Randomized quasi-Monte Carlo Simulation on GPU

Linlin Xu (Major Prof: Giray Ökten)

Monte Carlo and (randomized) quasi-Monte Carlo methods

Estimate

$$I = \int_{[0,1)^s} f(\boldsymbol{x})d\boldsymbol{x}$$

using sums of the form

$$I_N = \frac{1}{N}\sum_{n=1}^{N} f(\boldsymbol{q}^{(n)})$$

Monte Carlo → $\boldsymbol{q}^{(n)}$ is a (pseudo) random vector from $U(0,1)^s$. Convergence rate is $O(N^{-1/2})$

Quasi-Monte Carlo → $\boldsymbol{q}^{(n)}$ is the n^{th} term of an s-dimensional low-discrepancy sequence (Halton, Sobol', Faure, Niederreiter). Convergence rate is $O(N^{-1}\log^s N)$

Estimate

$$I = \int_{[0,1)^s} f(\boldsymbol{x})d\boldsymbol{x}$$

using sums of the form

$$I_N(\boldsymbol{q}_u) = \frac{1}{N}\sum_{n=1}^{N} f(\boldsymbol{q}_u^{(n)})$$

where $\boldsymbol{q}_u$ is a family of $s-$dimensional low-discrepancy sequences indexed by the random parameter u in **randomized quasi-Monte Carlo**.

$E[I_N(\boldsymbol{q}_u)] = I$
$Var(I_N(\boldsymbol{q}_u)) = O(N^{-2}\log^{2s} N)$

Random-start random-permuted Halton sequence (Rasrap)

The n^{th} term of the **van der Corput sequence**, $\phi_b(n)$, in base b, is defined as follows: First, write the base b expansion of n:

$$n = (a_k \cdots a_1 a_0)_b = a_0 + a_1 b + \cdots + a_k b^k,$$

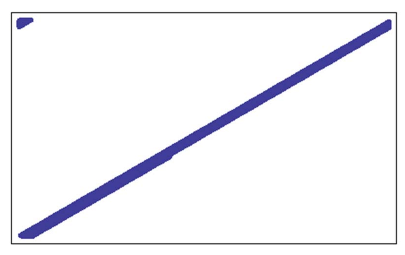
then compute

$$\phi_b(n) = (.a_0 a_1 \cdots a_k)_b = \frac{a_0}{b} + \frac{a_1}{b^2} + \cdots + \frac{a_k}{b^{k+1}}.$$

The **Halton sequence** in the bases $b_1, \dots, b_s$ is $(\phi_{b_1}(n), \dots, \phi_{b_s}(n)),\ n = 1, \cdots, \infty.$

Example: van der Corput sequence in base 2
$\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, \dots$

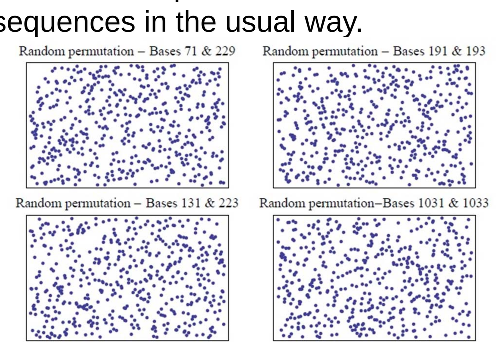
There is a well-known defect of the Halton sequence: in higher dimensions, certain components of the sequence exhibit very poor uniformity. This phenomenon is sometimes described as high correlation between higher bases.
Observing this deficiency of the Halton sequence, different appropriately chosen permutations to scramble digits were introduced.



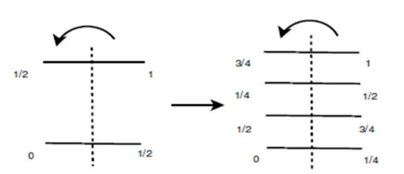
The **random-permuted van der Corput sequence** is:

$$\phi_b(n) = \frac{\sigma(a_0)}{b} + \frac{\sigma(a_1)}{b^2} + \cdots + \frac{\sigma(a_k)}{b^{k+1}}$$

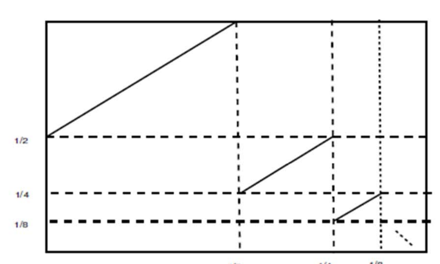
Where σ is a **random** permutation on the digit set $\{0, \dots, b-1\}$. The permuted Halton sequence is obtained from scrambled van der Corput sequences in the usual way.



von Neumann-Kakutani transformationn $T: [0,1) \to [0,1)$, is constructed inductively, by a *splitting* and *stacking* process.



Splitting & stacking process



$T : [0,1) \to [0,1)$

- The orbit of 0 under T is the van der Corput sequence
- The orbit of any $x \in (0,1)$ is a QMC sequence
- Choose x at random from $U(0,1)$ to obtain **random-start Halton sequence**

Parallel computing on GPU

Nodes: $p_1, p_2, \dots$

Want M estimates $\theta_N{}^m = \frac{1}{N}\sum_{i=1}^{N} f(\boldsymbol{q}_i{}^m), m = 1, \dots, M.$

Sequential computing versus "counter-based" computing

	p_1	..	p_m	..	p_M
1	×		q_1^m		×
2	×		q_2^m		×
.			.		
.			.		
N	×		q_N^m		×

Parallel environment

	p_1	p_2	..	p_i	..	p_N
1	×	×		×		×
..						
m	q_1^m	q_2^m		q_i^m		q_N^m
..						
M	×	×		×		×

Massively parallel environment

Problems:
- Pricing collateralized mortgage obligations (CMO)
- Caplet pricing with LIBOR market model

Computing environment:
- CPU: Intel i7-2630QM
- GPU: Nvidia GeForce GT 540M

Sequences used:
Rasrap, Philox, XORWOW, Twister, Sobol'

London interbank offered rate (LIBOR) model

The dynamics of the forward LIBOR rates follow a system of SDEs:

$$\frac{dL_n(t)}{L_n(t)} = \sum_{j=\eta(t)}^{n} \frac{\delta_j(t)L_j(t))\sigma_n(t)^{\intercal}\sigma_j(t)}{1+\delta_j L_j(t)} dt + \sigma_n(t)^{\intercal}(t)dW(t), 0 \le t \le T_n, n = 1, \dots, M,$$

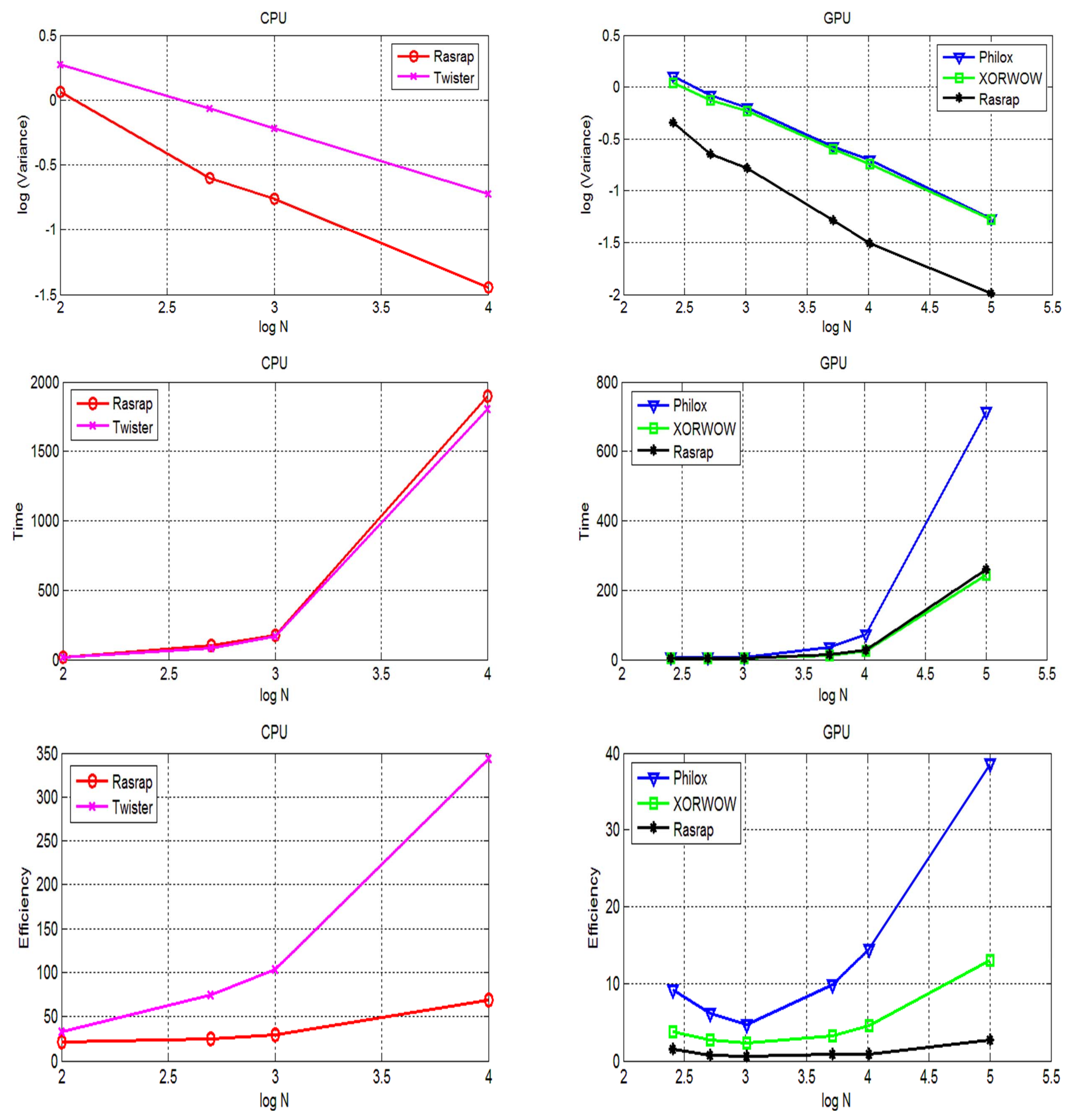
where $L_n(t)$ is the forward rate at time t over the period $[T_n, T_{n+1}]$, σ is volatility and δ denotes the lengths of the intervals between maturities.

Fix a time grid $0 = t_0 < t_1 < \cdots < t_m < t_{m+1}$ to simulate the LIBOR market model. Take $t_i = T_i$ so the simulation goes directly from one maturity date to the next. Assuming a constant volatility σ in the simulation:

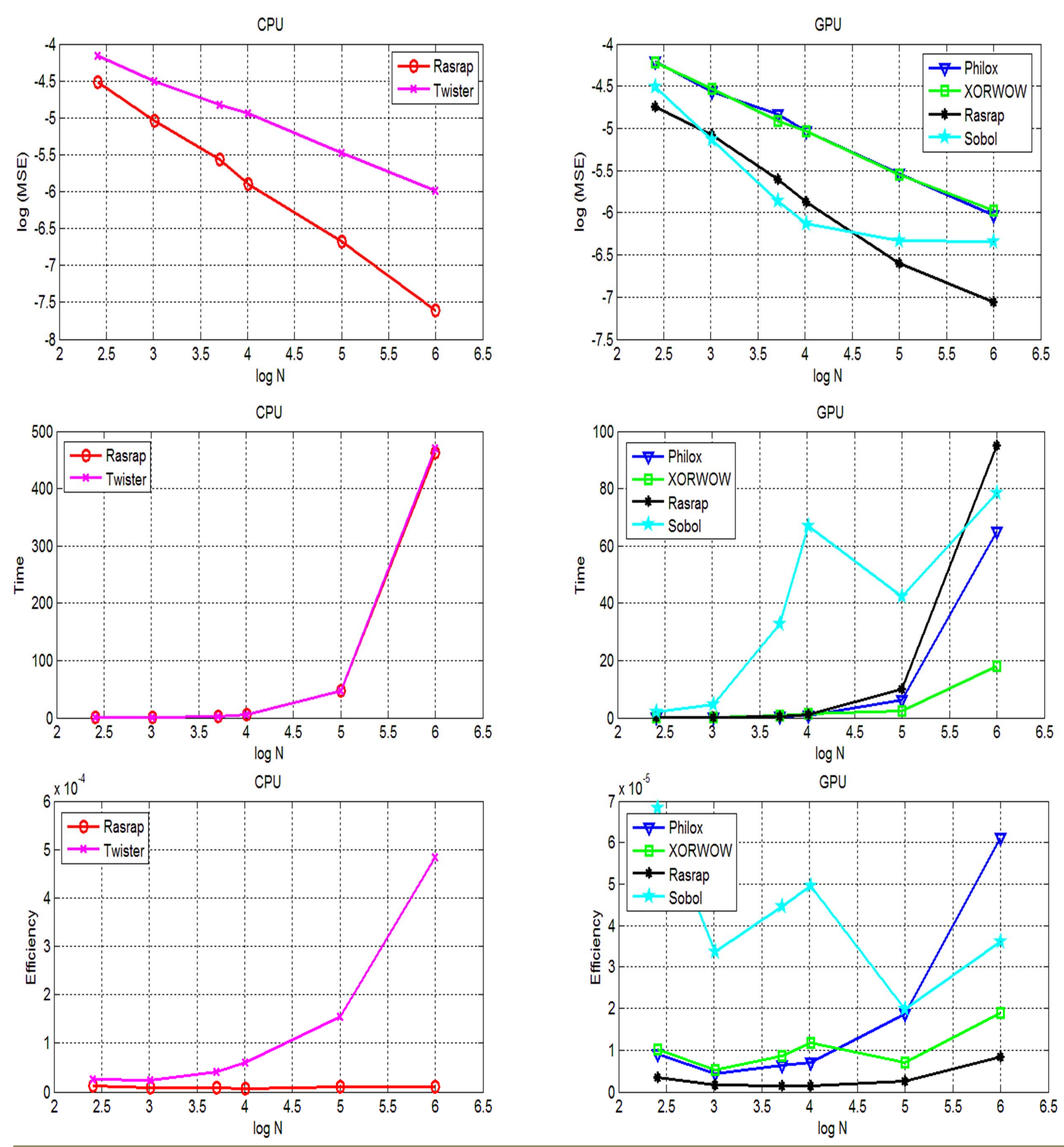
$$\hat{L}_n(t_{i+1}) = \hat{L}_n(t_i) + \mu_n(\hat{L}(t_i), t_i)\, \hat{L}_n(t_i)[t_{i+1} - t_i] + \hat{L}_n(t_i)\sqrt{t_{i+1} - t_i}\sigma_n(t_i)^{\intercal} Z_{i+1},$$

where $\mu_n(\hat{L}(t_i), t_i) = \sum_{j=\eta(t_i)}^{n} \frac{\delta_j(t)\,\hat{L}_j(t_i)\sigma_n(t_i)^{\intercal}\sigma_j(t_i)}{1+\delta_j \hat{L}_j(t_i)}$ and $Z_1, Z_2, \dots$ are independent $N(0, I)$ random vectors in $\mathbb{R}^d$. Here hats are used to identify discretized variables.

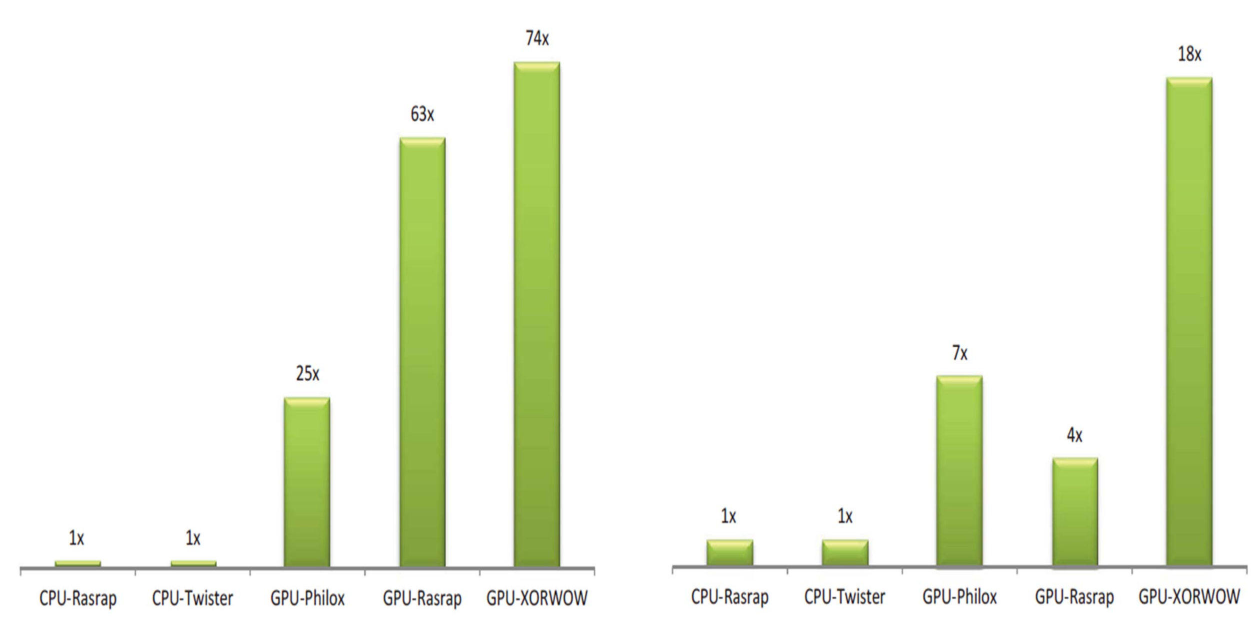
Pricing collateralized mortgage obligations (CMO)



Pricing caplets with LIBOR



Speedup



- Rasrap has the best efficiency among all sequences used in simulations
- The observed convergence rate for Monte Carlo sequences is between $O(N^{-0.49})$ and $O(N^{-0.52})$
- The observed convergence rate for Rasrap is between $O(N^{-0.63})$ and $O(N^{-0.85})$