Parallel Implementation of the Quantization Tree Algorithm for CUDA

Benedikt Wilbertz

joint work with G. Pagès and A.L. Bronstein

Université Paris 6, LPMA

Groupe de Travail GPGPU, 22th October 2009

Introduction

Problem Description

Swing options - A common contract in energy markets

The right to buy every day a certain quantity of gas/electricity for a given price, where the bought quantity has to respect certain daily and global constraints.

The fair premium of such an contract leads to a stochastic control problem (SCP)

\[
\text{esssup} \left\{ \mathbb{E} \left( \sum_{k=0}^{n-1} q_k V_k | \mathcal{F}_0 \right) \middle| q_k : (\Omega, \mathcal{F}_k) \to [0,1], \bar{q}_n \in [Q_{\min}, Q_{\max}] \right\}
\]

with \( \bar{q}_k := \sum_{l=0}^{k-1} q_l \) and \((V_k)\) some \((\mathcal{F}_k)\) adapted r.v. on a probability space \((\Omega, \mathcal{A}, \mathbb{P})\).

We assume that \( V_k = v_k(X_k), \) where \((X_k)\) is an \((\mathcal{F}_k)\) adapted Markov-Feller chain.
Important example: Gaussian 2-factor Model

\((X_k)\) is a 2-dim Ornstein-Uhlenbeck process

\[
X_k = \left( \int_0^{k\Delta t} e^{-\alpha_1(k\Delta t-s)} dW_1^s, \int_0^{k\Delta t} e^{-\alpha_2(k\Delta t-s)} dW_2^s \right)
\]

and the payoff-function \(v_k\) is given by

\[
v_k(z_1, z_2) = (x_0 \exp(\sigma_1 z_1 + \sigma_2 z_2 - \frac{1}{2} \Delta_k^2 \Delta t_k) - K).
\]

It follows that \((X_k)\) can be written as first-order auto regressive (AR(1)) process

\[
X_{k+1} = A_k X_k + T_k \epsilon_k
\]

with an i.i.d. standard normal sequence \((\epsilon_k)\).

\(\implies (X_k)\) has centered Gaussian marginals.

Backward Dynamic Programming Principle

In fact, one shows that we can solve this stochastic control problem by the Backward Dynamic Programming Principle (BDP), i.e. we set

\[
P_n^0 \equiv 0
\]

\[
P_k^n(Q^k) = \sup \left\{ xv_k(X_k) + \mathbb{E}(P_{k+1}^n(\chi^{n-k-1}(Q^k, x))|X_k), x \in I^{n-k-1}_Q \right\}
\]

with admissible set \(I^M_Q := [(Q_{\min}^k - M)^+ \land 1, Q_{\max}^k \land 1]\) and

\[
\chi^M(Q^k, x) := ((Q_{\min}^k - x)^+, (Q_{\max}^k - x) \land M).
\]

Then \(P_0^n(Q_{\min}, Q_{\max})\) is a solution to our SCP.
Furthermore, it is established in [Bardou/Bouthemy/Pagès ’07], that there exists a Bang-Bang control for the above problem. That is, there is an optimal control, which only takes the extreme values of the admissible set $I_{Q_k}^{n-k-1}$.

This leads to

$$P_n^n \equiv 0$$

$$P_n^n(Q_k) = \max \left\{ x v_k(X_k) + \mathbb{E}(P_{n+1}(\chi^{n-k-1}(Q_k^k, x))|X_k), x \in \{0, 1\} \cap I_{Q_k}^{n-k-1} \right\}$$

Discretization of the state space

To finally solve the BDP problem numerically, we have to discretize the state space of the random variables $(X_k)$. Hence, we assume that $\hat{X}_k$ is discrete r.v. which takes only $N_k$ values and that $\mathbb{E}\|X_k - \hat{X}_k\|^2$ is small. The corresponding BDP problem then writes

$$\hat{P}_n^n \equiv 0$$

$$\hat{P}_k^n(Q_k) = \max \left\{ x v_k(\hat{X}_k) + \mathbb{E}(\hat{P}_{n+1}(\chi^{n-k-1}(Q_k^k, x))|\hat{X}_k), x \in \{0, 1\} \cap I_{Q_k}^{n-k-1} \right\}$$

Under some moderate assumptions on $v_k$ and $(X_k)$ it holds

$$|P_0^n(Q) - \hat{P}_0^n(Q)| \leq C \sum_{k=0}^{n-1} (\mathbb{E}\|X_k - \hat{X}_k\|^2)^{1/2}$$

for any reasonable $Q$ (see [Bardou/Bouthemy/Pagès ’07]).
Approximation of conditional expectations

Assume that we want to compute a conditional expectation

\[ \mathbb{E}(f(\hat{X}_{k+1})|\hat{X}_k), \]

where \( \hat{X}_k \) takes its values in the finite grid \( \Gamma^k = (x_1^k, \ldots, x_{N^k}^k) \) in \( \mathbb{R}^d \). Since \( \hat{X}_k \) and \( \hat{X}_{k+1} \) are discrete r.v.s, this calculates as

\[
\mathbb{E}(f(\hat{X}_{k+1})|\hat{X}_k = x_i^k) = \sum_{j=1}^{N_{k+1}} f(x_{j}^{k+1}) \pi_{ij}^k
\]

where

\[
\pi_{ij}^k = \mathbb{P}(\hat{X}_{k+1} = x_{j}^{k+1}|\hat{X}_k = x_i^k), \quad k = 0, \ldots, n - 2,
\]

denotes the transition probability from the state \( x_i^k \) to \( x_{j}^{k+1} \).

Here the computation of the \( \pi_{ij}^k \)'s is the most time-consuming part of the calculations.

Optimal Quantization

In view of minimizing \( \mathbb{E}\|X - \hat{X}\|^2 \) for a general r.v. \( X \in L^2_{\mathbb{R}^d}(\mathbb{P}) \), one would choose the discretized r.v. \( \hat{X} \) as solution to

\[
\inf \left\{ \mathbb{E}\|X - \hat{X}\|^2 : \hat{X} \text{ r.v. with } \text{card}\{\hat{X}(\Omega)\} \leq N \right\}
\]

at some level \( N \in \mathbb{N} \) (see [Graf/Luschgy ’00] for further details).

One easily shows that this is equivalent to solving

\[
\inf \left\{ \mathbb{E} \min_{x \in \Gamma} \|X - x\|^2 : \Gamma \subset \mathbb{R}^d, \text{card}\{\Gamma\} \leq N \right\}.
\]

Such grids are precomputed and can be downloaded at

www.quantization.math-fi.com
How to construct an optimal $\hat{X}$ from an optimal grid $\Gamma = (x_1, \ldots, x_n)$?

Let $(C_{x_i}(\Gamma))_{1 \leq i \leq N}$ denote a Borel-partition of $\mathbb{R}^d$ satisfying

$$C_{x_i}(\Gamma) \subset \{ y \in \mathbb{R}^d : \| y - x_i \| \leq \min_{1 \leq j \leq N} \| y - x_j \| \}.$$ 

Then we set

$$\hat{X}^{\Gamma} := \sum_{i=1}^{N} x_i \cdot 1_{C_{x_i}(\Gamma)}(X).$$

This leads to

$$\pi_{k}^{ij} = \frac{\mathbb{P}(X_{k+1} \in C_j(x^{k+1}), X_k \in C_i(x^k))}{\mathbb{P}(X_k \in C_i(x^k))},$$

which we want to compute by MC-Simulation on a GPU for the rest of this talk.
### Naive Diffusion approach

For $m = 1, \ldots, M$ do

1. Initialization
   
   $x \leftarrow x_0$, $i \leftarrow 0$, $p_i^0 \leftarrow 1$

2. For $k = 1, \ldots, n - 1$ do
   
   - Simulate $\epsilon_k$
     
     $x \leftarrow A_k x + T_k \epsilon_k$
   
   - Find NN-Index $j$ of $x$ in $\Gamma_k$
   
   - Set
     
     $p_{k}^{ij} \leftarrow 1$
     
     $p_{k+1}^j \leftarrow 1$
   
3. $i \leftarrow j$

4. End For

5. End For

Set

$\pi_{k}^{ij} \leftarrow \frac{p_{k}^{ij}}{p_{k}^i}$, \hspace{1em} $1 \leq i, j \leq N$, $1 \leq k \leq n$

### Most compute intensive parts

#### Random number generation & Nearest Neighbor-Search

**Random number generation with `drand48`**

- Linear congruential random number generator in 48bit arithmetic
- Has a long period
- Easy to generate 10,000s of independent random number streams

**Nearest Neighbor-Search**

- Usually done by recursive methods (divide & conquer, kd-tree, ...)
- CUDA does not support recursive functions calls
- Iterative implementation of recursion via stacks is much too slow on CUDA
- Best choice for CUDA is simple Brute Force-Search
Numbers & Facts

Setting
- # MC-Samples: \( M = 100,000 \)
- # Exercise days: \( n = 365 \)
- Grid size: \( N = 100 \rightarrow 500 \)

Computation time on Intel Xeon CPU@3Ghz for \( N = 100 \): 58 sec.

Goal: < 1 sec on a NVIDIA GTX 280 GPU.

Problem dimensions
- \((\pi_k^{ij})\) and \((p_k^{ij})\): 40kB - 1MB per layer \( k \) \( \Rightarrow \) 15 - 365MB total
- # random numbers: 35M
- # NN-Searches: 35M
- \( \Gamma_k \): 800Byte - 4kB per layer \( k \) \( \Rightarrow \) 300kB - 1.5Mb total

Memory setup on CUDA-Devices

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Host/CPU Access</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>local memory</td>
<td>not cached</td>
<td>16kB per thread</td>
</tr>
<tr>
<td>constant memory</td>
<td>cached</td>
<td>64kB per device</td>
</tr>
<tr>
<td>shared memory</td>
<td>n/a</td>
<td>16kB per block</td>
</tr>
<tr>
<td>global memory</td>
<td>not cached</td>
<td>( \approx 1)GB per device</td>
</tr>
</tbody>
</table>
Imagine to split up the processing of the $M$ Paths into several threads:

- $(p_{ij}^k)$ is of size up to 365 MB
- $(p_{ij}^k)$ can only be placed in global memory

**Problem:** concurrent access on the same counter $p_{ij}^k$ by different threads

**Solution:**
- atomic operations
- computational time: 6 sec on GPU

**But:** atomic locks have a very bad impact on the performance, because the force many threads to wait

$$\implies$$ Further optimization possible

---

**Maximize parallel execution**

\[
\text{for } k = 0, \ldots, n - 1 \text{ do }
\]

\[
\text{for } m = 1, \ldots, M \text{ do }
\]

Simulate $X_k$, $\epsilon_k$

Find NN-Index $i$ of $X_k$ in $\Gamma_k$
Find NN-Index $j$ of $A_kX_k + T_k\epsilon_k$ in $\Gamma_{k+1}$

Set
\[
p_{ij}^k + = 1
\]
\[
p_i^k + = 1
\]

**end for**

**end for**

Set $\pi_{ij}^k \leftarrow \frac{p_{ij}^k}{p_i^k}$, $1 \leq i, j \leq N, 1 \leq k \leq n$
Making things faster

2nd Algorithm

Maximize parallel execution

Advantages
- The layers $k$ can be processed independently
- Each thread has to process less data (only the grids $\Gamma_k$ and $\Gamma_{k+1}$)

Important observation
- Computation of only the random numbers and the Nearest Neighbor-Searches on the GPU (without updating the counters) takes for $N = 100$ less than 0.5sec
- Storing the grids $\Gamma_k$ and $\Gamma_{k+1}$ in shared memory (instead of the global one) reduces the computational time to $\approx 0.2$sec.

Results

Combining CPU and GPU computations

Computation for $N = 100$
- Compute only the random numbers and the Nearest Neighbor-Searches on the GPU and store the results in global memory: 0.3sec
- Copy the results back to the CPU (300MB): 0.3sec
- Count the hit Voronoi-Cells on CPU and compute $(\pi_{ij}^k)$: 0.1sec

$\implies$ Overall computational time for $(\pi_{ij}^k)$: 0.75sec
Concerning other choice for the grid size $N$ we got

<table>
<thead>
<tr>
<th>$N$</th>
<th>100</th>
<th>250</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTX 280</td>
<td>0.75sec</td>
<td>1.26sec</td>
<td>2.11sec</td>
</tr>
</tbody>
</table>

Hence, we have shown how reduced the computational time for 365-day swing option in a Gaussian 2-factor model from $\approx 60\text{sec}$ on a CPU to less than $1\text{sec}$ on the GPU ($N = 100$). This means a speed-up of about $80$.

Moreover this estimation of the swing option’s premium has an accuracy of about $2\%$.

**Conclusions**

- identify compute intensive parts
- maximize parallel execution
- split problem up into smaller parts
- avoid locks due to atomic operations
- use fastest memory
- combine GPU and CPU advantages
- sometimes even prefer simple sub-optimal procedure (Nearest Neighbor-Search)