Optimisation of MLMC on FPGAs

Mike Giles, Irina Nimerenco

Mathematical Institute, University of Oxford

Online Monte Carlo Seminar, October 8, 2024

Outline

- FPGAs
- Monte Carlo for option pricing
- Multilevel Monte Carlo
- approximate Normal random variables
- fixed-point numbers and arithmetic
- rounding error analysis
- optimisation of combined cost/error model
- conclusions and future work

FPGAs

Field-Programmable Gate Arrays have been around for a long time (Altera part of Intel, Xilinx part of AMD) with some niche application areas, including chip simulation and low-latency options trading.

Potentially very efficient for low-precision fixed-point arithmetic, with the flexibility to specify how many bits are used for each variable.

By exploiting Multilevel Monte Carlo, I think they could be really useful in computational finance.

Monte Carlo for Option Pricing

We approximate solutions of SDEs which in 1D have the form

$$\mathrm{d}S_t = a(S_t, t)\, \mathrm{d}t + b(S_t, t)\, \mathrm{d}W_t$$

where $\mathrm{d}W_t$ is the increment of a Brownian motion – Normally distributed with variance $\mathrm{d}t$.

This is usually approximated by the simple Euler-Maruyama method

$$\widehat{S}_{t_{n+1}} = \widehat{S}_{t_n} + a(\widehat{S}_{t_n}, t_n)\, h + b(\widehat{S}_{t_n}, t_n)\, \Delta W_n$$

with uniform timestep h, and increments ΔW_n with variance h.

In simple applications, we want to estimate the expected value $\mathbb{E}[P]$ where P is a function of the final path value:

$$\widehat{P} \equiv f(\widehat{S}_T)$$

Monte Carlo for Option Pricing

The Monte Carlo estimate for $\mathbb{E}[\widehat{P}]$ is an average of N independent samples based on random inputs $\omega^{(n)}$:

$$Y = N^{-1} \sum_{n=1}^{N} \widehat{P}(\omega^{(n)}).$$

If samples have variance V and cost C, then ε RMS accuracy requires

$$N \approx \varepsilon^{-2} V$$

samples, at a total cost of approximately

$$\varepsilon^{-2} V\, C$$

Two-level Monte Carlo

If we want to estimate $\mathbb{E}[\widehat{P}_1]$ but it is much cheaper to simulate $\widehat{P}_0 \approx \widehat{P}_1$, then since

$$\mathbb{E}[\widehat{P}_1] = \mathbb{E}[\widehat{P}_0] + \mathbb{E}[\widehat{P}_1 - \widehat{P}_0]$$

we can use the estimator

$$N_0^{-1} \sum_{n=1}^{N_0} \widehat{P}_0^{(0,n)} \; + \; N_1^{-1} \sum_{n=1}^{N_1} \left(\widehat{P}_1^{(1,n)} - \widehat{P}_0^{(1,n)} \right)$$

Optimising the number of samples N_0, N_1, the total cost for ε RMS accuracy is

$$\varepsilon^{-2} \left(\sqrt{V_0\, C_0} + \sqrt{V_1\, C_1} \right)^2$$

where V_0, C_0 are variance and cost of $\widehat{P}_0$,

and V_1, C_1 are variance and cost of $\widehat{P}_1 - \widehat{P}_0$.

Multilevel Monte Carlo

Natural generalisation: given a sequence $\widehat{P}_0, \widehat{P}_1, \ldots, \widehat{P}_L$

$$\mathbb{E}[\widehat{P}_L] = \sum_{\ell=0}^{L} \mathbb{E}[\Delta \widehat{P}_\ell]$$

with $\Delta \widehat{P}_\ell \equiv \widehat{P}_\ell - \widehat{P}_{\ell-1}$, and $\widehat{P}_{-1} \equiv 0$, we can use the estimator

$$\sum_{\ell=0}^{L} \left\{ N_\ell^{-1} \sum_{n=1}^{N_\ell} \left(\Delta \widehat{P}_\ell^{(\ell,n)} \right) \right\}$$

with independent estimation for each level of correction. The total cost generalises to

$$\varepsilon^{-2} \left(\sum_{\ell=0}^{L} \sqrt{V_\ell C_\ell} \right)^2$$

where V_ℓ, C_ℓ are the variance and cost of $\Delta \widehat{P}_\ell$.

Nested Multilevel Monte Carlo

For the FPGA, we nest the 2-level treatment inside the standard MLMC to get

$$\sum_{\ell=0}^{L} \left\{ N_\ell^{-1} \sum_{n=1}^{N_\ell} \left(\widetilde{\Delta \widehat{P}_\ell}^{(\ell,n,1)} \right) \right\} + \sum_{\ell=0}^{L} \left\{ \widetilde{N}_\ell^{-1} \sum_{n=1}^{\widetilde{N}_\ell} \left(\Delta \widehat{P}_\ell^{(\ell,n,2)} - \widetilde{\Delta \widehat{P}_\ell}^{(\ell,n,2)} \right) \right\}$$

where $\widetilde{\Delta \widehat{P}_\ell}$ corresponds to a low-accuracy fixed-point calculation on an FPGA. The total cost is then approximately

$$\varepsilon^{-2} \left(\sum_{\ell=0}^{L} \sqrt{V_\ell \, \widetilde{C}_\ell} + \sqrt{\widetilde{V}_\ell \, C_\ell} \right)^2$$

where $\widetilde{C}_\ell$ is the cost of evaluating $\widetilde{\Delta \widehat{P}_\ell}$, and $\widetilde{V}_\ell = \mathbb{V}\left[\Delta \widehat{P}_\ell - \widetilde{\Delta \widehat{P}_\ell} \right]$.

Nested Multilevel Monte Carlo

On level ℓ,

$$\sqrt{V_\ell\, \widetilde{C}_\ell} + \sqrt{\widetilde{V}_\ell\, C_\ell} = \sqrt{V_\ell\, C_\ell}\ \left(\sqrt{\widetilde{C}_\ell\, / C_\ell} + \sqrt{\widetilde{V}_\ell\, / V_\ell}\ \right)$$

so our objective is to simultaneously achieve $\widetilde{C}_\ell \ll C_\ell$, $\widetilde{V}_\ell \ll V_\ell$, which implies an optimisation, trading off accuracy, $\widetilde{V}_\ell$, with cost, $\widetilde{C}_\ell$.

What can we adjust?

- accuracy in generating Normal random variables from uniform r.v.'s
- number of bits for all variables in the calculation

Approximate Normal random variables

Given uniform $(0,1)$ random variables U (which can be generated efficiently on both CPUs and FPGAs) the usual way to convert them to standard Normal r.v.'s is through the transform

$$Z = Q(U)$$

where $Q \equiv \Phi^{-1}$ is the inverse of the Normal CDF.

For the FPGA, we will use the leading d random bits of U (giving us $\widetilde{U}$) for an approximate conversion,

$$\widetilde{Z} = \widetilde{Q}(\widetilde{U}).$$

Approximate Normal random variables

Remembering we want to keep it cheap, there are 3 good options:

- piecewise constant approximation on intervals of size 2^{-d}

  implement through LUT (Look-Up Table) based on d bits – not clear what the "cost" of this is on a FPGA

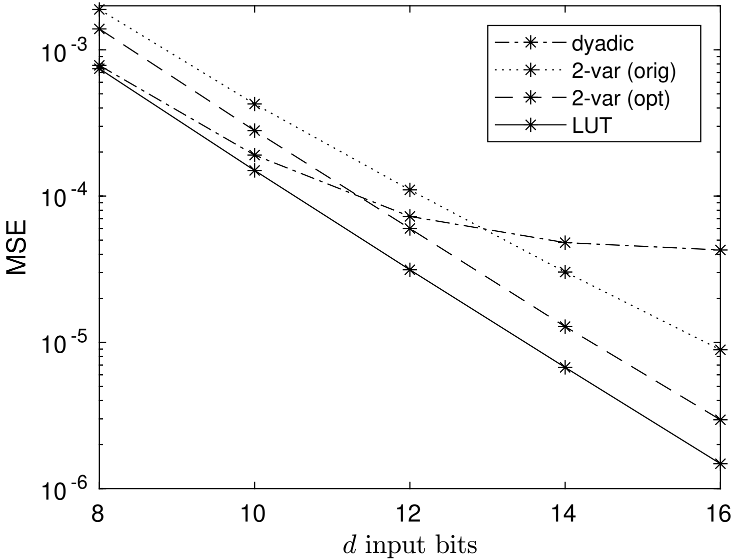
- piecewise linear approximation on dyadic intervals, $[2^{-(n+1)}, 2^{-n}]$

  used on FPGAs previously by Luk, Cheung *et al*, and probably the basis of Intel's vectorised implementation with higher polynomials

- 2-variable method

  split d bits into two sets of 1 sign bit plus $d/2{-}1$ bits for a LUT lookup, to obtain Z_1, Z_2 each of which is approximately Normal with variance $1/2$, then return $Z_1{+}Z_2$; initial LUT values given by best $d/2$-bit piecewise constant approximation, but can be optimized to improve accuracy of output $Z_1{+}Z_2$

Approximate Normal random variables



Fixed point variables and arithmetic

A floating point variable has representation

$$(-1)^s \; m \, 2^e$$

where the sign bit s, integer exponent e, and the bits representing the mantissa $1 \leq m < 2$ are concatenated in a 32-bit or 64-bit variable.

Fixed-point numbers are similar except that the exponent e is fixed.
If there are d bits, not including the optional sign bit, then a variable is represented as

$$(-1)^s \; n \, 2^{e-d}$$

where $0 \leq n < 2^d$. The fixed exponent e determines the range, and is chosen to be big enough for the requirements of the application.

Fixed point variables and arithmetic

When adding two numbers,

$$c = a + b$$

the "cost" is roughly proportional to $\frac{1}{2}(d_a + d_b)$, and when the output is rounded the rounding error is bounded by $2^{e_c - d_c - 1}$.

When multiplying two numbers,

$$c = a \times b$$

the "cost" is roughly proportional to $d_a\, d_b \leq \frac{1}{2}(d_a^2 + d_b^2)$, and the rounding error is again bounded by $2^{e_c - d_c - 1}$.

Error analysis

If an output P is based on a sequence of calculations of intermediate variables x_i, each with a rounding error δx_i, then to leading order the error in the output is given by

$$\delta P = \sum_i \overline{x}_i \; \delta x_i, \qquad \overline{x}_i \equiv \frac{\partial P}{\partial x_i}$$

For a single path calculation, $\overline{x}_i$ can be evaluated efficiently for all variables using adjoint sensitivity analysis (like back-propagation in machine learning).

Error analysis

The errors δx_i and sensitivities $\overline{x}_i$ will vary from path to path, due to the input random numbers.

Being conservative, with possible dependency in the errors, we have

$$\sqrt{\mathbb{V}[\delta P]} \;\leq\; \sum_i \sqrt{\mathbb{V}[\overline{x}_i\, \delta x_i]} \;\leq\; \sum_i \sqrt{\mathbb{E}[\overline{x}_i^2]}\; 2^{e_i - d_i - 1}$$

On the other hand, if we model the errors as statistically independent, and the rounding error is well modelled as uniformly distributed over $[-2^{e_i-d_i-1}, 2^{e_i-d_i-1}]$, then we get

$$\mathbb{V}[\delta P] \;=\; \sum_i \mathbb{V}[\overline{x}_i\, \delta x_i] \;\leq\; \sum_i \mathbb{E}[\overline{x}_i^2]\; 4^{e_i - d_i}/12$$

Geometric Brownian Motion

We test the error modelling using a very simple example:

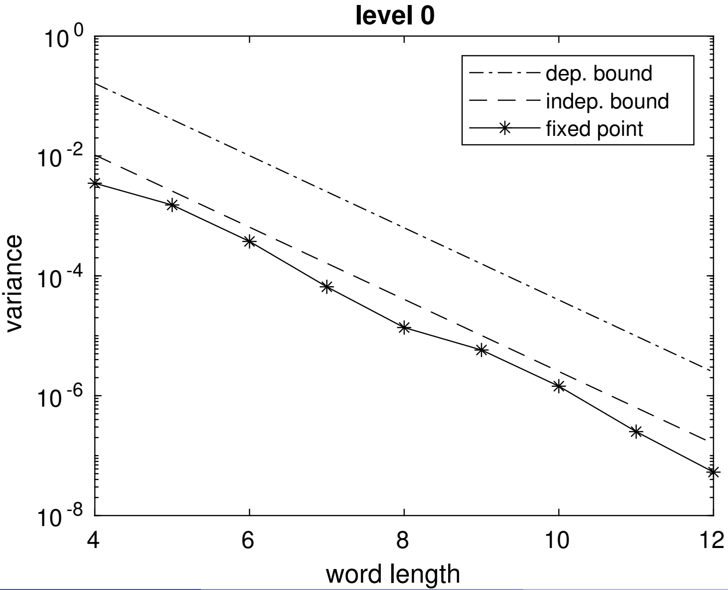
Input: timestep h, timesteps N, constants $\mathrm{con}_1 = rh$, $\mathrm{con}_2 = \sigma\sqrt{h}$, initial states $S_\ell = S_{\ell-1} = S_0$, strike K

for $n = 0, N-1$ **do**
  if n even **then**
    $S_{\ell-1}^{old} := S_{\ell-1}$
  end if
  generate Normal r.v. $Z \sim N(0,1)$
  $\mathrm{mul}_1 := \mathrm{con}_2 \times Z$
  $\mathrm{sum}_1 := \mathrm{con}_1 + \mathrm{mul}_1$
  $\mathrm{mul}_2 := S_\ell \times \mathrm{sum}_1$
  $S_\ell \quad := S_\ell + \mathrm{mul}_2$
  $\mathrm{mul}_2 := S_{\ell-1}^{old} \times \mathrm{sum}_1$
  $S_{\ell-1} := S_{\ell-1} + \mathrm{mul}_2$
end for

$\Delta P_\ell := \max(S_\ell - K, 0) - \max(S_{\ell-1} - K, 0)$

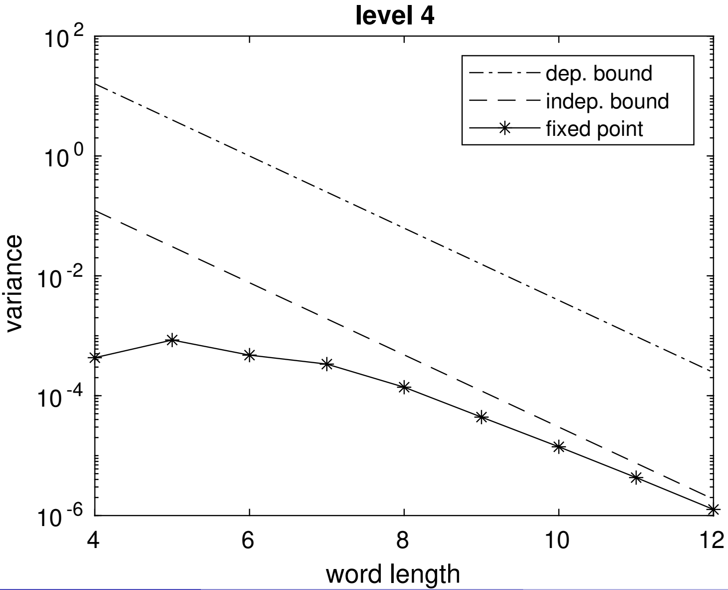
Geometric Brownian Motion

Errors when using same number of bits d for all variables.



Geometric Brownian Motion

Errors when using same number of bits d for all variables.



Optimisation

On the basis of these results, we use the error model

$$\widetilde{V} = \sum_i \mathbb{E}[\overline{x}_i^2]\; 4^{e_i - d_i}/12$$

Using the same number of bits for both S_ℓ and $S_{\ell-1}$, and all instances of Z, sum_1, mul_1, mul_2, this gives us a sum over 7 terms:

$$\widetilde{V} = \sum_{k=1}^{7} V_k \, 4^{-d_k}$$

Counting up the additions and multiplications involving each variable, we get a corresponding quadratic cost model:

$$\widetilde{C} = \frac{1}{2} \sum_{k=1}^{7} a_k d_k + m_k d_k^2$$

Note this is ignoring the error and cost due to the approximate Normal random numbers.

Optimisation

The goal now is to minimise

$$\sqrt{\widetilde{C}_\ell \,/\, C_\ell} + \sqrt{\widetilde{V}_\ell \,/\, V_\ell}$$

where V_ℓ is known and we arbitrarily set C_ℓ to be 10^4 per timestep on the CPU.

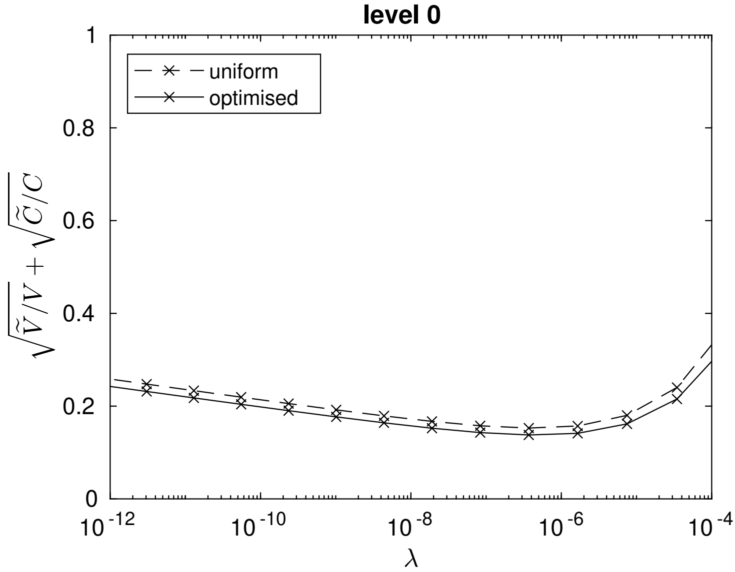
Setting the partial derivative w.r.t. d_n to zero gives the equation

$$\frac{\partial \widetilde{V}_\ell}{\partial d_k} + \lambda \, \frac{\partial \widetilde{C}_\ell}{\partial d_k} = 0$$

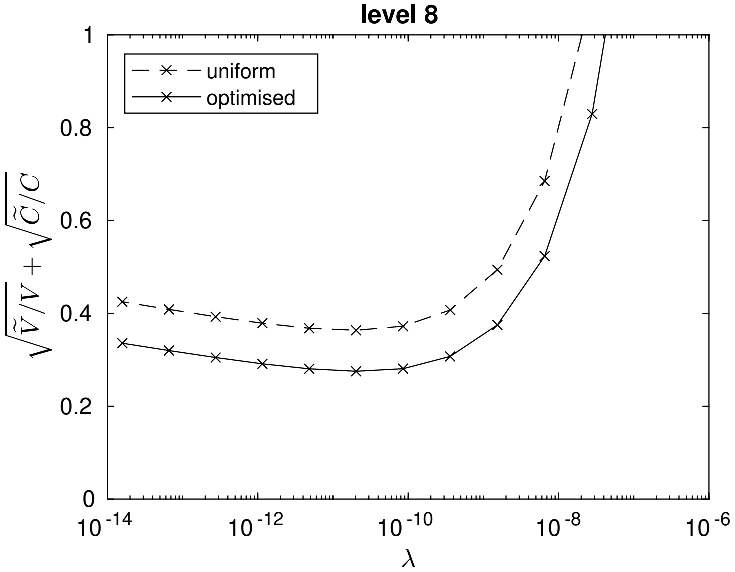
where $\lambda = \sqrt{V_\ell \, \widetilde{V}_\ell / C \, \widetilde{C}_\ell}$ gives the optimal tradeoff between cost and variance.

The optimisation can be carried out by doing a golden section minimisation on λ, using a Newton iteration to find the optimal d_k for each λ.

Geometric Brownian Motion

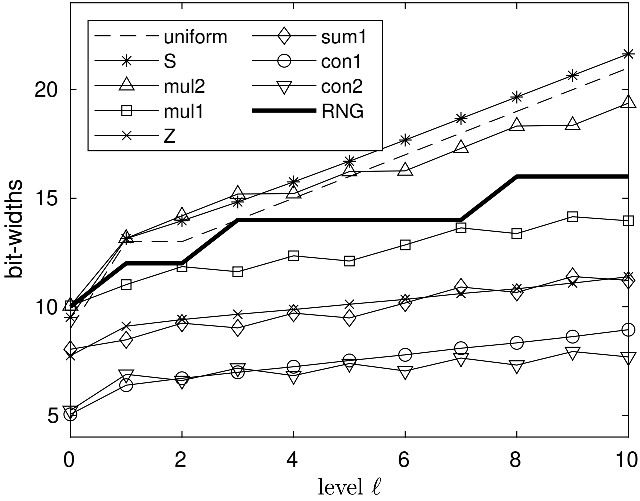


Geometric Brownian Motion



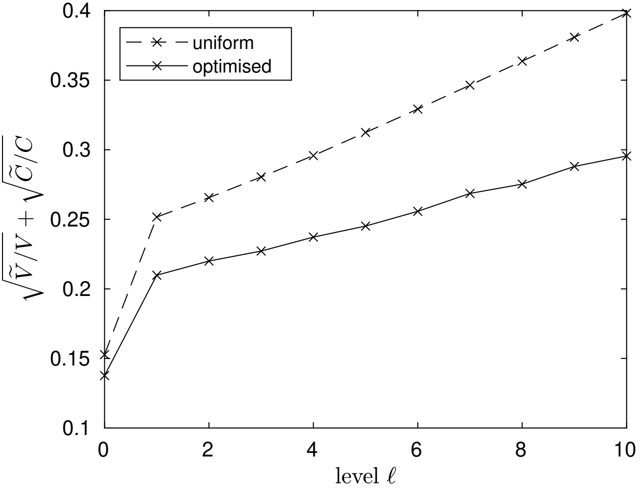
Geometric Brownian Motion

Optimal bit-widths for different variables



Geometric Brownian Motion

Optimal "cost" on each level



Conclusions

We have taken the optimisation as far as we can without doing the FPGA implementation.

Taking the overall cost to be

$$\varepsilon^{-2} \left(\sum_{\ell=0}^{L} \sqrt{V_\ell \, \widetilde{C}_\ell} + \sqrt{\widetilde{V}_\ell \, C_\ell} \right)^2$$

the reduction factor for the GBM testcase is approximately

- 15 with uniform bit-widths on each level
- 24 with optimised bit-widths for each variable on each level

This ignores the cost of generating the random numbers, but we think the proposed 2-variable approach will be very efficient.

Future work

- FPGA implementation, probably using AMD's Vitis Unified Software framework for C/C++ programming
- assessment of the computational cost of the different ways of converting uniform r.v.'s to Normals
- assessment, and possible improvement, of the cost model for fixed-point arithmetic
- implementation of the CPU part of the algorithm, and validation of the MLMC telescoping summation
- determination of overall cost savings
- comparison to use of reduced precision floating point arithmetic on CPUs (AVX512-FP16 on Intel "Sapphire Rapids" Xeon) and GPUs (bfloat16, fp16 on NVIDIA and AMD GPUs)
- extension to other SDEs

References

Irina Nimerenco, “A nested Multilevel Monte Carlo framework for efficient simulations on FPGAs”, MSc thesis, 2024.

R.C.C. Cheung, D.-U. Lee, W. Luk, J.D. Villasenor. “Hardware generation of arbitrary random number distributions from uniform distributions via the inversion method”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 15(8):952–962, 2007

C. Brugger, C. de Schryver, N. Wehn, S. Omland, M. Hefter, K. Ritter, A. Kostiuk, and R. Korn. “Mixed precision Multilevel Monte Carlo on hybrid computing systems”. In Proceedings of the IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr), pp.215-222. IEEE, 2014.

M.B. Giles, O. Sheridan-Methven. “Analysis of nested multilevel Monte Carlo using approximate normal random variables”. SIAM/ASA Journal on Uncertainty Quantification, 10:200–226, 2021.