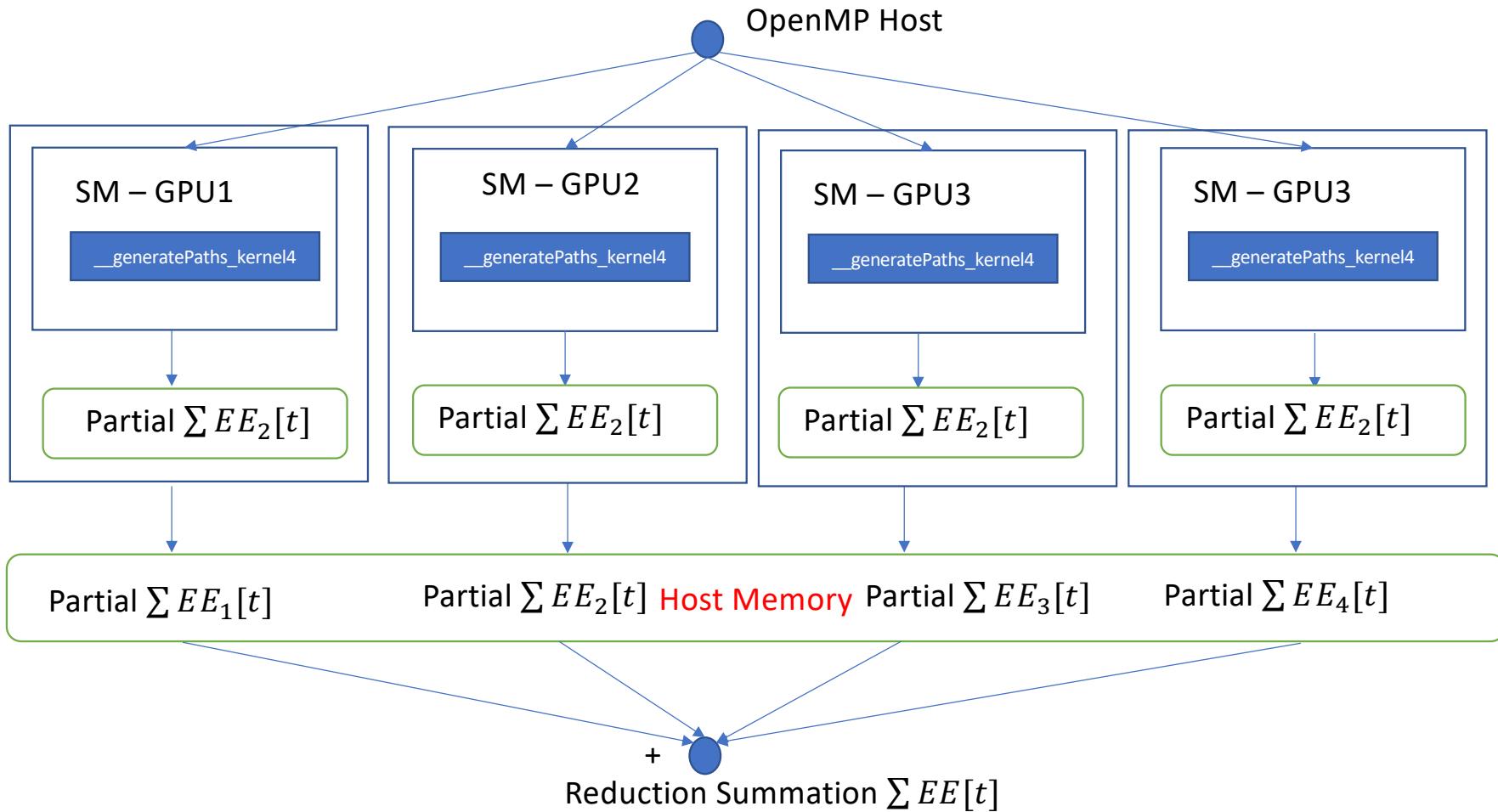
SM

Global Memory

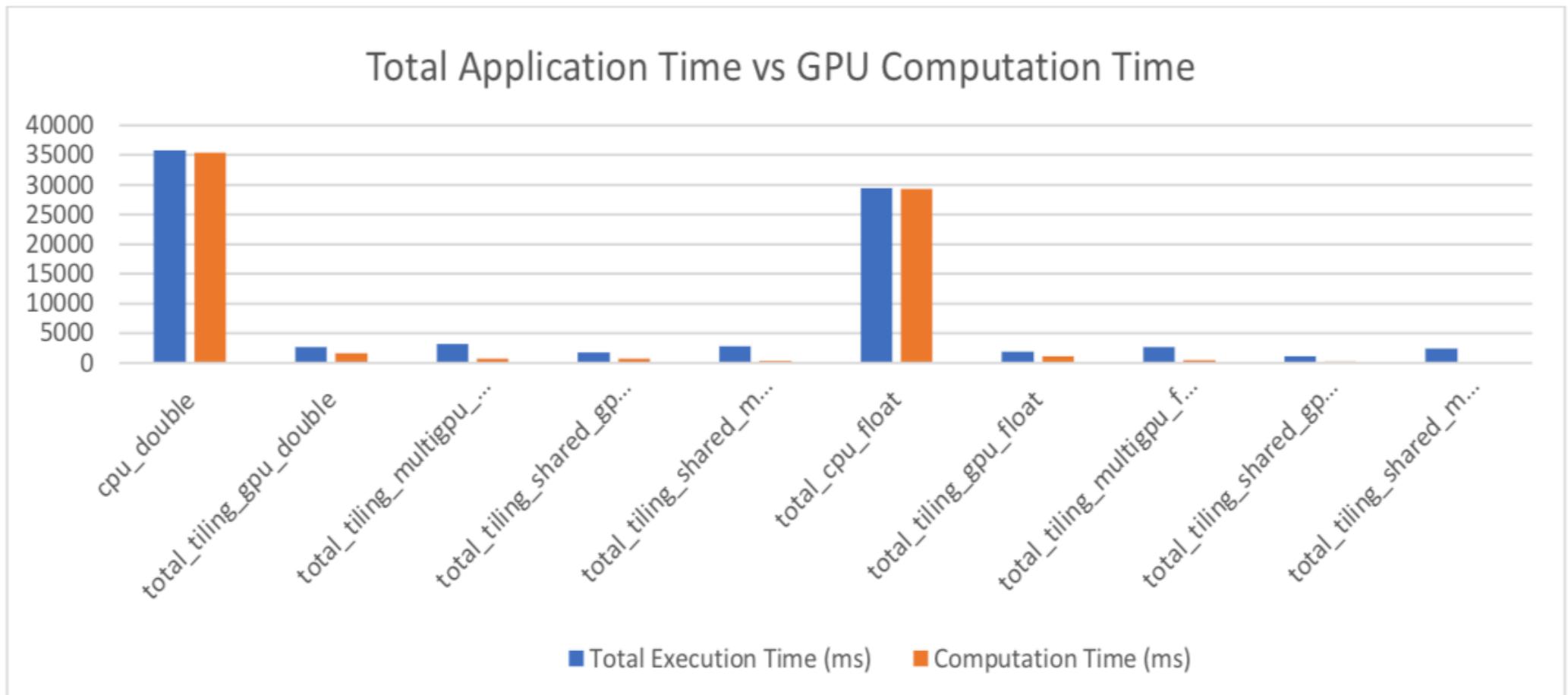
https://github.com/mahsanchez/cva_hjm_cuda

Optimization – Using Multiple GPUs

The Scenarios Generation is distributed across 4 GPUs



Performance Results



Performance Results

Latency Comparison between CPU, GPU and Multi-GPU (50,000 scenarios)

Implementation	Total application execution-time (ms)	Initialization-time (ms)	Computation-time (ms)	Computation -time Speed up
cpu_double	35777		35461	
tiling_GPU_double	2642	852.49	1583	22.40
tiling_multiGPU_double	3154	1816	500	70.92
tiling_shared_GPU_double	1765	772	543	65.30
tiling_shared_multiGPU_double	2846	1824	150	236.40
cpu_float	29479		29279	
tiling_GPU_float	1916	1837	967	30.27
tiling_multiGPU_float	2674	1837	342	85
tiling_shared_GPU_float	1064	751	157.486	185.91
tiling_shared_multiGPU_float	2153	1808	46	636

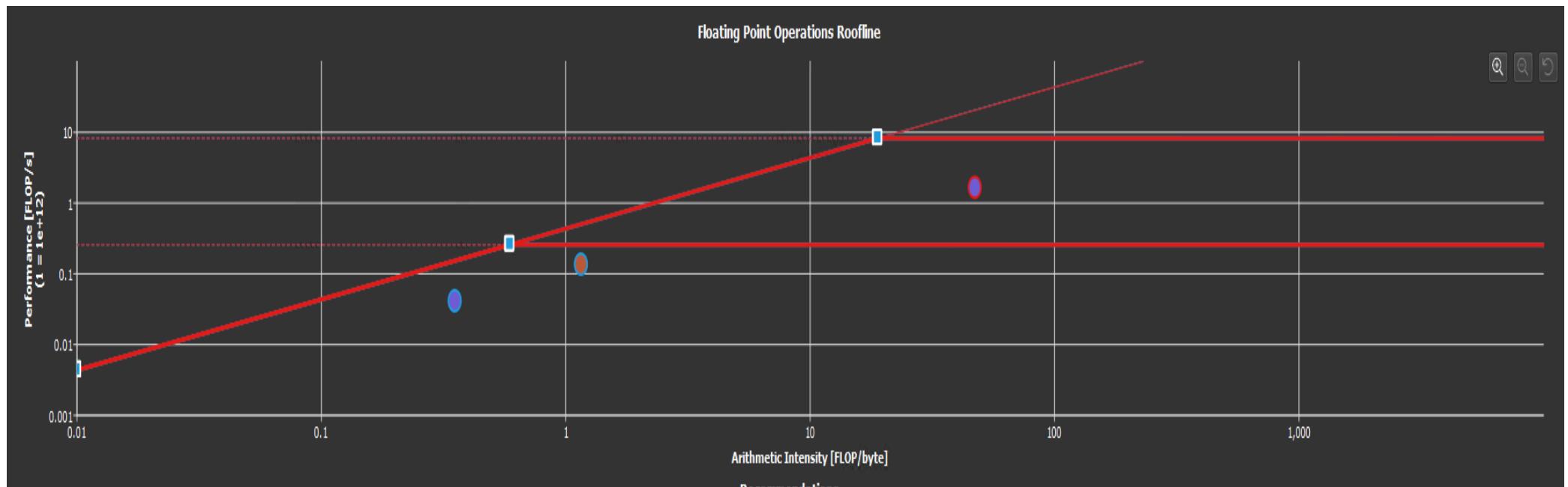
Performance Results

Floating Point Accuracy Analysis (50,000 scenarios)

Implementation	Mean-square errors	Error
tiling_GPU_double	0.01377044	10^-2
tiling_multiGPU_double	0.01397286	10^-2
tiling_shared_GPU_double	0	10^-6
tiling_shared_multiGPU_double	0.00059256	10^-4
tiling_GPU_float	0.01380047	10^-2
tiling_multiGPU_float	0.0143285	10^-2
tiling_shared_GPU_float	0.000006	10^-6
tiling_shared_multiGPU_float	0.003549	10^-3

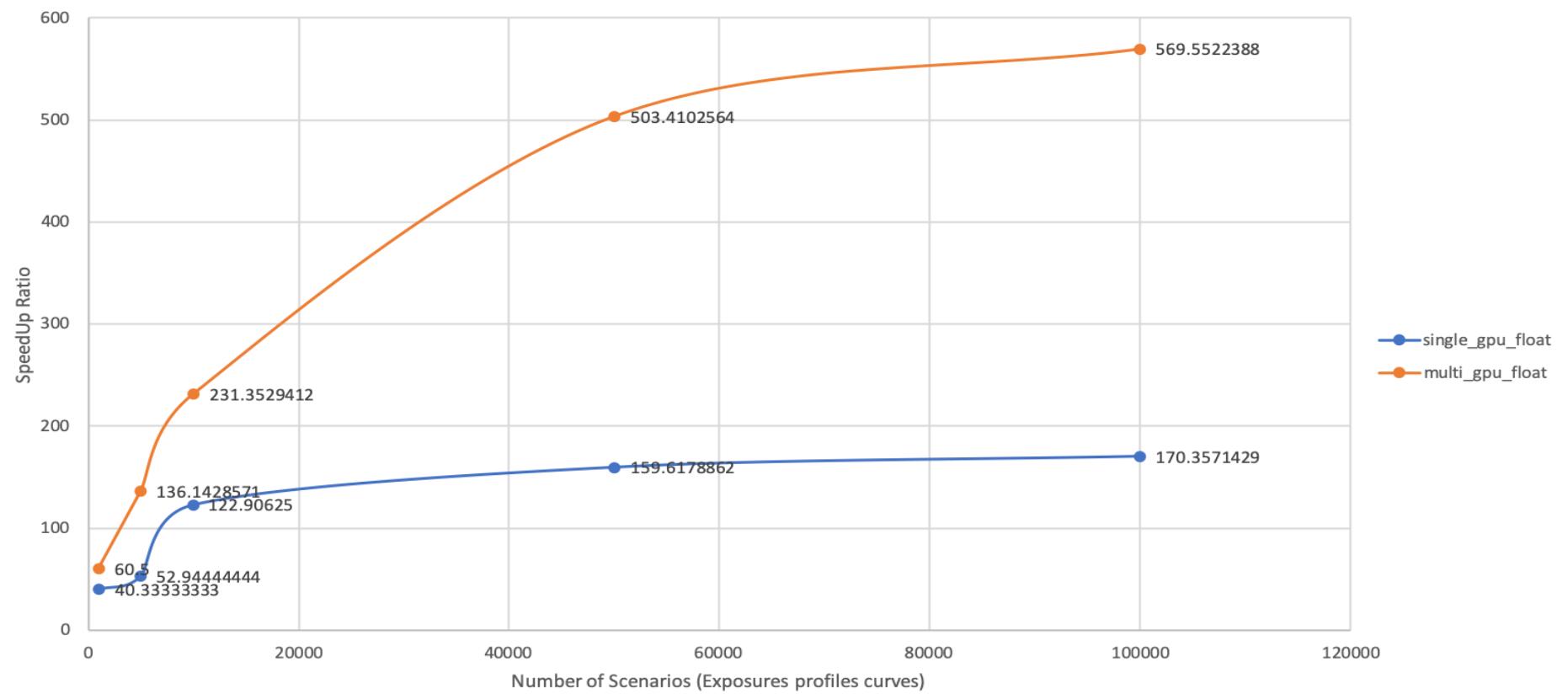
Performance Results

Roof Line Analysis Naïve Parallelization vs Shared Memory Optimization (50,000 scenarios)



Performance Results

Speed Up ratio of kernel __generatePaths_kernel4 on NVIDIA GeForce RTX 2070 fp 32
in/out fp 32 compute



Conclusions

- GPU/CUDA programming is a great technology to accelerate MC Simulation methods.
- The GPU accelerated version of the code outperform the serial reference CPU implementation.
- As the number of scenarios increase the application achieved a linear scalability by using multiples GPU.
- A single GPU implementation achieved better accuracy results.
- Possible Extensions of this work could potentially be related with the usage of CUDA Streams, improving the floating point accuracy with fixing point methods and complete the simulation with sensitivity analysis using Adjoint Algorithmic Differentiation technique.
- Heterogeneous Computing is great technology to tackle challenges problems like Credit Valuation Adjustment Calculation for portfolios in Fixed Income markets.