Extended Abstract: A Functional Approach to Monte Carlo based American Option Pricing

Wojciech Michal Pawlak
Department of Computer Science,
University of Copenhagen
SimCorp Technology Labs
wmp@di.ku.dk,wmpk@simcorp.com

Martin Elsman
Department of Computer Science,
University of Copenhagen
mael@di.ku.dk

Cosmin Eugen Oancea
Department of Computer Science,
University of Copenhagen
cosmin.oancea@diku.dk

Abstract

We study the feasibility and performance efficiency of expressing a complex financial numerical algorithm with high-level functional parallel constructs. The algorithm we investigate is a least-square regression-based Monte-Carlo simulation for pricing American options. We propose an accelerated parallel implementation in Futhark, a high-level functional data-parallel language. The Futhark language targets GPUs as the compute platform and we achieve a performance comparable to an implementation optimised by NVIDIA CUDA engineers. In absolute terms, we can price a put option with 1 million simulation paths and 100 time steps in 20ms on a NVIDIA Tesla V100 GPU.

CCS Concepts • **Computing methodologies** → **Shared memory algorithms**; **Massively parallel algorithms**; **Massively parallel and high-performance simulations**; Parallel programming languages; • **Applied computing** → **Economics**; • **Computer systems organization** → *Multicore architectures*;

Keywords High-Performance Computing, Parallel (GPU) Programming, Functional Programming, Compilers, Computational Finance, Derivative Pricing

ACM Reference Format:
Wojciech Michal Pawlak, Martin Elsman, and Cosmin Eugen Oancea. 2019. Extended Abstract: A Functional Approach to Monte Carlo based American Option Pricing. Presented at *ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing (FHPNC'19).* 4 pages.

1 Introduction

Pricing American options is a fundamental business case in the financial services sector as such financial instruments are widely traded in the derivative markets. American options can be exercised at any time between the present date and the time to maturity. This aspect puts them in contrast to European options, that can only be exercised at their maturity. In the usual case, the option holder is expected to exercise the option as soon as it is more profitable to do so rather than wait until its expiration. Effectively, the value of an American option is the value achieved by exercising optimally. This embedded optimisation (optimal stopping) problem is the main challenge. As there is no general closed-form formula solutions [17], it is necessary to approximate the option value accurately with a numerical simulation. This is a substantial and timely computational effort. It has to be significantly reduced to become acceptable for time-critical applications in financial practice. Therefore it is a valid case for performance acceleration through massive parallelisation on GPUs.

Current most efficient accelerated simulations are implemented in dedicated languages and frameworks like CUDA [1, 15, 27], MPI [10], OpenMP [32], and other technologies [9]. The challenge with these implementations is the poor expressibility, which makes them inaccessible to domain experts and limits them to the specialist developers. It also results in code that is difficult to maintain. On top of that, one needs to be aware of the low-level properties of the underlying hardware architecture.

We propose a functional approach to the implementation of an accelerated option pricing model. The use of high-level parallel constructs lets us express the algorithm in an intuitive manner, without the implementation concerns of mapping the code to the architecture. The Futhark language and the optimizing compiler behind it make this possible [21]. Previous work has investigated the use of Futhark for implementing Monte Carlo based European option pricing [2], which covered a number of advanced features that the present work does not consider, including Sobol sequence generation [18]. Quite a few approaches exist aiming at generating efficient data-parallel GPGPU code for applications written using high-level array language constructs, including the work on Obsidian [11, 29, 30] and Accelerate [8], which are both domain specific languages embedded in Haskell, but, which do not feature arbitrary nested parallelism. Approaches that support arbitrary nested parallelism includes the seminal work on flattening of nested parallelism in NESL [4, 5], which was extended to operate on a richer set of values in Data-parallel Haskell [7], and the work on data-only flattening [33]. However, such general compiler-based flattening is challenging to implement efficiently on GPUs [3]. Other promising attempts at compiling NESL to GPUs include Nessie [28], which is still under development, and CuNesl [33], which aims at mapping different levels of nested

FHPNC'19, August 18, 2019, Berlin, Germany
2019.

parallelism to different levels of parallelism on the GPU, but which lacks critical optimisations such as fusion.

The main contribution of this work is an efficient Futhark implementation of a pricing model for American options, a Longstaff-Schwartz algorithm using Monte Carlo Simulation with Least-Square Regression (abbreviated LSMC) [25]. We present our implementation and compare the performance results to a benchmark CUDA version. We obtain results that are on par with a hand-tuned version written in a dedicated low-level programming model.

2 Design and Implementation

Futhark is a statically typed parallel functional array language. The language is based on an ML or Haskell style syntax and is equipped with a number of second-order array combinators (SOACs), such as `map`, `reduce`, `scan`, and `filter`. The Futhark language features a higher-order module system [14], polymorphism, and a restricted form of higher-order functions [23], concepts that are all eliminated at compile time and introduce no overhead at runtime. The Futhark compiler supports aggressive fusion of parallel constructs [20], and specialised code generators for key parallel operators, such as map-scan and (segmented) map-reduce compositions [19, 24]. For generating parallel GPU kernels, Futhark flattens nested parallel constructs using a number of flattening techniques [21, 22].

Several authors have proposed the use of regression to estimate continuation values from simulated paths [6, 12, 25, 31]. The structure of a regression-based simulation algorithm can be summarised as follows.

1. Generate a matrix $W(n, m)$ of random numbers drawn from a standard normal distribution.
2. Using W, simulate by *forward induction* n independent paths $S_{1j}, \ldots, S_{mj}, j = 1, \ldots, n$ of Geometric Brownian Motion stochastic processes for the underlying asset prices.
3. At the last step T (at maturity), compute the option value $\hat{V}_{mj} = p_m(X_{mj}), j = 1, \ldots, n$ applying the payoff function p.
4. Apply *backward induction* for each step $i = m-1, \ldots, 1$ to compute cashflows:
   a. Select the paths that are in-the-money.
   b. Compute the matrix A and the right hand side b of the least-square linear equation $Ax = b$ to approximate the continuation function from asset prices S_i and cashflows $\hat{V}_{i+1}$ only for the paths that are in-the-money.
   c. Decide to early-exercise based on:

$$\hat{V}_{ij} = \begin{cases} p_i(S_{ij}), & p_i(S_{ij}) \geqslant \hat{C}_i(S_{ij}); \\ \hat{V}_{i+1,j}, & p_i(S_{ij}) < \hat{C}_i(S_{ij}). \end{cases} \quad (1)$$

5. Set $\hat{V}_0 = (\hat{V}_{11} + \cdots + \hat{V}_{1n})/n$

The computational effort of a Monte Carlo simulation is determined by the number of simulation paths and time steps. A large number of paths n, usually 100.000 to 1.000.000, needs to be generated to obtain an accurate value approximation [16]. The number of time steps m is bound to the number of early-exercise opportunities and is usually much smaller than n. We use a minimum standard pseudo-random number generator in a parallel skip-ahead fashion.

```
1   map(n)                          -- Path Generation
2     loop(m)
3   transpose
4   map(m)                          -- SVD Preparation
5     loop(n)
6       scan(chunk)
7     map(n) |> reduce(n)
8     map(n)
9   map(m)
10  loop(m)                         -- Main Loop
11    map(n)
12    map(n)
```

Listing 1. High-level structure of the implemented algorithm presented as a combination of possibly nested parallel constructs. The algorithm consists of three parts with n denoting the number of paths and m denoting the number of time steps. The `transpose` function performs matrix transposition.

The backward dynamic programming steps 4a and 4b in the intrinsically sequential loop are the main performance bottlenecks as we need to (1) perform operations on a matrix of size $n \times 3$ in the worst case, (2) deal with matrix sizes that vary across steps (thread divergence), and (3) make sure that threads are synchronised after each step. The payoff function determines the number of relevant (in-the-money) paths at each time step, and thus the total computational effort of this part. We therefore hoist this computation out of the loop and prepare small fixed-size matrices by performing the computation in parallel across time steps. We achieve this goal through a chain of linear algebra transformations, such as SVD and QR decomposition as well as a pseudo-inverse transform. In addition, we specialise the algorithm to work with a 3-degree monomial basis function. We do not claim any contributions to this algorithm. In fact, for this part, our implementation closely follows the implementation proposed by NVIDIA [13, 26]. We focus on the goal to match the performance of this publicly available benchmark implementation. The algorithm outlined in Listing 1 is implemented using a nested composition of sequential `loop`s and parallel `map`, `reduce`, and `scan` constructs.

3 Experimental Results

We have run experiments on a Linux system with a 26-core 2-way HT Intel Xeon Platinum 8167M CPU (2.00GHz), 754 GB DDR RAM and NVIDIA Tesla **V100** SXM2 GPU (2688

Model Parameters	Value
Option Type, Payoff	Put, $\max(K - S)$
Initial spot price (S0)	80.0
Strike price (K)	90.0
Time to maturity (T)	1 year
Risk free rate (r)	5%
Volatility (σ)	30%
Simulation Parameters	
Time steps/Early Exercise dates	100
Paths	1.000.000
Option Price (Binomial Tree)	13.804

Table 1. Set of model and simulation parameters for the American option pricing. We provide an option price obtained from a different numerical method (binomial tree) for reference. We use it to validate the result.

Volta *FP64* cores, 16 GB HBM2) using CUDA 10.1. The pricing test case is presented in Table 1. The performance results are presented in Table 2.

	Path	SVD	Main	Total	Speedup	Value
Ref	4.7	1.8	8.9	**15.4**	1.45×	13.778
V1	5.9	2.1	14.3	**22.3**	1.00×	13.789
V2	5.4	1,9	13.1	**20.4**	1.09×	13.789

Table 2. Execution times for the test case. **Ref** is the CUDA benchmark, while **V1** is Futhark compiled to OpenCL and **V2** is Futhark compiled to CUDA. Both total and partial execution times for each part of the algorithm are shown. Execution times are given in *ms*. The runtimes are averaged based on 250 runs. We compare the speedups against the slowest runtime.

References

[1] L.A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier. 2014. Pricing derivatives on graphics processing units using Monte Carlo simulation. *Concurrency and Computation: Practice and Experience* 26, 9 (June 2014), 1679–1697. https://doi.org/10.1002/cpe.2862

[2] Christian Andreetta, Vivien Bégot, Jost Berthold, Martin Elsman, Fritz Henglein, Troels Henriksen, Maj-Britt Nordfang, and Cosmin E. Oancea. 2016. FinPar: A Parallel Financial Benchmark. *ACM Trans. Archit. Code Optim.* 13, 2, Article 18 (June 2016), 27 pages.

[3] Lars Bergstrom and John Reppy. 2012. Nested Data-parallelism on the Gpu. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming (ICFP '12)*. ACM, New York, NY, USA, 247–258. https://doi.org/10.1145/2364527.2364563

[4] Guy E Blelloch. 1990. *Vector models for data-parallel computing*. Vol. 75. MIT press Cambridge.

[5] Guy E. Blelloch. 1996. Programming Parallel Algorithms. *Communications of the ACM (CACM)* 39, 3 (1996), 85–97.

[6] Jacques F. Carriere. 1996. Valuation of the early-exercise price for options using simulations and nonparametric regression. *Insurance: Mathematics and Economics* 19, 1 (Dec. 1996), 19–30. https://doi.org/10.1016/S0167-6687(96)00004-2

[7] Manuel M. T. Chakravarty, Roman Leshchinskiy, Simon Peyton Jones, Gabriele Keller, and Simon Marlow. 2007. Data Parallel Haskell: A Status Report. In *Int. Work. on Decl. Aspects of Multicore Prog. (DAMP)*. 10–18.

[8] Manuel MT Chakravarty, Gabriele Keller, Sean Lee, Trevor L McDonell, and Vinod Grover. 2011. Accelerating Haskell array codes with multicore GPUs. In *Proc. of the sixth workshop on Declarative aspects of multicore programming*. ACM, 3–14.

[9] Ching-Wen Chen, Kuan-Lin Huang, and Yuh-Dauh Lyuu. 2015. Accelerating the least-square Monte Carlo method with parallel computing. *The Journal of Supercomputing* 71, 9 (Sept. 2015), 3593–3608. https://doi.org/10.1007/s11227-015-1451-7

[10] A. R. Choudhury, A. King, S. Kumar, and Y. Sabharwal. 2008. Optimizations in financial engineering: The Least-Squares Monte Carlo method of Longstaff and Schwartz. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–11. https://doi.org/10.1109/IPDPS.2008.4536290

[11] Koen Claessen, Mary Sheeran, and Bo Joel Svensson. 2012. Expressive Array Constructs in an Embedded GPU Kernel Programming Language. In *Work. on Decl. Aspects of Multicore Prog DAMP*. 21–30.

[12] Emmanuelle Clément, Damien Lamberton, and Philip Protter. 2002. An analysis of a least squares regression method for American option pricing. *Finance and Stochastics* 6, 4 (Oct. 2002), 449–471. https://doi.org/10.1007/s007800200071

[13] Julien Demouth. 2014. Monte-Carlo Simulation of American Options with GPUs. http://on-demand.gputechconf.com/gtc/2014/presentations/S4784-monte-carlo-sim-american-options-gpus.pdf. Presentation at NVIDIA GPU Technology Conference.

[14] Martin Elsman, Troels Henriksen, Danil Annenkov, and Cosmin E. Oancea. 2018. Static Interpretation of Higher-order Modules in Futhark: Functional GPU Programming in the Large. *Proceedings of the ACM on Programming Languages* 2, ICFP, Article 97 (July 2018), 30 pages.

[15] Massimiliano Fatica and Everett Phillips. 2013. Pricing American Options with Least Squares Monte Carlo on GPUs. In *Proceedings of the 6th Workshop on High Performance Computational Finance (WHPCF '13)*. ACM, New York, NY, USA, 5:1–5:6. https://doi.org/10.1145/2535557.2535564 event-place: Denver, Colorado.

[16] Paul Glasserman. 2004. *Monte Carlo methods in financial engineering*. Springer, New York.

[17] Espen Gaarder Haug. 2007. *The Complete Guide to Option Pricing Formulas* (2nd ed.). McGraw-Hill Education, New York.

[18] Troels Henriksen, Martin Elsman, and Cosmin E. Oancea. 2018. Modular Acceleration: Tricky Cases of Functional High-Performance Computing. In *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional High-Performance Computing (FHPC '18)*. ACM, New York, NY, USA.

[19] Troels Henriksen, Ken Friis Larsen, and Cosmin E. Oancea. 2016. Design and GPGPU Performance of Futhark's Redomap Construct. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming (ARRAY 2016)*. ACM, New York, NY, USA, 17–24. https://doi.org/10.1145/2935323.2935326

[20] Troels Henriksen and Cosmin Eugen Oancea. 2013. A T2 graph-reduction approach to fusion. In *Proceedings of the 2nd ACM SIGPLAN workshop on Functional high-performance computing*. ACM, 47–58.

[21] Troels Henriksen, Niels G. W. Serup, Martin Elsman, Fritz Henglein, and Cosmin E. Oancea. 2017. Futhark: Purely Functional GPU-programming with Nested Parallelism and In-place Array Updates. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017)*. ACM, New York,

NY, USA, 556–571. https://doi.org/10.1145/3062341.3062354

[22] Troels Henriksen, Frederik Thorøe, Martin Elsman, and Cosmin Oancea. 2019. Incremental Flattening for Nested Data Parallelism. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP '19)*. ACM, New York, NY, USA, 53–67. https://doi.org/10.1145/3293883.3295707

[23] Anders Kiel Hovgaard, Troels Henriksen, and Martin Elsman. 2018. High-performance defunctionalization in Futhark. In *Symposium on Trends in Functional Programming (TFP'18)*.

[24] Rasmus Wriedt Larsen and Troels Henriksen. 2017. Strategies for Regular Segmented Reductions on GPU. In *Proceedings of the 6th ACM SIGPLAN International Workshop on Functional High-Performance Computing (FHPC 2017)*. ACM, New York, NY, USA, 42–52. https://doi.org/10.1145/3122948.3122952

[25] Francis A. Longstaff and Eduardo S. Schwartz. 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies* 14, 1 (Jan. 2001), 113–147. https://doi.org/10.1093/rfs/14.1.113

[26] NVIDIA. 2014. NVIDIA Developer Blog Code Samples repository at GitHub. https://github.com/NVIDIA-developer-blog/code-samples/tree/master/posts/american-options.

[27] Gilles Pagès and Benedikt Wilbertz. 2012. GPGPUs in computational finance: massive parallel computing for American style options. *Concurrency and Computation: Practice and Experience* 24, 8 (2012), 837–848. https://doi.org/10.1002/cpe.1774

[28] John Reppy and Nora Sandler. 2015. Nessie: A NESL to CUDA Compiler. Presented at the *Compilers for Parallel Computing Workshop (CPC '15)*. Imperial College, London, UK.

[29] Joel Svensson. 2011. *Obsidian: GPU Kernel Programming in Haskell*. Ph.D. Dissertation. Chalmers University of Technology.

[30] Joel Svensson, Mary Sheeran, and Koen Claessen. 2011. Obsidian: A Domain Specific Embedded Language for Parallel Programming of Graphics Processors. In *Proceedings of the 20th International Conference on Implementation and Application of Functional Languages (IFL'08)*. Springer-Verlag, Berlin, Heidelberg, 156–173. http://dl.acm.org/citation.cfm?id=2044476.2044485

[31] J. N. Tsitsiklis and B. Van Roy. 2001. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks* 12, 4 (July 2001), 694–703. https://doi.org/10.1109/72.935083

[32] Shuai Zhang, Zhao Wang, Ying Peng, Bertil Schmidt, and Weiguo Liu. 2017. Mapping of option pricing algorithms onto heterogeneous many-core architectures. *The Journal of Supercomputing* 73, 9 (Sept. 2017), 3715–3737. https://doi.org/10.1007/s11227-017-1968-z

[33] Yongpeng Zhang and Frank Mueller. 2012. CuNesl: Compiling Nested Data-Parallel Languages for SIMT Architectures. In *Proceedings of the 2012 41st International Conference on Parallel Processing (ICPP'12)*. IEEE Computer Society, Washington, DC, USA, 340–349.