Analysis of Parallel Processor Architectures for the Solution of the Black-Scholes PDE

Endre László *, Zoltán Nagy † *, Michael B. Giles ‡, István Reguly ‡, Jeremy Appleyard § and Péter Szolgay *†
*Pázmány Péter Catholic University, Budapest, Hungary
Email: {laszlo.endre,szolgay}@itk.ppke.hu
†Hungarian Academy of Sciences, Institute for Computer Science and Control, Budapest, Hungary
Email: nagyz@sztaki.mta.hu
‡University of Oxford, Oxford, United Kingdom
Email: mike.giles@maths.ox.ac.uk, istvan.reguly@oerc.ox.ac.uk
§NVIDIA Corporation, United Kingdom
Email: jappleyard@nvidia.com

***Abstract*—Common parallel computer microarchitectures offer a wide variety of solutions to implement numerical algorithms. The efficiency of different algorithms applied to the same problem vary with the underlying architecture which can be a multi-core CPU, many-core GPU, Intel's MIC (Many Integrated Core) or FPGA architecture. Significant differences between these architectures exist in the ISA (Instruction Set Architecture) and the way the compute flow is executed. The way parallelism is expressed changes with the ISA, thread management and customization available on the device. These differences pose restrictions to the implementable algorithms. The aim of the work is to analyze the efficiency of the algorithms through the architectural differences. The problem at hand is the one-factor Black-Scholes option pricing equation which is a parabolic PDE solved with explicit and implicit time-marching algorithms. In the implicit solution a scalar tridiagonal system of equations needs to be solved. The possible CPU, GPU implementations along with novel FPGA solutions with HLS (High Level Synthesis) will be shown. Performance is also analyzed and remarks on efficiency are made.**

I. Introduction

The aim of the present paper is to further examine the architectural, programmability and development issues regarding novel CPU, GPU and FPGA architectures in the case of one dimensional finite difference problem like the one-factor Black-Scholes (BS) PDE. The BS PDE is a parabolic type defined with Dirichlet boundary conditions. Therefore the solution of this problem is similar to the solution of the heat equation in one dimension. The problem can be solved using explicit and implicit time marching. Although the explicit solution is programmatically much simpler, it requires significantly more time step computations than the implicit method, due to stability limit of the explicit method. Besides, the implicit method doesn't pose such restrictions on the grid resolution which is defined by the convergence criteria of the explicit method.

The balance of computational speed, programming effort and power efficiency are the key factors that decide where a certain architecture will be used in the engineering and research practice. Therefore, in the current study the following architecture-parallelisation tool combinations were chosen:

- Intel Xeon 2 socket server CPU with Sandy Bridge architecture: algorithms implemented using AVX ISA (Instruction Set Architecture) intrinsics in C/C++.
- NVIDIA Tesla K40 GPU with Kepler architecture: algorithms implemented with CUDA C programming language
- Xilinx Virtex7 FPGA: algorithms implemented with Vivado HLS (High Level Synthesis) C/C++ language.

The way parallelism is executed and implemented on these hardware platforms is usually very diverse. This is especially true in the case of the FPGA where the implementation of parallelism uniquely depends on the problem at hand and the effort of the developer. Although the standard way of FPGA development is through the use of VHDL or Verilog hardware description languages, the development effort with these approaches are not comparable with the C/C++ and CUDA C development efforts, as hardware description languages are more fine-grained. Therefore, Vivado HLS (High Level Synthesis) has been chosen to create a high performing FPGA implementation from C/C++ source code.

The literature on the finite difference solution of the Black Scholes PDE is abundant, but only a few papers provide a thorough comparison of solvers on novel CPU, GPU and FPGA architectures. See [1] for efficient algorithms on CPUs and GPUs, [2] for comparison between GPU and FPGA implementations. FPGA implementations of the explicit solver are studied in [3] and [4], while implicit solution is considered in [5].

II. Black-Scholes equation and its numerical solution

The Black-Scholes (BS) PDE for derivative security pricing is a celebrated tool of the Black-Scholes theory [6]. The one-factor BS in the case of European call options is shown on Eq. (1).

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} - rV = 0 \qquad (1)$$

The BS equation is a second order, parabolic, convection-diffusion type PDE sharing common numerical features with

978-1-4799-8391-9/15/$31.00 ©2015 IEEE

the heat equation. Solving these equations with explicit or implicit time-marching is feasible in 1 dimension (one-factor form). In higher dimensions the cost of the implicit solution increases greatly which can be avoided by more sophisticated numerical methods. Such methods are not the scope of the present paper. Both explicit and implicit solutions iterate in time backwards, since the problem is a final value problem rather than an initial value problem. After explicit (backward time differentiation) discretization of Eq. (1) the BS PDE boils down to the algebraic expression shown in Eq. (2). Evaluating this expression gives the price curve in the next time-step.

$$V_k^{(n+1)} = a_k V_{k-1}^{(n)} + b_k V_k^{(n)} + c_k V_{k+1}^{(n)} \tag{2}$$

with coefficients

$$\begin{aligned} a_k &= \tfrac{1}{2}\sigma^2 k^2 \Delta t - \tfrac{1}{2} r k \Delta t \\ b_k &= 1 - \sigma^2 k^2 \Delta t - r\Delta t \\ c_k &= \tfrac{1}{2}\sigma^2 k^2 \Delta t + \tfrac{1}{2} r k \Delta t \end{aligned}$$

where $(n) = 0, ..., N-1$ superscript is the time coordinate with Δt time step, $k = 0, ..., K-1$ subscript is the price coordinate with ΔS price step (price step is factorized out), σ is the volatility of the underlying (risky) asset, r is the risk-free interest rate.

Eq. (3) shows the implicit form of the solution of BS PDE, which requires the solution of a tridiagonal system of equations.

$$a_k V_{k-1}^{(n+1)} + b_k V_k^{(n+1)} + c_k V_{k+1}^{(n+1)} = V_k^{(n)} \tag{3}$$

where

$$\begin{aligned} a_k &= -\tfrac{1}{2}\sigma^2 k^2 \Delta t + \tfrac{1}{2} r k \Delta t \\ b_k &= 1 + \sigma^2 k^2 \Delta t + r\Delta t \\ c_k &= -\tfrac{1}{2}\sigma^2 k^2 \Delta t - \tfrac{1}{2} r k \Delta t \end{aligned}$$

Since many options need to be calculated in a real-world scenario, a natural parallelism arises in computing these options in parallel.

III. Multi and manycore algorithms

The problem of solving the explicit and implicit time-marching is done using stencil operations in the explicit case and using the Thomas algorithm for solving the tridiagonal system of equations arising in the implicit case. In depth details and comparison of the CPU and GPU implementations can be found in [1].

A. Stencil operations for explicit time-marching

The one dimensional, stencil-based computations required for the one-factor application can be efficiently implemented on both CPUs and GPUs.

The system specification of the CPU system used in our work consists of a two socket Intel Xeon E5-2690 (Sandy Bridge) 8 core/socket server processors. Each core has 32KB L1 and 256KB L2 cache and all the cores in a socket share a 20MB of L3 shared cache. Each socket has a 51.2GB/s bandwidth to RAM memory and a total 66GB/s bandwidth is measured for the two sockets by the STREAM benchmark [7]. The theoretical maximal floating point computational limit of a single socket is 318 GFLOPS for single and 171 GFLOPS for double precision. The total maximal dissipated power of the two socket is 270 W.

The GPU used in the current comparison is an NVIDIA Tesla K40 card with the GK110b micro architecture. The theoretical maximal bandwidth of the system towards the main memory is 288 GB/s. The maximum bandwidth achieved with the CUDA toolkits memory bandwidth test program is 229 GB/s. The theoretical maximal floating point computational limit 5.04 TFLOPS for single and 1.68 TFLOPS for double precision. The total maximal dissipated power of the GPU card is 235 W.

a) CPU implementation: Stencil based computations can be efficiently implemented on CPUs equipped with vector instructions sets such as AVX or AVX2. Due to the efficient and large L1 cache (32KB) these problems on CPUs tend to be compute limited rather than bandwidth limited. On a typical CPU implementation the computations are parallelized over the set of options. Smaller subsets of options are solved using multithreading with OpenMP and options within a subset are solved within the lanes of CPU vectors.

b) GPU implementation: GPUs, due to the implemented thread level parallelism and compile-time register allocation give more freedom for better vectorization and optimization and therefore the achievable computational efficiency is very high. Among the many possible efficient algorithms discussed in [1] the best performing utilizes the new register shuffle instructions to share the work-load of computing a single timestep. The shuffle instruction (introduced in the Kepler architecture) makes communication possible between the threads inside a warp, ie. shuffle allows data to be passed between lanes (threads) in a vector (warp).

B. Thomas algorithm for implicit time-marching

The solution of the implicit form of the discretized BS equation requires the solution of a tridiagonal system of equations. The solution of the system is essentially the Gauss-Jordan elimination process of the matrix equation Eq. 4 with the Thomas algorithm shown in Alg. 1.

$$\begin{pmatrix} b_0 & c_0 & 0 & 0 & \cdots & 0 \\ a_1 & b_1 & c_1 & 0 & \cdots & 0 \\ 0 & a_2 & b_2 & c_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{K-1} & b_{K-1} \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{K-1} \end{pmatrix} = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ d_{K-1} \end{pmatrix} \tag{4}$$

a) CPU implementation: The CPU implementation relies on L1 cache performance and vectorization over options with either auto vectorization or explicit vectorization with instrinsic functions. Due to vectorization, 4 options with double precision or 8 options with single precision are solved in parallel with AVX or AVX2 instructions. The workload of the complete set of options is then parallelised using multithreading with OpenMP.

b) GPU implementation: Just as in the case of the explicit solver, the GPU solver allows for more optimization. The discussion of these novel, optimized algorithms can be found in [1]. Beside the Thomas and the PCR (Parallel Cyclic

ALGORITHM 1: Thomas algorithm

Require: $thomas(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d})$
$d_0^* := d_0 / b_0$
$c_0^* := c_0 / b_0$
for $i = 1, \ldots, K-1$ **do**
  $r := 1 \,/\, (b_i - a_i \, c_{i-1}^*)$
  $d_i^* := r \, (d_i - a_i \, d_{i-1}^*)$
  $c_i^* := r \, c_i$
end for
$u_{K-1} := d_{K-1}^*$
for $i = K-2, \ldots, 0$ **do**
  $u_i := d_i^* - c_i^* \, u_{i+1}$
end for
return $\mathbf{u}$

Reduction) algorithms, a new hybrid Thomas/PCR algorithm is also introduced in [1]. The efficiency of the latter algorithm relies on the aforementioned register shuffle instruction and compile-time register allocation. The algorithm works on a work-sharing basis in three steps: 1) the system of size N is split into 32 pieces which are partially solved and results in a reduced system of size 64; 2) threads cooperate to solve the 64 size system with PCR; 3) the solution of the reduced system is used to solve the partially computed systems in step 1).

IV. FPGA implementation with Vivado HLS

For many years there has been no breakthrough for FPGAs in the field of HPC (High Performance Computing). Over the years, research has been carried out to create tools to support development for FPGAs in C language. Recently syntheser tools with OpenCL (Open Computing Language) support appeared for Altera and Xilinx FPGAs. Xilinx in its Vivado HLS (High Level Synthesis) suite started a new software-based syntheser for the C,C++ and System C languages. These tools appeared as a response to the need of faster system realization. The Vivado HLS support for C with its customized support for the Xilinx FPGA is a potentially efficient solution.

A. Stencil operations for explicit time-marching

Stencil operations are a widely studied area of FPGA algorithms, see [8]. Many signal and image processing algorithms implemented on FPGAs utilize similar solutions to that used in explicit solution of one dimensional PDEs with stencil operations.

One of the key optimizations in FPGA circuit design is the maximization of the utilization of processing elements. This involves careful implementation that allow for pipelining computations to keep the largest possible hardware area busy. In HPC terminology this optimization is similar to cache-blocking, although the problem is not the data movement but rather keeping the system busy.

One way to create such a (systolic) circuitry is to create multiple interleaved processing elements with simple structure to allow low latency and high throughput. Each processing element is capable of handling the computation associated with three neighboring elements $f(u_{k-1}^{(n)}, u_k^{(n)}, u_{k+1}^{(n)})$ within a single timestep n, where f is the stencil operation. One may stack a number of such processing elements to perform consecutive timestep computations by feeding the result of processor $p = 1$ to $p = 2$ as shown on Figure 1. The result $u_{k+1}^{(1)}$ of $f(u_k^{(0)}, u_{k+1}^{(0)}, u_{k+2}^{(0)})$ is fed into the third input of the element processing element $p = 2$. This way the processing elements stay busy until they reach the end of the system. There are a number of warm-up cycles until all the processors get the data on which they can operate. This introduces an insignificant run-up delay if the size of the system to be calculated is larger than the number of the processors. Each processing element needs 2 clock cycles to feed in the necessary 2 elements in their FIFO. For the third clock cycle the third element is also available and the computation can be executed. Since there are P number of processing elements that need to be initialized to perform the lock-step time iteration on a given system, the total delay of the system is $2P$, ie. the Pth processor starts executing a stencil operation after $2 * P$ clock cycles of the start of the simulation.

TABLE I. FPGA Resource statistics for a single processor

	# BRAM		# DSP slice		Clock [ns]		$\times 10^6$ Ticks	
	SP	DP	SP	DP	SP	DP	SP	DP
Explicit	24	64	1200	3570	4.09	4.01	334	315
Implicit	256	512	37	94	4.26	4.3	14.2	12.6

Note: SP - Single Precision, DP - Double Precision

Larger units, called processors are composed of 80 and 85 processing elements in the case of single and double precision solutions. Resource requirements of these processors are shown in Table I. For single and double precision respectively 3 and 1 such processors can be crammed onto the FPGA used in the study.

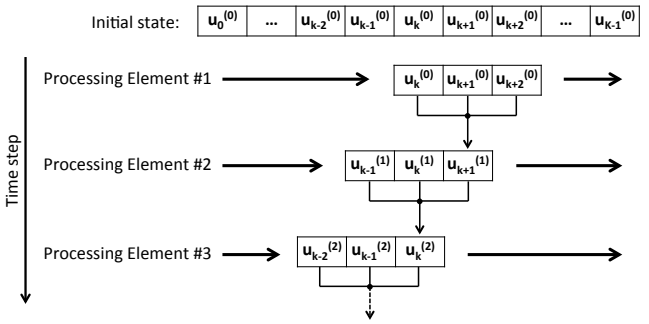

Fig. 1. Stacked FPGA processing elements to pipeline stencil operations in a systolic fashion. The initial system variables $u_0^{(0)}, \ldots, u_{K-1}^{(0)}$ are swept by the first processing element. The result $u_{k+1}^{(1)}$ is fed into the second processing element.

B. Thomas algorithm for implicit time-marching

The optimization of the implicit solver relies on the principle of creating independent processors to perform the calculation of independent options. Each of these processors is capable of pipelining the computation of more options into the same processor with the associated cost of storing the temporary (c^*, d^*) arrays of each option. The number of options that can be pipelined is defined by the depth of the forward sweep of the Thomas algorithm, which is 67 clock cycles in the present case. The temporary storage is implemented in the available Block RAM memories, but due

TABLE II. PERFORMANCE - SINGLE PRECISION

	ps/element			GFLOPS			GFLOPS/W		
	C	G	F	C	G	F	C	G	F
Explicit	15.2	2	17.4	394	3029	344	1.46	12.9	8.6
Implicit	142.7	14.5	766	139	1849	26	0.51	7.9	0.65

Note: C - 2 Xeon CPUs, G - Tesla K40 GPU, F - Virtex 7 FPGA

TABLE III. PERFORMANCE - DOUBLE PRECISION

	ps/element			GFLOPS			GFLOPS/W		
	C	G	F	C	G	F	C	G	F
Explicit	29.8	4.1	48.2	201	1463	124	0.74	6.2	3.1
Implicit	358.8	43.5	1748	48	892	9.8	0.18	3.8	0.24

Note: C - 2 Xeon CPUs, G - Tesla K40 GPU, F - Virtex 7 FPGA

to the deep pipeline the BRAM memory requirement limits the number of deployable processors.

The current HLS compiler fails to recognize that no data hazard exist between c_i^* and c_{i-1}^* in the two consecutive iterations of the forward pass in the tridiagonal algorithm and therefore refuses to properly pipeline the loop. A secondary temporary array is used to store a copy of the temporary arrays – this changes the data flow and guides the compiler towards pipelining.

A single processor unit for the implicit solver has resource requirements specified on Table I. For single and double precision respectively 11 and 5 of these units can be crammed onto the utilized FPGA.

V. PERFORMANCE COMPARISON

FPGA performance is compared to highly optimized CPU and GPU code. Estimations based on the Xilinx Vivado toolset are made to predict the achievable clock frequency on a Xilinx Virtex 7 VX690T. Performance metric is calculated on a grid element basis to eliminate the effects of architectural differences, ie. the total execution time is divided by the number of system elements K and total N time steps made. To mimic a real-world scenario the a_k, b_k, c_k coefficients of the explicit and implicit methods are set up in the first phase of the computation and stored in $\mathbf{a}, \mathbf{b}, \mathbf{c}$ arrays for each option independently. Every implementation uses these arrays to perform the computation.

The chosen FPGA is equiped with 108k slices, 3600 DPS slices and 3000 BRAM of size 18Kb. The total maximal power dissipation allowed by the packaging is expected to be less then 40 W.

The results of measurements are present on Table II and III. Based on the figures it can be stated that the proposed FPGA implementation is slower than the highly optimized CPU implementation, but it is significantly more power efficient. Compared to the GPU version the FPGA is significantly slower and even if the power dissipation of the GPU is higher than the FPGA the overall power efficiency is higher for the sake of the GPU.

VI. CONCLUSION

In the current paper a well studied finite difference problem – the solution of the Black-Scholes PDE – is chosen to compare novel CPU, GPU and FPGA architectures. Efficient FPGA based implementation of the explicit and implicit BS solvers have been created using Vivado HLS. The relatively low programming effort and the achieved efficiency of the resulting circuitry shows a promising step towards the applicability of FPGAs in an HPC environment and more specifically in finite difference calculations. Although the overall performance is not significantly higher than the CPU, the higher power efficiency makes this approach viable in power constrained system and solutions. The results on the other hand clearly show the superior performance of GPUs both in terms of computational efficiency and power efficiency.

ACKNOWLEDGMENT

The research has been supported by the TÁMOP-4.2.1./B-11/2/KMR-2011-002, TÁMOP - 4.2.2./B-10/1-2010-0014 projects at PPCU - University of National Excellence. The research at the Oxford e-Research Centre has been partially supported by the ASEArch project on Algorithms and Software for Emerging Architectures, funded by the UK Engineering and Physical Sciences Research Council. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work.

REFERENCES

[1] M. Giles, E. László, I. Reguly, J. Appleyard, and J. Demouth, "Gpu implementation of finite difference solvers," in *Proceedings of the 7th Workshop on High Performance Computational Finance*, ser. WHPCF '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/WHPCF.2014.10

[2] Q. Jin, D. Thomas, and W. Luk, "Exploring reconfigurable architectures for explicit finite difference option pricing models," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, Aug 2009, pp. 73–78.

[3] G. Chatziparaskevas, A. Brokalakis, and I. Papaefstathiou, "An FPGA-based parallel processor for Black-Scholes option pricing using finite differences schemes," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 709–714.

[4] T. Becker, Q. Jin, W. Luk, and S. Weston, "Dynamic constant reconfiguration for explicit finite difference option pricing," in *Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on*, Nov 2011, pp. 176–181.

[5] S. Palmer and D. Thomas, "Accelerating implicit finite difference schemes using a hardware optimised implementation of the thomas algorithm for fpgas," *arXiv preprint arXiv:1402.5094*, 2014.

[6] F. Black and M. S. Scholes, "The Pricing of Options and Corporate Liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–54, May-June 1973. [Online]. Available: http://ideas.repec.org/a/ucp/jpolec/v81y1973i3p637-54.html

[7] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995.

[8] Z. Nagy, Z. Vörösházi, and P. Szolgay, "Emulated Digital CNN-UM Solution of Partial Differential Equations: Research Articles," *Int. J. Circuit Theory Appl.*, vol. 34, no. 4, pp. 445–470, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1002/cta.v34:4