Facultad de Informática de A Coruña
UNIVERSIDADE DA CORUÑA

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

MASTER DISSERTATION

MASTER IN HIGH PERFORMANCE COMPUTING

Student: Michel Agustin Herrera Sanchez

Supervisor: Dr. Diego Andrade Canosa

Cohort 2020/2021

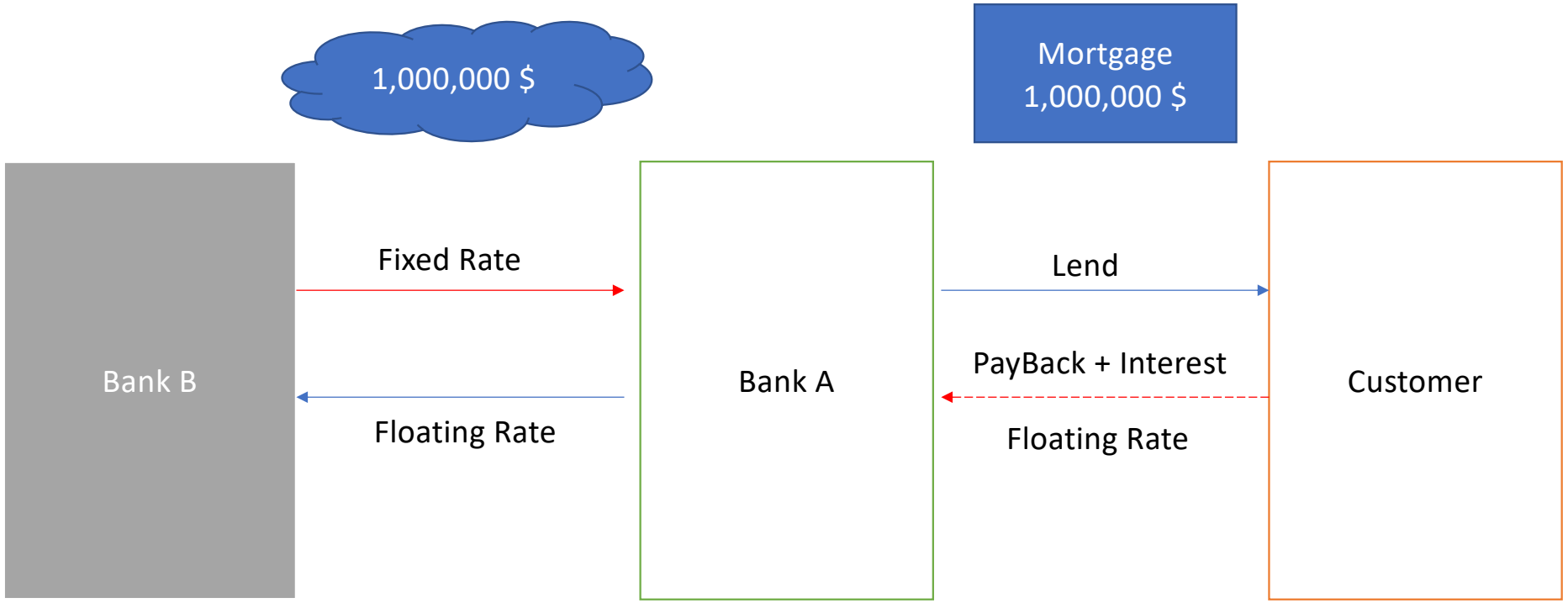
The purpose of this study is the GPU acceleration of simulation problems, under certain level of floating point accuracy, in the context of Computational Finance using NVIDIA massively parallel processors.

Part of the motivation of this research came from my daily job interaction with the topic of Counterparty Credit Risk.

Presentation Outline

- Financial Derivatives an Introduction
- Credit Valuation Adjustment
  - The CVA Computational Challenge
- Expected Exposure Profile Simulation (EE)
  - Interest Rate Swap Portfolio
  - Heath-Jarrow-Merton Gaussian Model
  - Monte Carlo Simulation
  - Mark-to-Market
- GPU Parallelization of Expected Exposure Profile
- Optimization
  - Exploiting Oversubscription
  - Using Shared Memory
  - Scaling with Multiple GPUs
- Performance Results
- Conclusions

Financial Derivatives an Introduction



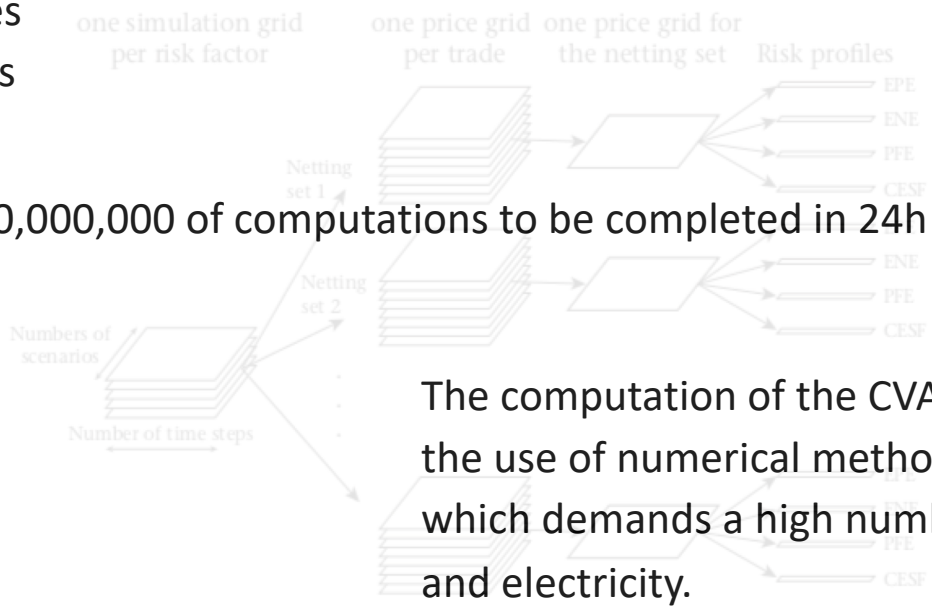
Bank A enter into an Interest Rate Swap Contract with Bank B, receiving fix rates payments, to protect against a potential lose of money due to a lower value of interest rates.

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 200,000,000,000 of computations 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.

Credit Valuation Adjustment

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

Where:

- 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 τ, $\tau \leq T$

Expected Exposure Profile Simulation

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

$$r(t+dt) = r(t) + \left(\sum_{n=1}^{3} vol_n[t] \int_t^T vol_n[s]ds\right) dt + \sum_{n=1}^{3}\left(vol_n[t]\, phi[n]\, \sqrt{dt}\,\right) + \frac{dF}{dtau}\, dt$$

Model:

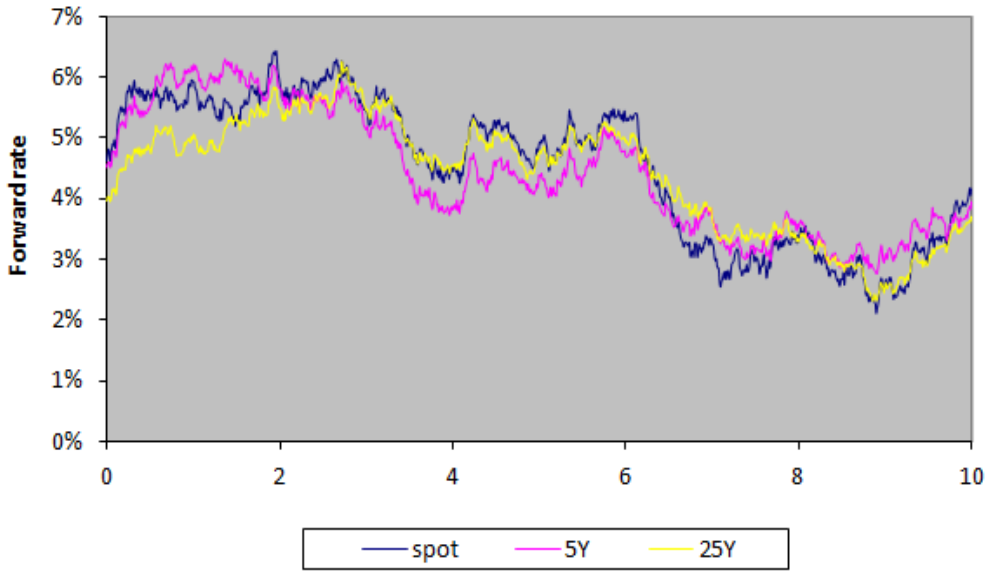
- $r_j(t+dt)$ is the forward rate change over time dt
- t takes values between 0 and T expiry ***(25)***
- dt is the size of the simulation step ***(0.01)***
- dtau is the difference between two consecutives tenors
- $vol_n[t]$ the calibrated volatility i = 0..2 from historic market data
- phi[n] is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

Monte Carlo Simulation:

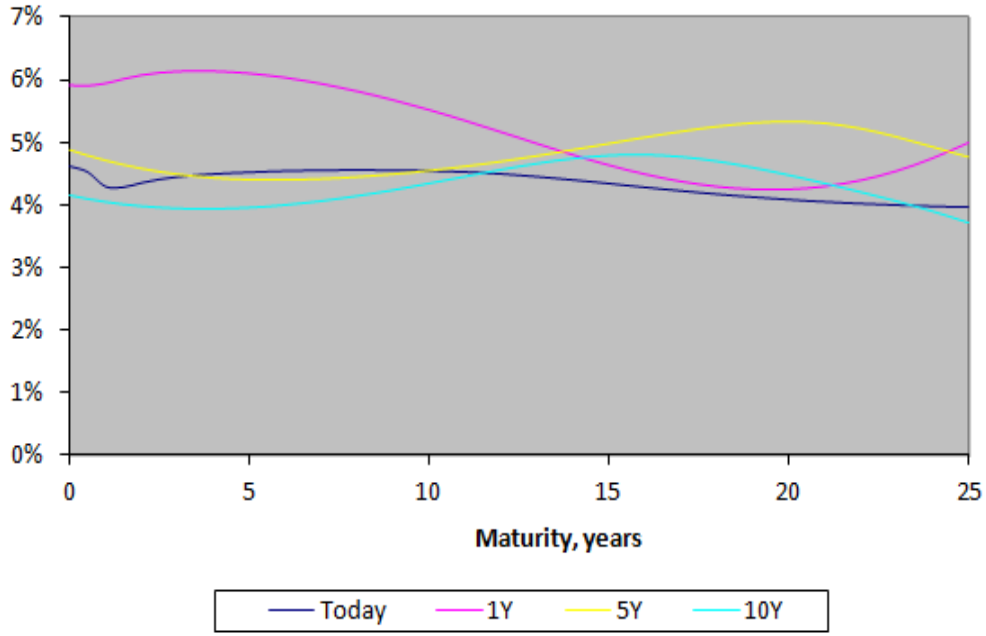
- Path generation using Euler Maruyana Numerical Scheme
- No variance reduction Scheme applied
- CURAND_RNG_PSEUDO_XORWOW pseudo random generator
- Number of simulation steps 2500, across 25 tenors
- Accuracy 0.000001

Expected Exposure Profile Simulation

Forward Rates Evolution



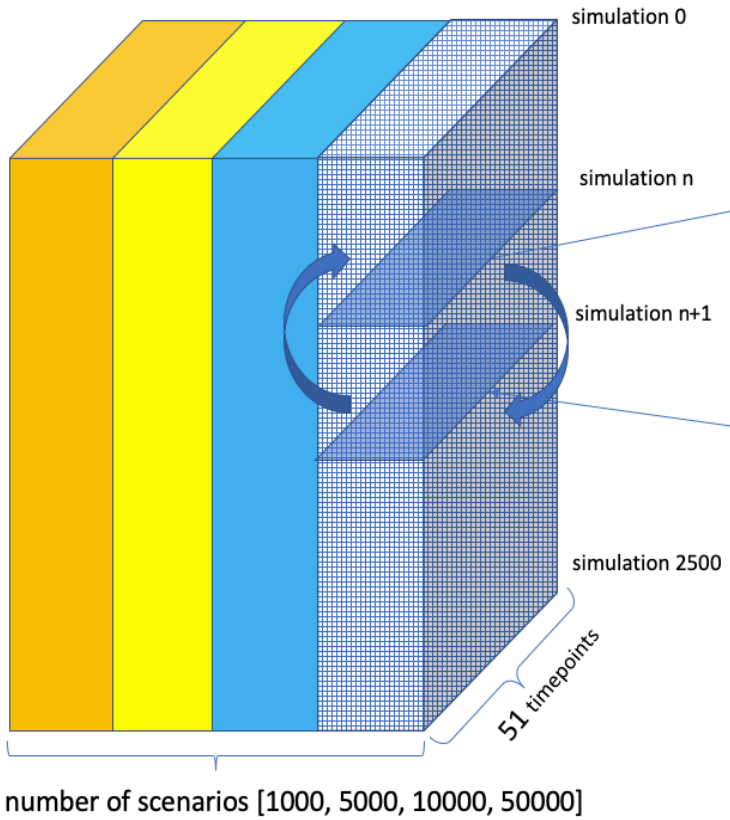
(a) Rate Evolution by Tenors

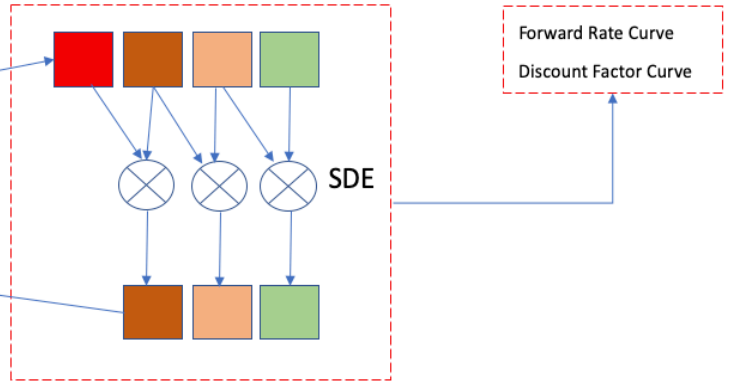


(b) Term Structure – Yield Curve

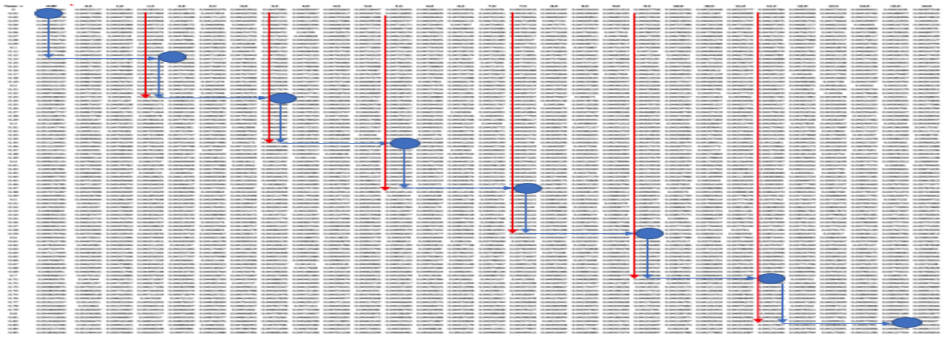
Expected Exposure Profile Simulation

Risk Factor Simulation Algorithm Complexity O(CN^2)



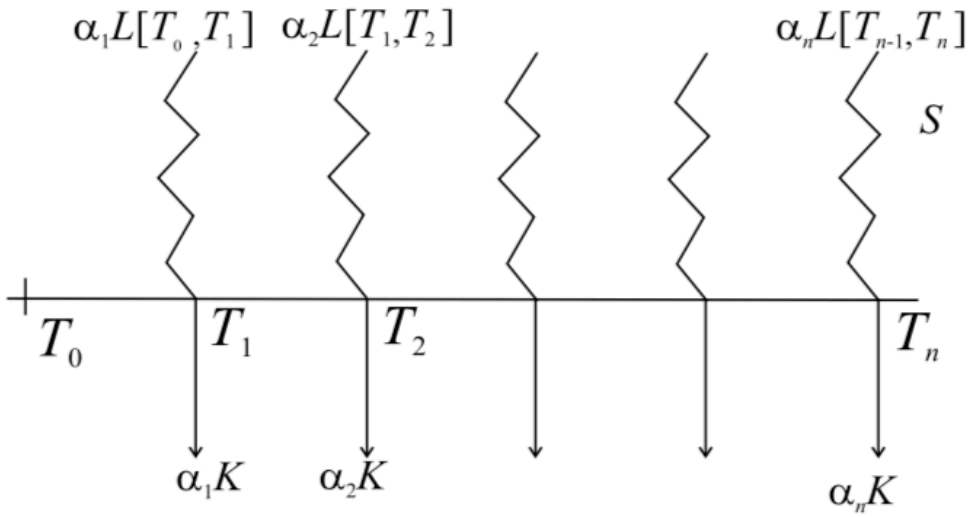


2500 rows x 51 columns



Expected Exposure Profile Simulation

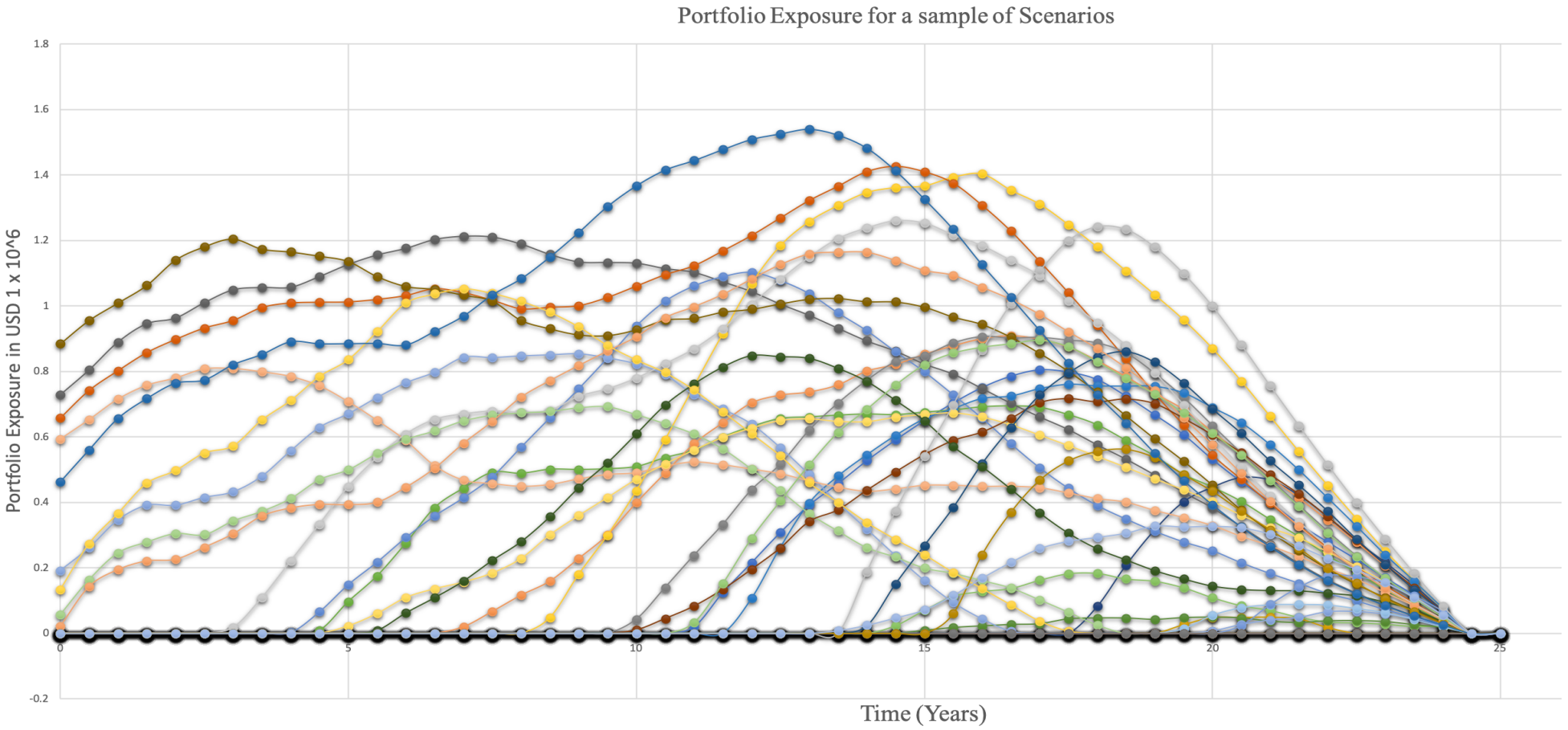
Vanilla Interest Rate Swap Net Present Value NPV(0)



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

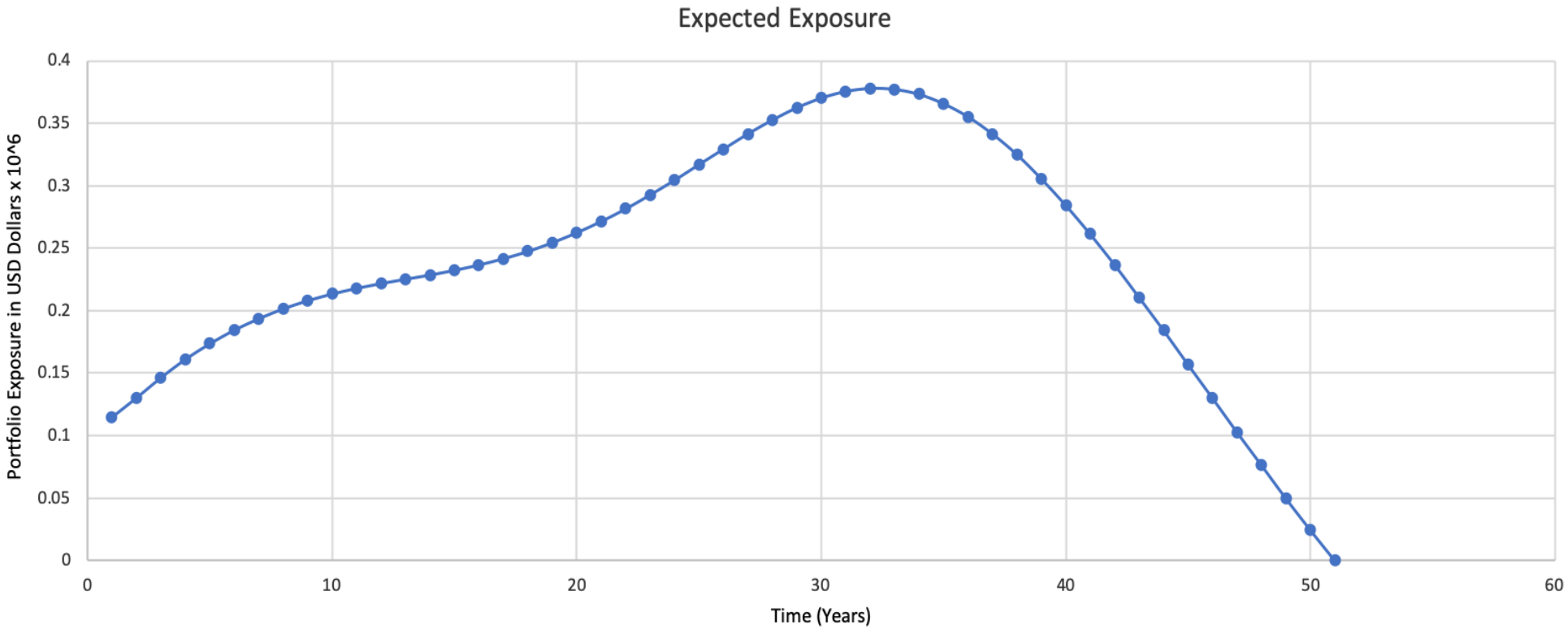
Expected Exposure Profile Simulation

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



Expected Exposure Profile Simulation

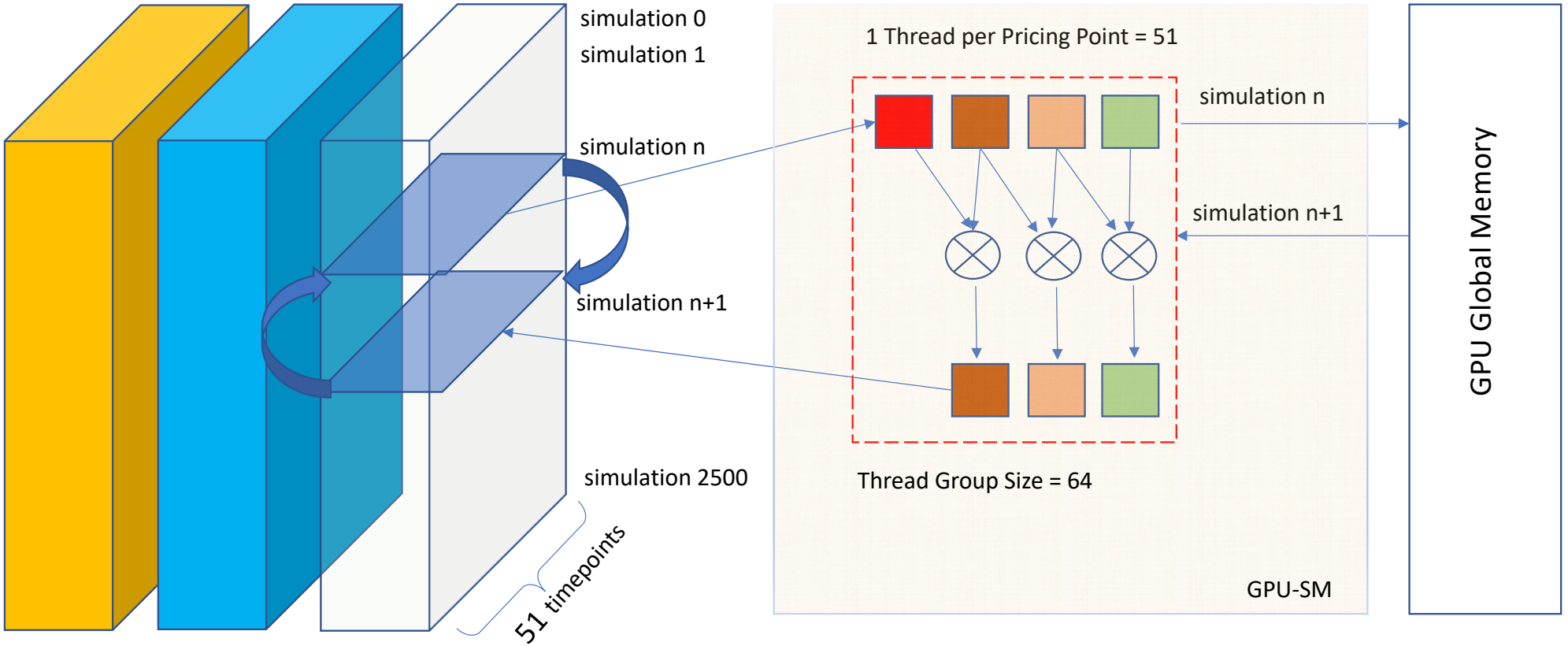
Monte Carlo Simulation of Expected Exposure Profiles



GPU Parallelization of Expected Exposure Profile

Loop Parallelization, Tiling and Stencil parallel patterns

__generate_paths_kernel<<number_scenarios, curve_size>>()



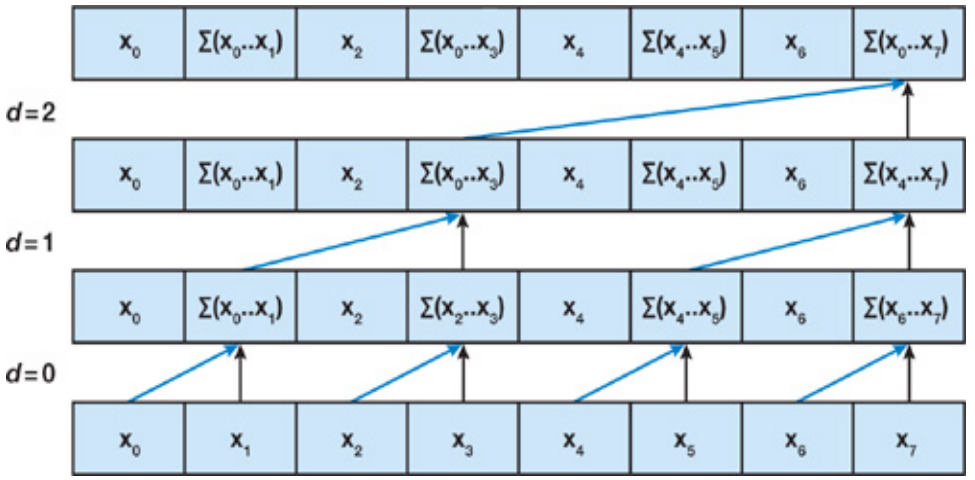
Grid Size = 50,000 / 1 Thread Group Per Scenario

https://github.com/mahsanchez/cva_hjm_cuda

GPU Parallelization of Expected Exposure Profile

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

```
forall t_k  in parallel do
    temp[k] = max(cashflow(t_k), 0);
reverse_prefix_sums( exposure, temp );
```



https://github.com/mahsanchez/cva_hjm_cuda

GPU Parallelization of Expected Exposure Profile

Parallel Column Summation Reduction using cublasDgemv (Matrix Vector Multiplication)

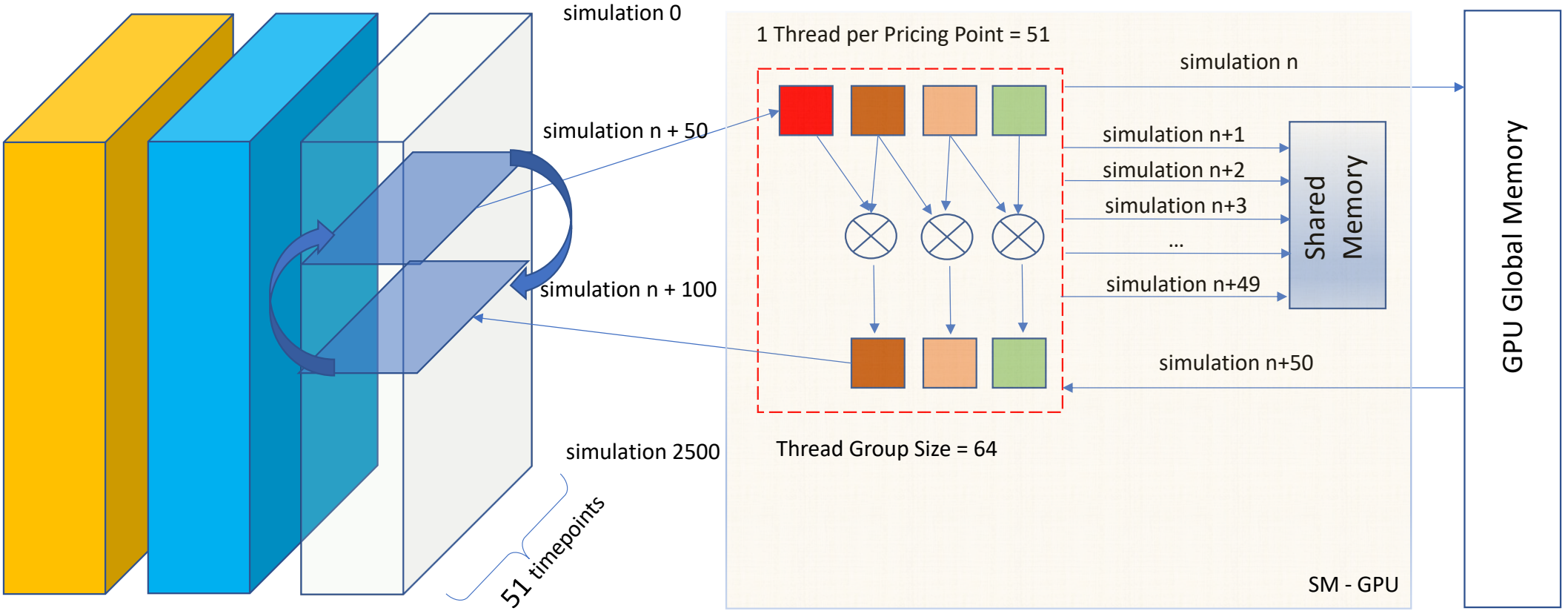
$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}^{T} (1 \quad 1 \quad 1 \quad \cdots \quad 1_n)^{T} = \left(\sum_{k=1}^{n} a_{k,1} \quad \sum_{k=1}^{n} a_{k,2} \quad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

Accelerating Reduction and Scan Using Tensor Core Units

Optimization Shared Memory

Loop Parallelization, Tiling and Stencil parallel patterns

__generate_paths_kernel4<<number_scenarios, curve_size>>()

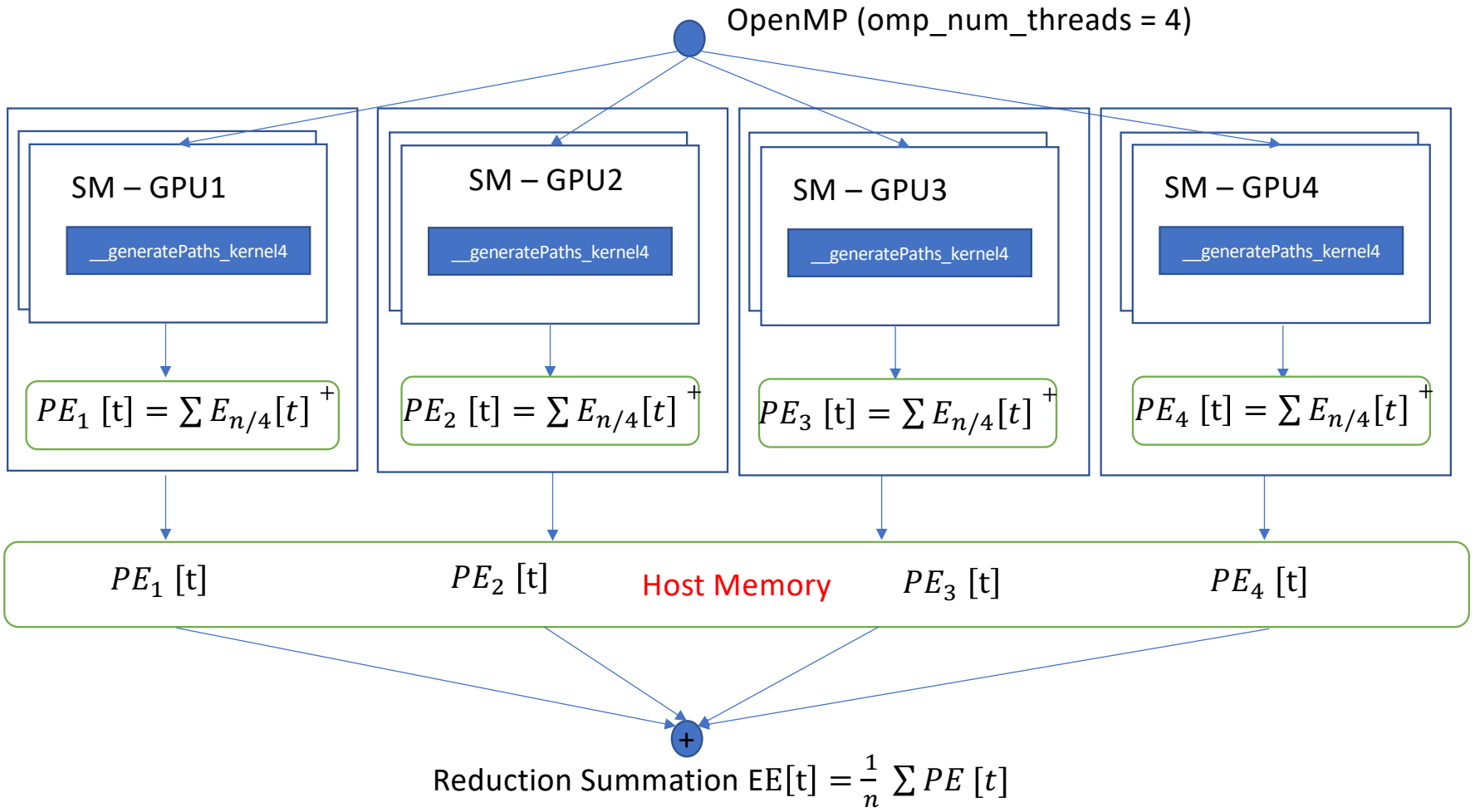


Grid Size = 50,000 / 1 Thread Group Per Scenario

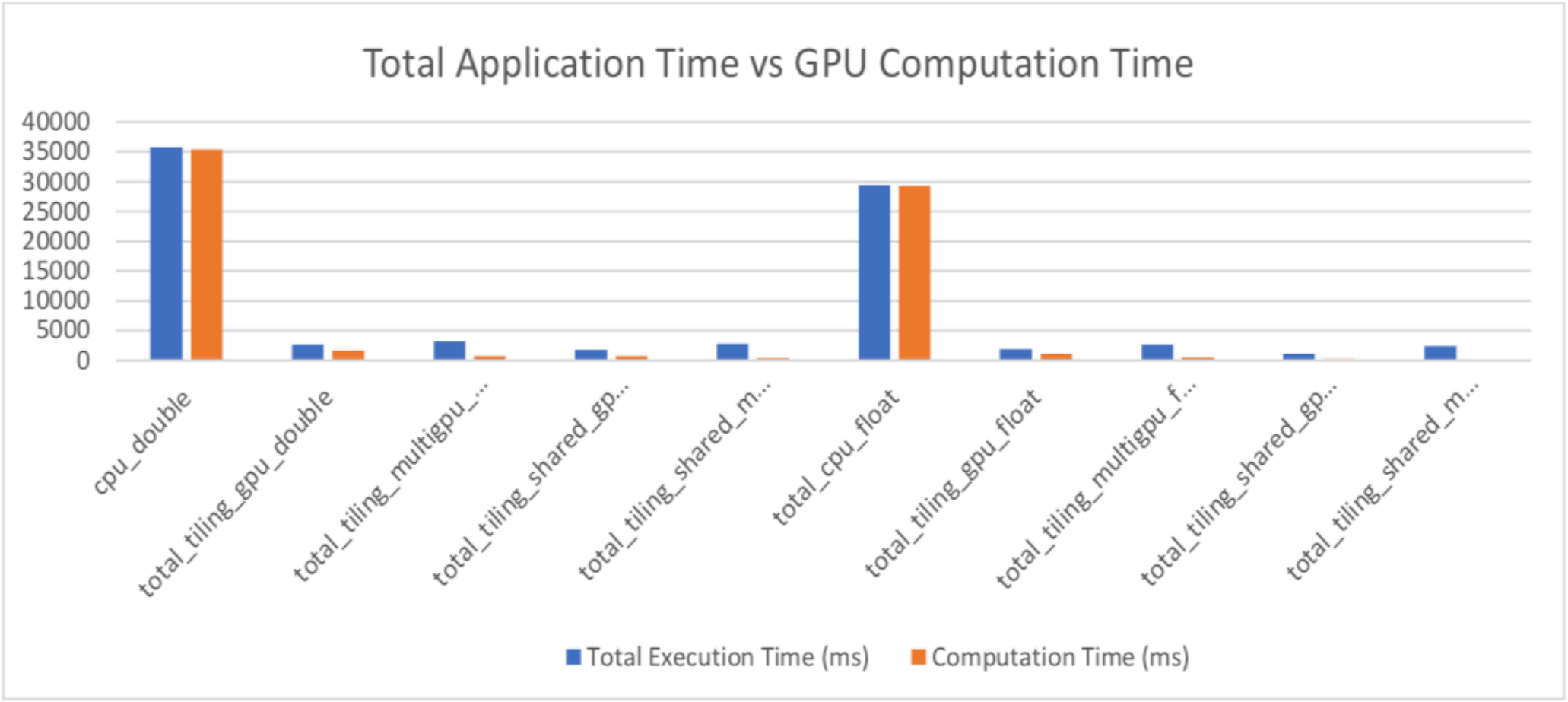
https://github.com/mahsanchez/cva_hjm_cuda

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

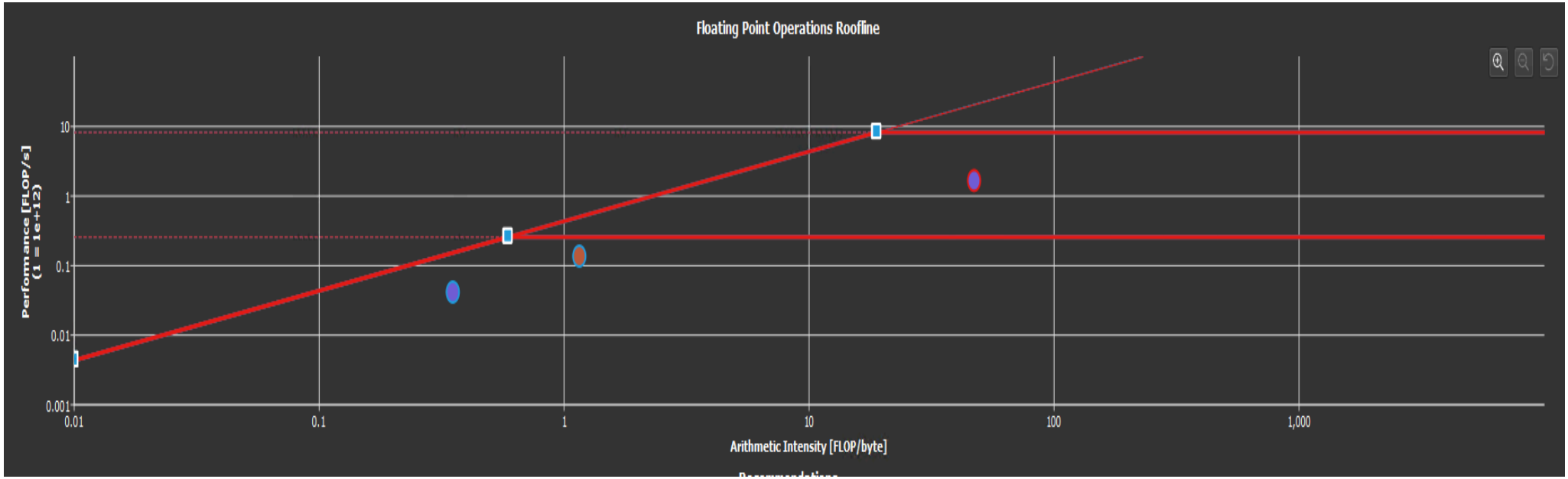
Performance Results

Floating Point Accuracy Analysis (50,000 scenarios)

Implementation	Mean Squared Error	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

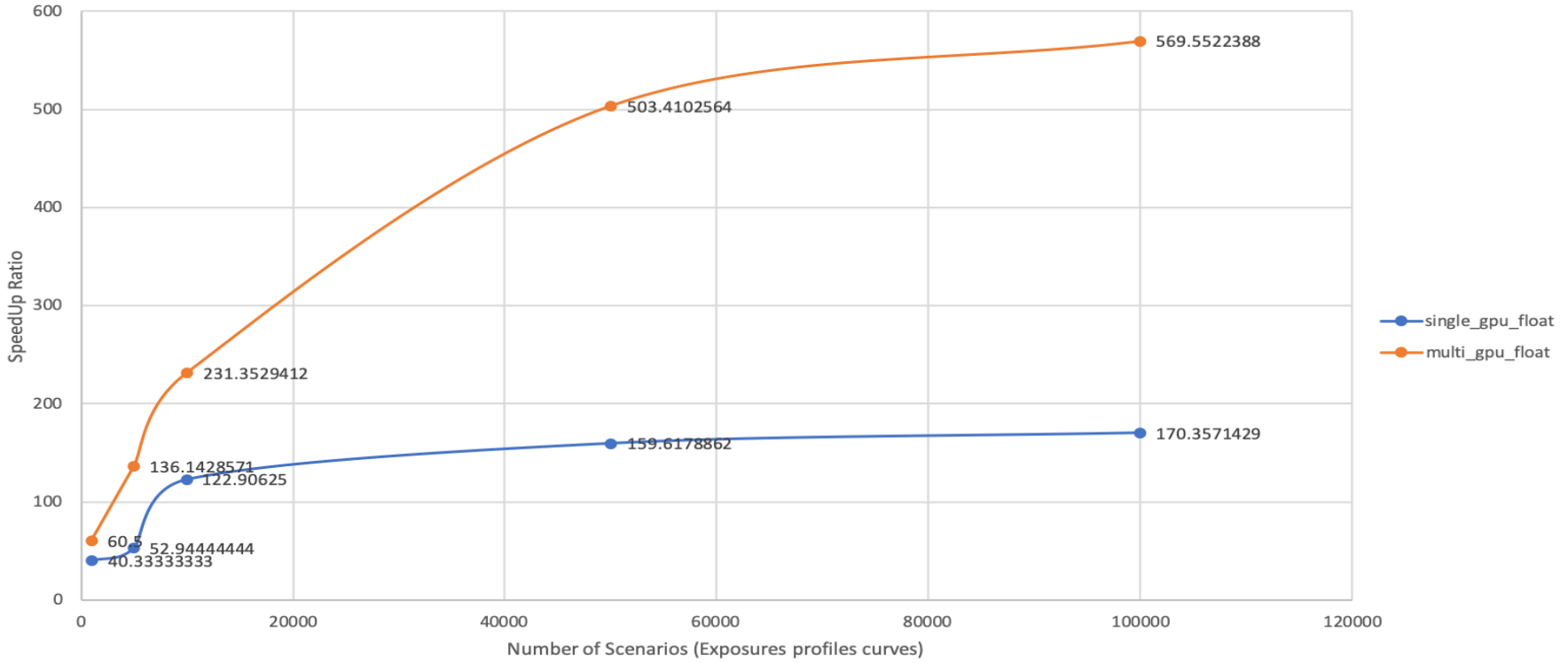
Roof Line Analysis Naïve Parallelization vs Shared Memory Optimization



__generatePaths_kernel vs __generatePaths_kernel4 (single gpu 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.
- Thanks to my participation in the PUMPS+AI 2021 Hackathon at Barcelona Super Computer Center a lot of ideas about performance measurement, floating point accuracy and further parallelization techniques enriched the realization of this study.
- 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.