Step by step implementation and optimization of simulations in quantitative finance

Lokman Abbas-Turki

UPMC, LPMA

10-12 October 2017

Plan

Plan

What make bankers change their mind?

- The CVA (Credit Valuation Adjustment) or XVA (X=C, D, F, K, M Valuation Adjustment) are tipping point applications,
- FRTB (Fundamental Review of the Trading Book) is the other important application with a deadline in 2019.
- Electronic trading and deep learning.
- Lloyds Blankfein declared about Goldman Sachs: "We are a technology firm".

HPC in banks

- From distribution to parallelization.
- From small to big nodes.
- The efficiency of GPUs becomes undeniable.
- Use the .net C, C++ and C#.

Remaining challenges and fears

- Code management.
- Possible conflicts within quant teams.
- Can we extend the results for toy models to more general models?

Plan

Local volatility from implied volatility

First array: r_g

$$dS_t = S_t r_g(t)dt + S_t \sigma_{loc}(S_t, t)dW_t, \quad S_0 = x_0.$$

- S is the stock price process where x_0 is the spot price
- W is a Brownian motion with $W_0 = 0$
- r_g is the risk-free rate, assumed piecewise constant
- $\sigma_{loc}(x, t)$ is a local volatility function: $\mathbb{R}_+^* \times \mathbb{R}_+ \to \mathbb{R}_+^*$

Dupire Equation

Given a family $C(K, T)_{K,T}$ of call prices with strike K and maturity T

$$\sigma_{loc}^2(K, T) = 2\frac{\partial C/\partial T + K r_g(T)\partial C/\partial K}{K^2(\partial^2 C/\partial K^2)}$$

From implied to local

Using the Black & Scholes implied volatility $\sigma_{imp}(x, t)$, Andersen and Brotherton-Ratcliffe (1997) showed that

$$\sigma_{loc}^2(x, t) = \frac{2\dfrac{\partial \sigma_{imp}}{\partial t} + \dfrac{\sigma_{imp}}{t} + 2x r_g(t)\dfrac{\partial \sigma_{imp}}{\partial x}}{x^2\left[\dfrac{\partial^2 \sigma_{imp}}{\partial x^2} - d_+\sqrt{t}\left(\dfrac{\partial \sigma_{imp}}{\partial x}\right)^2 + \dfrac{1}{\sigma_{imp}}\left(\dfrac{1}{x\sqrt{t}} + d_+\dfrac{\partial \sigma_{imp}}{\partial x}\right)^2\right]}$$

with $d_+ = \frac{1}{2}\sigma_{imp}\sqrt{t} + \left[\log(x_0/x) + \int_0^t r_g(u)du\right] / \left[\sigma_{imp}\sqrt{t}\right]$

SVI to simulate implied volatility

+2 arrays: T_g, K_g

For $(x,t) \in K_g \times T_g$, we assume that the observed implied volatility

$$\sigma_{imp}(x,t) = \sqrt{\frac{w(x,t)}{t}}$$

where the cumulative variance is parameterized by

$$w(x,t) = \frac{\theta_t}{2}\left[\frac{1+\rho\varphi(\theta_t)\left(\log(\frac{x}{x_0}) - \int_0^t r_g(u)du\right) + }{\sqrt{\left[\varphi(\theta_t)\left(\log(\frac{x}{x_0}) - \int_0^t r_g(u)du\right)+\rho\right]^2 + (1-\rho^2)}}\right] \quad (1)$$

$$\text{with } \varphi(\theta_t) = \frac{\eta}{\theta_t^{\gamma}(1+\theta_t)^{1-\gamma}} \text{ and } \theta_t = a^2\left(t + b(1-e^{-\lambda t})\right).$$

All parameters of (1) are discussed in Gatheral & Jacquier (2013)

Interpolation on $]0, \max(K_g)] \times]0, \max(T_g)]$

We compute some values of $\sigma_{imp}(x,t)$ on the grid $\in K_g \times T_g$. Then, we interpolate these values to obtain $\bar{\sigma}(x,t)$ defined on $]0, \max(K_g)] \times]0, \max(T_g)]$ and we assume

$$\sigma_{imp}(x,t) \approx \bar{\sigma}(x,t) \text{ when } (x,t) \in]0, \max(K_g)] \times]0, \max(T_g)]$$

Bicubic interpolation for implied volatility

Let k, q with $(x,t) \in]K_g[k], K_g[k+1]] \times]T_g[q], T_g[q+1]]$

$$\bar{\sigma}(x,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} C_g(k,q,i,j) l^i u^j$$

$$\text{where } l = \frac{t - T_g[q]}{T_g[q+1] - T_g[q]} \text{ and } u = \frac{x - K_g[k]}{K_g[k+1] - K_g[k]}$$

Fourth array: C_g

$$C_g(k,q,i,j) = C_g[k*(n_t-1)*16 + q*16 + i*4 + j]$$

with $(k,q,i,j) \in \{0,...,n_k-1\} \times \{0,...,n_t-1\} \times \{0,...,3\} \times \{0,...,3\}$ and n_k, n_t are the size of K_g and T_g

Approximated local volatility

$$\sigma^2_{loc}(x,t) \approx \min(\max(\widetilde{\sigma}^2(x,t), 0.0001), 0.5)$$

$$\text{where } \widetilde{\sigma}^2(x,t) = \frac{2\frac{\partial\bar{\sigma}}{\partial t} + \frac{\bar{\sigma}}{t} + 2xr_g(t)\frac{\partial\bar{\sigma}}{\partial x}}{x^2\left[\frac{\partial^2\bar{\sigma}}{\partial x^2} - d_+\sqrt{t}\left(\frac{\partial\bar{\sigma}}{\partial x}\right)^2 + \frac{1}{\bar{\sigma}}\left(\frac{1}{x\sqrt{t}} + d_+\frac{\partial\bar{\sigma}}{\partial x}\right)^2\right]}$$

$$\text{with } d_+ = \tfrac{1}{2}\bar{\sigma}\sqrt{t} + \left[\log(x_0/x) + \int_0^t r_g(u)du\right] / \left[\bar{\sigma}\sqrt{t}\right]$$

Plan

Pricing bullet option with Monte Carlo (MC)

Price for $t \in [0, T)$

$$F_t = e^{-\int_t^T r_g(u)du} E\left((S_T - K)_+ 1_{\{I_T \in [P_1, P_2]\}} | \mathfrak{F}_t\right) \text{ with } I_t = \sum_{T_i \leq t} 1_{\{S_{T_i} < B\}}$$

- K, T are respectively the contract's strike and maturity
- $0 < T_1 < ... < T_M < T$ is a predetermined schedule
- The barrier B should be crossed I_T times with $\{P_1, ..., P_2\} \subset \{0, ..., M\}$

(S_t, I_t) is Markov

$$F(t, x, j) = e^{-\int_t^T r_g(u)du} E(X | S_t = x, I_t = j), \; X = (S_T - K)_+ 1_{\{I_T \in [P_1, P_2]\}}$$

MC procedure

Simulate $F(0, x_0, 0) = E(X)$ using a family $\{X_i\}_{i \leq n}$ of i.i.d $\sim X$

- **Strong law of large numbers**:

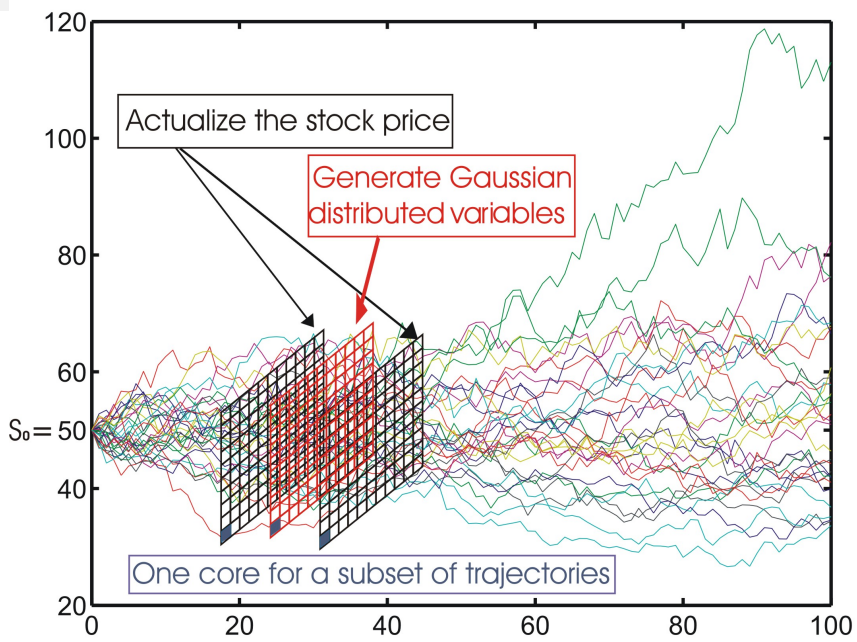
$$P\left(\lim_{n \to +\infty} \frac{X_1 + X_2 + ... + X_n}{n} = E(X)\right) = 1$$

- **Central limit theorem**: Denoting $\epsilon_n = E(X) - \dfrac{X_1 + X_2 + ... + X_n}{n}$

$$\frac{\sqrt{n}}{\sigma} \epsilon_n \to G \sim \mathcal{N}(0, 1), \; \sigma \text{ is the standard deviation of } X$$

- There is a 95% chance of having: $\epsilon_n \leq 1.96 \dfrac{\sigma}{\sqrt{n}}$

Discritization set $0 = t_0 < t_1 < ... < t_{N_t} = T$ finer than $0 < T_1 < ... < T_M < T$ with $\delta t = \sqrt{t_{k+1} - t_k}$



Iterating for path-dependant contract ($x_0 = 50$)

For each t_k, $k = 0, ..., N_t - 1$:

1. Random number generation (RNG) of independent Normal variables G_i
2. Stock price actualization $S^i_{t_{k+1}} = S^i_{t_k}(1 + r_g(t_k)\delta t \delta t + \sigma_{loc}(S^i_{t_k}, t_k)\delta t G_i)$
3. If $t_{k+1} = T_l$ with $l \in \{1, ..., M\}$, $I^i_{T_l} = I^i_{T_{l-1}} + (S^i_{T_l} < B)$

At t_{N_t} Compute the payoff X^i then average

P. L'Ecuyer CMRG on GPU to generate uniformly distributed random variables

General Form of linear RNGs

Without loss of generality:

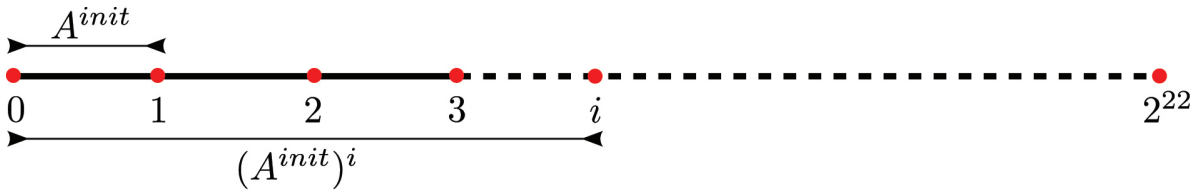
$$X_n = (AX_{n-1} + C)\ mod(m) = (A\ :\ C)\begin{pmatrix} X_{n-1} \\ \dots \\ 1 \end{pmatrix}\ mod(m) \qquad (3)$$

Parallel-RNG from Period Splitting of* One RNG

Pierre L'Ecuyer proposed a very efficient RNG (1996) which is a CMRG on 32 bits: Combination of two Multiple Recursive Generator (MRG) with $lag = 3$ for each MRG.

* Very long period $\sim 2^{185}$

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + a_3 x_{n-3}) mod(m)$$



Pre-computations We launch as many parallel RNGs as the number of paths

Use We prefer local variables to store the RNG's state vector

Some OpenACC pragmas: #pragma acc clause

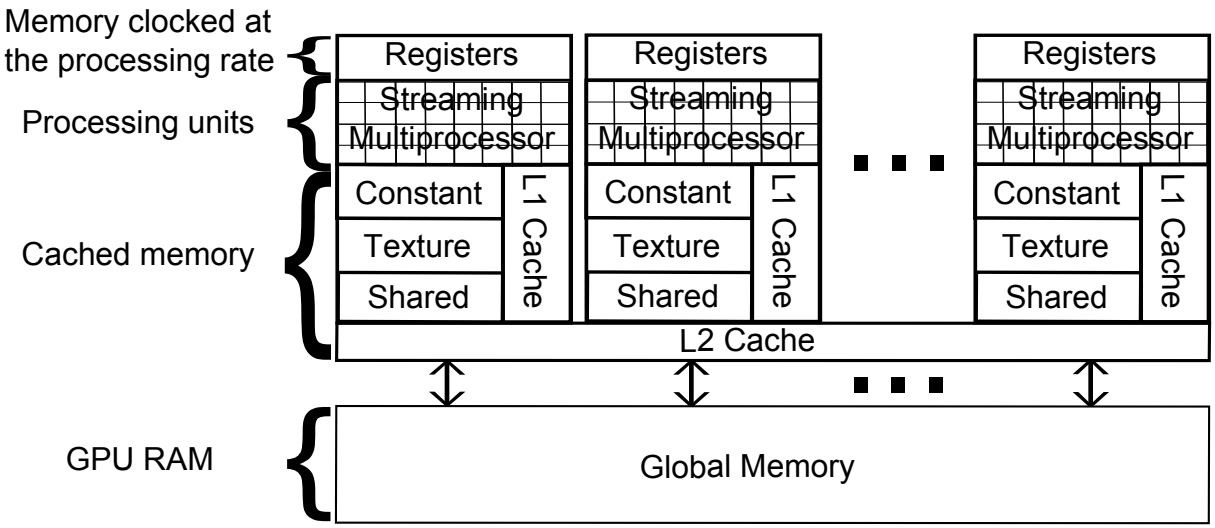
global variables — **clause = declare device_resident**, applied to r_g, K_g, T_g and C_g. These arrays are known by all GPU functions

copying to GPU — **clause = enter data copyin** copies the values to the GPU

embarrassingly parallel — **clause = parallel loop present (+reduction(operation,arguments))** making parallel the execution of the loop without data movement. **reduction(operation,arguments)** reduces all the private copies of **arguments** into one final result using **operation**

Plan

GPU architecture



Hardware software equivalence

- Streaming processor: Executes threads
- Streaming multiprocessor: Executes blocks

Built-in variables Known within functions excuted on GPU: threadIdx.x, blockIdx.x, blockDim.x, gridDim.x

Global memory GPU RAM, allocated thanks to **cudaMalloc**. Transfer values from GPU/CPU to CPU/GPU thanks to **cudaMemcpy**

Constant memory Read only, declared globally. Data transfer with **cudaMemcpyToSymbol**

Shared Declared in the kernel

Function declaration and calling

Standard C functions

The same as for C or C++ programming

Device functions

- Called by the GPU and executed on the GPU
- Declared as

```
__device__ void myDivFun (...)  { ...; }
__device__ float myDivFun (...)  { ...; }
```

- Called simply by `myDivFun(...)` but only within other device functions or kernels

Kernel functions

- Called by the CPU and executed on the GPU
- Declared as `__global__ void myKernel (...)  { ...; }`
- Called standardly by
  `myKernel<<<numBlocks, threadsPerBlock>>>(...);`
  where
  - `numBlocks` should take into account the number of multiprocessors
  - `threadsPerBlock` should be equal to 128, 256, 512 or 1024
- Dynamic parallelism: kernels can be called within kernels by the GPU and executed on the GPU

Adaptation steps

Extension Change it from MC.cpp and rng.cpp to MC.cu and rng.cu

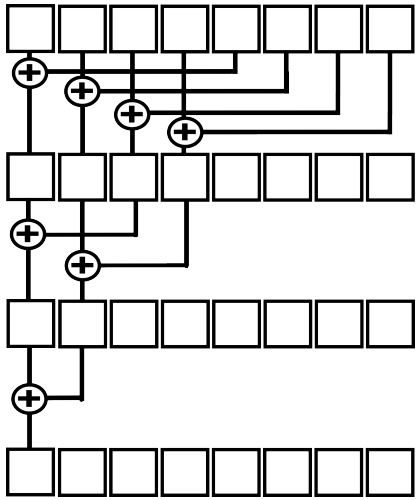
__constant__ Declare T_g, K_g, r_g and C_g using constant memory

Copying Copy the values T_g, K_g, r_g, C_g and *CMRG* on the GPU

__device__ Except main, MC, VarMalloc, FreeVar and parameters, define all other functions in MC.cu using __device__

Parallel Declare MC function as a kernel using __global__ + Replace the Monte Carlo loop using threadIdx.x, blockDim.x, blockIdx.x in MC kernel

Reduction Perform the reduction using __shared__ memory in kernel MC and atomicAdd function



GPU timer

```
float Tim;
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start,0);

/*****************************************************

To compute the execution time of this part of the code

*****************************************************/

cudaEventRecord(stop,0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&Tim, start, stop);
cudaEventDestroy(start);
cudaEventDestroy(stop);
```

Plan

PDE, $u(t,x,j)$ with respect to x, Black & Scholes PDE

The price

$F(t,x,j) = e^{-\int_t^T r_g(u)du} E(X|S_t = x, I_t = j),\ X = (S_T - K)_+ 1_{\{I_T \in [P_1, P_2]\}}$

For $u(t,x,j) = e^{\int_t^T r_g(u)du} F(t, e^x, j)$, we equivalently solve u PDE given by

$$\frac{1}{2}\sigma_{loc}^2(x,t)\frac{\partial^2 u}{\partial x^2}(t,x,j) + \mu(x,t)\frac{\partial u}{\partial x}(t,x,j) = -\frac{\partial u}{\partial t}(t,x,j) \tag{5}$$

$\mu(x,t) = r_g(t) - \dfrac{\sigma_{loc}^2(x,t)}{2}$, $(x,t,j) \in]0, \max(K_g)] \times]T_k, T_{k+1}[\times [0, P_2]$
with $T_0 = 0$ and $T_{M+1} = T$

Final and boundary conditions

$u(T,x,j) = \max(e^x - K, 0)$ for any (x,j) and heuristically
$u(t, \log[\min(Kg)], j) = \texttt{pmin} = 0$ and
$u(t, \log[\max(Kg)], j) = \texttt{pmax} = \max(Kg) - K$

Crank-Nicolson scheme

Denoting $u_{k,i,j} = u(t_k, x_i, j)$, $\sigma = \sigma_{loc}(x_i, t_k)$ and $\mu = \mu(x_i, t_k)$

$$q_u u_{k,i+1} + q_m u_{k,i} + q_d u_{k,i-1} = p_u u_{k+1,i+1} + p_m u_{k+1,i} + p_d u_{k+1,i-1}$$

$$q_u = -\frac{\sigma^2 \Delta t}{4\Delta x^2} - \frac{\mu \Delta t}{4\Delta x}, \quad q_m = 1 + \frac{\sigma^2 \Delta t}{2\Delta x^2}, \quad q_d = -\frac{\sigma^2 \Delta t}{4\Delta x^2} + \frac{\mu \Delta t}{4\Delta x}$$

$$p_u = \frac{\sigma^2 \Delta t}{4\Delta x^2} + \frac{\mu \Delta t}{4\Delta x}, \quad p_m = 1 - \frac{\sigma^2 \Delta t}{2\Delta x^2}, \quad p_d = \frac{\sigma^2 \Delta t}{4\Delta x^2} - \frac{\mu \Delta t}{4\Delta x}$$

PDE, $u(t,x,j)$ with respect to j, barrier condition

$$u_t(x,j) = \mathbb{E}\left[(S_T - K)_+ 1_{\left\{\sum_{i=1}^{M} 1_{\{S_{T_i}<B\}} \in [P_1,P_2]\right\}} \middle| S_t = x, \sum_{T_i \le t} 1_{\{S_{T_i}<B\}} = j\right]$$

$t \in [T_M, T[$

$$u_t(x,j) = \mathbb{E}[(S_T - K)_+ | S_t = x]$$

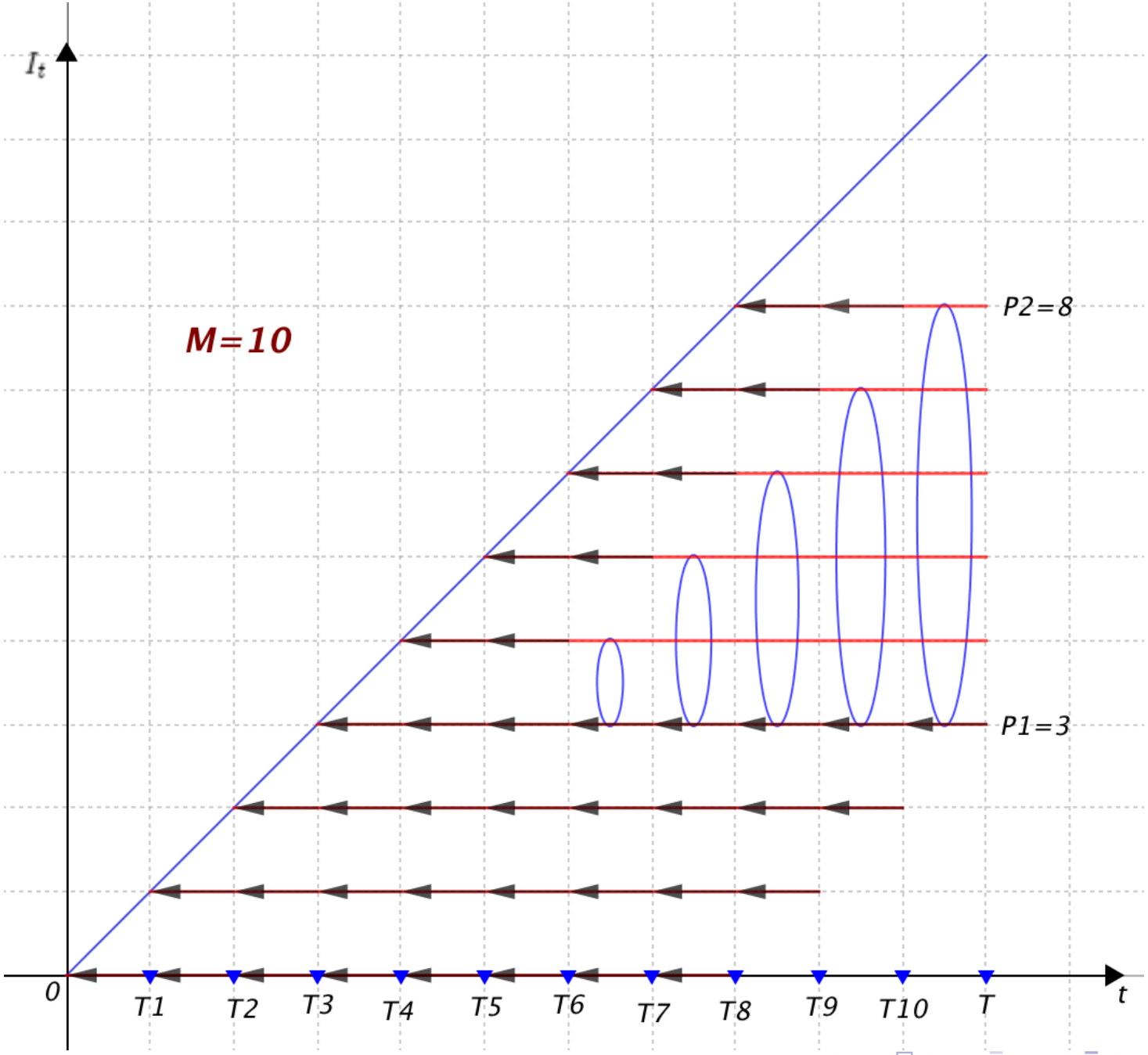
$t \in [T_{M-1}, T_M]$

$$\begin{aligned} u_t(x,j) &= \mathbb{E}[(S_T - K)_+ 1_{\{S_{T_M} \ge B\}} \mid S_t = x] 1_{\{j=P_2\}} \\ &+ \mathbb{E}[(S_T - K)_+ 1_{\{S_{T_M} < B\}} \mid S_t = x] 1_{\{j=P_1-1\}} \\ &+ \mathbb{E}[(S_T - K)_+ | S_t = x] 1_{\{j \in [P_1, P_2-1]\}} \end{aligned}$$

$[T_{M-k-1}, T_{M-k}]$
$k = M-1, \dots, 1$
$(T_0 = 0)$

$$\begin{aligned} u_t(x,j) &= \mathbb{E}[u_{T_{M-k}}(S_{T_{M-k}}, P_2) 1_{\{S_{T_{M-k}} \ge B\}} \mid S_t = x] 1_{\{j=P_2\}} \\ &+ \mathbb{E}[u_{T_{M-k}}(S_{T_{M-k}}, p_k^1) 1_{\{S_{T_{M-k}} < B\}} \mid S_t = x] 1_{\{j=p_k^1-1\}} \\ &+ \mathbb{E}\left[\begin{matrix} u_{T_{M-k}}(S_{T_{M-k}}, j) 1_{\{S_{T_{M-k}} \ge B\}} \\ + u_{T_{M-k}}(S_{T_{M-k}}, j+1) 1_{\{S_{T_{M-k}} < B\}} \end{matrix} \middle| S_t = x\right] 1_{\{j \in [p_k^1, P_2-1]\}} \end{aligned}$$

with $p_k^1 = \max(P_1 - k, 0)$

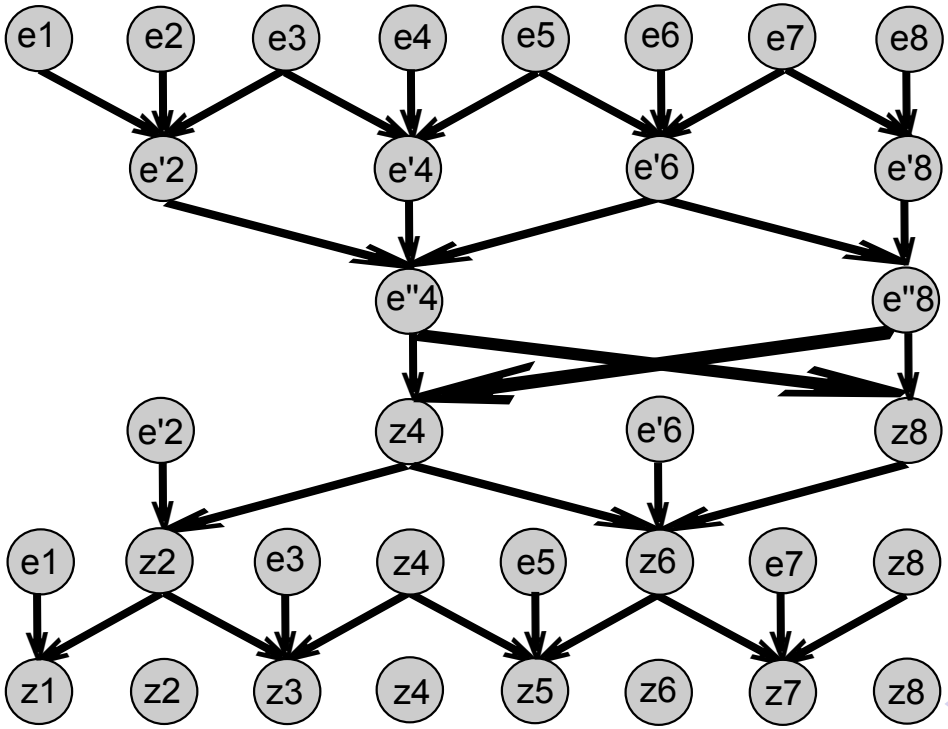

I_t
M=10
P2=8
P1=3
0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T
t

Plan

Solve tridiagonal system for the implicit part

$$T = \begin{pmatrix} d_1 & c_1 & & & & \\ a_2 & d_2 & c_2 & & 0 & \\ & a_3 & d_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & a_n & d_n \end{pmatrix}.$$

Cyclic Reduction



Step 1: Forward reduction to a 4-unknown system involving z2, z4, z6 and z8

Step 2: Forward reduction to a 2-unknown system involving z4 and z8

Step 3: Solve 2-unknown system

Step 4: Backward substitution to solve the rest 2 unknowns

Step 5: Backward substitution to solve the rest 4 unknowns

$$
\begin{pmatrix}
d_1 & c_1 & & & & & \\
c_1 & d_2 & c_2 & & & & \\
 & c_2 & d_3 & c_3 & & & \\
 & & c_3 & d_4 & c_4 & & \\
 & & & c_4 & d_5 & c_5 & \\
 & & & & c_5 & d_6 & c_6 \\
 & & & & & c_6 & d_7
\end{pmatrix}
\xrightarrow{(R)}
\begin{pmatrix}
d'_1 & 0 & c'_2 & & & & \\
0 & d'_2 & 0 & c'_3 & & & \\
c'_2 & 0 & d'_3 & 0 & c'_4 & & \\
 & c'_3 & 0 & d'_4 & 0 & c'_5 & \\
 & & c'_4 & 0 & d'_5 & 0 & c'_6 \\
 & & & c'_5 & 0 & d'_6 & 0 \\
 & & & & c'_6 & 0 & d'_7
\end{pmatrix}
$$

$$
\xrightarrow{(P)}
\begin{pmatrix}
d'_1 & c'_2 & & & & & \\
c'_2 & d'_3 & c'_4 & & & & \\
 & c'_4 & d'_5 & c'_6 & & & \\
 & & c'_6 & d'_7 & 0 & & \\
 & & & 0 & d'_2 & c'_3 & \\
 & & & & c'_3 & d'_4 & c'_5 \\
 & & & & & c'_5 & d'_6
\end{pmatrix}
$$

$$
\xrightarrow{(R)}
\begin{pmatrix}
d''_1 & 0 & c''_2 & & & & \\
0 & d''_3 & 0 & c''_4 & & & \\
c''_2 & 0 & d''_5 & 0 & & & \\
 & c''_4 & 0 & d''_7 & 0 & & \\
 & & & 0 & d''_2 & 0 & c''_3 \\
 & & & & 0 & d''_4 & 0 \\
 & & & & c''_3 & 0 & d''_6
\end{pmatrix}
$$

$$
\xrightarrow{(P)}
\begin{pmatrix}
d''_1 & c''_2 & & & & & \\
c''_2 & d''_5 & 0 & & & & \\
 & 0 & d''_3 & c''_4 & & & \\
 & & c''_4 & d'_7 & 0 & & \\
 & & & 0 & d''_2 & c''_3 & \\
 & & & & c''_3 & d''_6 & 0 \\
 & & & & & 0 & d''_4
\end{pmatrix}
$$

Adaptation steps

Extension	Change it from PDE.cpp to PDE.cu
__constant__	Declare T_g, K_g, r_g and C_g using constant memory
Copying	Copy the values T_g, K_g, r_g, C_g on the GPU
__device__	Except for main, PDE_Diff, Out2In, VarMalloc, FreeVar and parameters, define all other functions in PDE.cu using __device__
Registers	Compare the CPU version of PCR to the GPU version given to you. Remark the large use of shared memory and registers.
Parallel	Declare PDE_Diff and Out2In functions as kernels using __global__ + Replace loops related to S_t by threadIdx.x and loops related to I_t by blockIdx.x in PDE_Diff and Out2In kernels as well as in Exp device function.

GPU timer

```
float Tim;
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start,0);

/*****************************************************

To compute the execution time of this part of the code

*****************************************************/

cudaEventRecord(stop,0);
cudaEventSynchronize(stop);
cudaEventElapsedTime(&Tim, start, stop);
cudaEventDestroy(start);
cudaEventDestroy(stop);
```

References

- L. A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier. "Pricing derivatives on graphics processing units using Monte Carlo simulation". Concurrency and Computation: Practice and Experience, vol. 26(9), pp. 1679–1697, 2014.
- L. A. Abbas-Turki and Stef Graillat, "Resolving small random symmetric linear systems on graphics processing units", The Journal of Supercomputing 73(4), pp. 1360–1386, 2017.
- L. B. G. Andersen and R. Brotherton-Ratcliffe, "The equity option volatility smile: an implicit finite difference approach", Journal of Computational Finance, vol. 1(2), pp. 5–37, 1997.
- J. Gatheral and A. Jacquier, "Arbitrage-free SVI volatility surfaces", Quantitative Finance, vol. 14(1), pp. 59–71, 2013.
- P. L'Ecuyer, "Combined multiple recursive random number generators". Operations Research, vol. 44(5), pp. 816–822, 1996.