Speed-Up Computational Finance Simulations with OpenCL on Intel Xeon Phi

Michail Papadimitriou[1], Joris Cramwinckel[2], and Ana Lucia Varbanescu[3(✉)]

[1] Delft University of Technology, Delft, The Netherlands
m.papadimitriou@student.tudelft.nl
[2] Ortec Finance, Rotterdam, The Netherlands
joris.cramwinckel@ortec-finance.com
[3] University of Amsterdam, Amsterdam, The Netherlands
a.l.varbanescu@uva.nl

Abstract. Computational finance is a domain where performance is in high demand. In this work, we investigate the suitability of Intel Xeon Phi for computational finance simulations. Specifically, we use a scenario based ALM (Asset Liability Management) model and propose a novel OpenCL implementation for Xeon Phi. To further improve the performance of the application, we apply several optimization techniques (data layout and data locality improvement, loop unrolling) and study their effects. Our results show that the optimized OpenCL code deployed on the Phi can run up to 135x faster than the original scalar code running on an Intel i7 GPP. Furthermore, we also show that choosing the optimal work-item/work-group distribution has a compelling effect on massively parallel and heavily-branching code. Overall, these results are significant for the computational finance specialists, as they enable a major increase in model accuracy, because 10x more simulations can be performed in less than a 10th of the original time.

Keywords: OpenCL · Computing · Accelerated architectures · Intel Xeon Phi · MIC · GPGPU · Parallel computing · Asset Liability Management

1 Introduction

Modern applications targeting the finance industry become popular candidates for using high performance computing (HPC) platforms and techniques. Almost 10% of the TOP500 supercomputers, are dedicated for computational finance purposes [5]. This trend occurs because of the nature of applications that the financial sector has to offer and the increasing amount of data related to these applications. Examples of such applications are stock market data streaming, option pricing, high frequency trading, or risk management. They are loosely clustered in the fast-growing field of computational finance.

A computational finance instrument is the OPAL platform offered by Ortec-Finance, which provides goal based financial planning for private investors. The

© Springer International Publishing AG 2017
F. Desprez et al. (Eds.): Euro-Par 2016 Workshops, LNCS 10104, pp. 199–208, 2017.
DOI: 10.1007/978-3-319-58943-5_16

feasibility of potential goals is estimated based on high-quality scenario projections. These projections are influenced by investment decisions, market changes, clients financial situation and future goals [15]. Therefore, being able to increase efficiently the number of projections of the future can result into a more accurate investment plan. For this work, we have extracted from OPAL a test case of Asset Liability Management (ALM), to investigate the potential performance and/or accuracy increase when utilizing HPC platforms such as Intel Xeon Phi. ALM was chosen because it can have several applications within the finance industry such as risk management and it's need to comply with regulations. Typical regulations are Solvency II[1] and MiFID[2].

In general, the vast majority of accelerated applications from computational finance are using GPUs [2,6,11] and highly parallelizable (Monte Carlo and PDEs) methods [4,17]. Because of the prevalence of GPUs, some areas of computational finance, such as risk management, are less likely to be accelerated, as they contain extensive branching.

To address performance, OpenCL solution of the extracted test case was implemented. Then, a series of optimizations were applied for increasing its potential performance. As this model works with several conditional statements, GPU implementation approach can be very challenging. Therefore, Intel Xeon Phi due to it's CPU like behaviour, was chosen as the implementation platform. The performance on the Intel Xeon Phi was evaluated, as well as the individual effect for each of the optimization. In addition, solutions scalability was studied to determine the correlation between the effective speed-up and the number of future projections (scenarios).

Our results show that there is an great improve in performance which varied from x17 to x135 depending on the number of future projections. In addition, we studied the optimizations that lead to these speed-ups and their contributions to this performance.

The main contributions of our work are as follows:

- We chose a case study extracted from the financial sector industry, where improve in performance is in high demand.
- We propose a novel OpenCL implementation of the chosen case study.
- We applied various optimizations on the OpenCL implementation.
- We evaluate its performance on Intel Xeon Phi co-processor and the effect of the individual optimizations.

The rest of the paper is organized as follows: Sect. 2 provides the necessary background information on the test case model, programming language and development platform. Section 3 introduces the model, the OpenCL implementation and the different optimizations techniques used. Section 4, presents the obtained results on the Intel Xeon Phi along with the individual effect of each optimization. Finally, conclusion and future work is featured in Sect. 5.

[1] Solvency II is a new regulatory framework for insurance companies.
[2] MiFID is a directive that ensures investors protection in financial instruments, such as bonds, shares and derivatives.

2 Background

This section contains a brief introduction to Scenario based ALM case used, the OpenCL programming language and the Intel Xeon Phi which is proposed implementation platform.

2.1 Scenario Based ALM

The private investor has to make a decision about investments and chose an optimum investment strategy. The investment strategy usually lies between the balance of risk and reward. It is a plan of attack based on individual goals, risk tolerance, future capital needs and potential hazards [12]. In addition, these investments strategies taking in account asset allocation, buy and sell guidelines and risk guidelines. Therefore, the combination of this factors leads to changes in the chosen investment strategy.

An analysis using various different economic scenarios is crucial to get an accurate insight in risk and return. Thus, simulation techniques are clearly favored above analytical formulas here, because simulation can take into account a multitude of different variables, such as deposits, withdrawals, taxes, inflation, etc., and do so across a range of investment strategies and portfolios. Therefore, scenario based analysis instead of predicting the economic future, tries to assemble as realistic as possible projections of it.

Consider, for example, a typical pension fund case where 10,000 real world scenarios with a horizon of 64 years (monthly frequency), then 768000 evaluations are required in total. Assuming that this computation is the most computational intensive part of a larger process pipeline (scenario generation, pattern extraction etc.), it can take up to several minutes for completion. Therefore, the number of scenarios is the primary constraint for future development of the model and its accuracy.

2.2 The OpenCL Programing Lanaguage

OpenCL (Open Computing Language) is a framework which allows the composition of programs aiming for heterogeneous platforms. These platforms can consist of CPUs, GPUs, FPGAs, DSPs and other hardware such as co-processors (Intel Xeon Phi, Cell) [7].

In the early stages of development, OpenCL was initially a side project of Apple Inc. Later, Khronos Compute Working Group consisting of CPU, GPU, embedded-processor and other vendors. Therefore, in the OpenCL Ecosystem hardware (IBM, AMD, Intel, ARM, NVIDIA, ALTERA, XILINX) and software (codeplay, Sony, vmware, Adobe) dedicated members can be found. Finally, in 2008 an approved technical specification was released [7].

Figure 1 represents an overview of the OpenCL architecture. There is a host device which is able to control more than one of Compute Devices. For instance, these Compute devices can be either CPUs or GPUs. Each of these devices

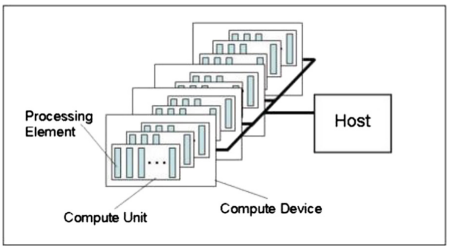

Fig. 1. Opencl architecture overview [7]

contains several Compute Units such as cores. Eventually, every Processing Unit contains several Processing Elements which execute the OpenCL kernels.

One of the greatest advantages of OpenCL is portability. Although, even with the code to be highly portable, the performance is not working in the same manner. Therefore, with OpenCL code which is cross-platform executable, unique optimizations need to performed for each platform.

2.3 Intel Xeon Phi Co-processor

The Intel Xeon Phi co-processor [9,10], is equipped with 60 general purposed cores. These cores are connected with a high speed bidirectional ring. Also, the cores are based on an updated Intel Pentium architecture (P54C), enhanced with 64-bit instructions and 512-bit vector instructions. These instructions are able to perform 16 single-precision operations or 8 double precision operations per instruction. In addition, the co-processor contains two levels of cache memory. The cache structure corresponds to a 32KB L1 for data, 32KB L1 for instructions and a 512KB L2 cache for every core [3]. The co-processor is able to provide 1.1 Tflops and 2.1 TFlops, peak performance for double and single precision operations, respectively. Additional features of the co-processor are the PCI express system interface, the 16 memory channels that it offers and it's Linux based micro OS. Also, The it offers two main modes, where applications can run on either native or offload mode. This allows application to run independently on the device or offloading highly computational and parallel parts from the CPU.

In terms of programming, Intel Xeon Phi offers a broad range of tools and programming tools, very similar to the ones available for a regular CPU [18]. In more detail, OpenCL [20], OpenMP [14], Intel Cilk Plus [16], Pthreads [13] and specialized math libraries like Intel Math Kernel Library [1] are available.

3 OpenCL Implementation

In this section, the Scenario based ALM, extracted from OPAL, is presented. Also, the proposed OpenCL implementation is outlined, along with the individual optimizations that applied.

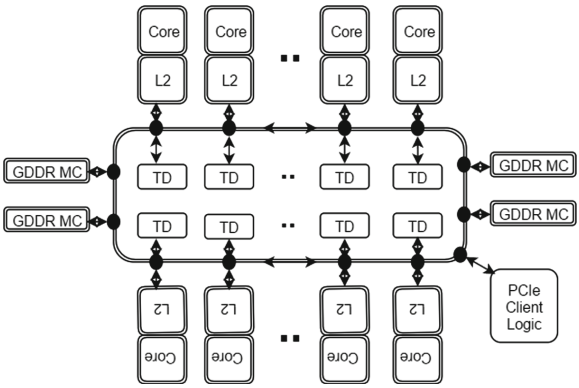

Fig. 2. Intel Xeon Phi architecture

3.1 Scenario Based ALM

The Scenario based ALM kernel, is a part of a larger process pipeline (scenario generation, statistical interpretation etc.), but still the most computationally expensive one. As it can be seen from Algorithm 1, the given application allows a level of parallelism among the different scenarios. Each scenario, has zero inference with the rest and therefore provide us with an initial degree of parallelization freedom.

Each scenario performs a number of computations for a given portfolio. Each portfolio can contain several assets (up to 20). Usually, these assets represent cash, bonds, stocks and equities from different regions (UK, US, JPN). Also, as each scenario needs to comply with real world financial needs such as taxation and rebalancing of the capital between the assets, extensive branching is present in that kernel. Eventually, the value of each asset of each portfolio and the level of taxes needs to be recorded at every iteration of every scenario.

$$PortfolioValue = \sum_{i=1}^{n} = scenAssetweight(i) * currentAssetValue(i) \quad (1)$$

Equation 1, represents how the total value of each portfolio is calculated. The value of each is multiplied by a weight correspond to the current iteration of the current scenario. Therefore, it contributes on increasing significantly the number of accesses to global memory. Each weight is different for each scenario as it is related to a different projection of the economy and a different financial decision.

3.2 OpenCL Implementation

The Scenario based ALM model, presented in Sect. 3.1 is implemented as a single kernel. Each individual scenario is simulated by a work item, in 1D work groups. OpenCL allows the compilation of kernels to take place during the execution time. Therefore, a very large part of the parameters can be passed as preprocessed constants and save resources from parameter passing. Under this

```
Input: Scenarios, Years, Months, portA, portB, portC, portD
Output: totalValue, valueA, valueB, cvalueC, valueD, valueTax
for Number of Scenarios do
    for Number of Years do
        for Twelve Months do
            for each Portfolio do
                for each Asset do
                    Calculate new value;
                end
                Sum of Assets value;
            end
            if Month is December then
                Calculate amount of taxes;
            end
            Store Current Value of each Portfolio and each Asset;
            Store Tax Value;
            Store Total Value of Portfolios;
        end
    end
end
return totalValue, valueA, valueB, cvalueC, valueD, valueTax
```

Algorithm 1. Abstract representation of Scenario based ALM

structure all of the required constants by the model can be passed at a minimum cost. For this first OpenCL implementation, we tried to keep as simple as possible, without utilizing specific hardware or OpenCL features.

3.3 Optimizations

For increasing the performance of the proposed OpenCL implementation, a selection of four different optimizations were applied. By experimenting with this optimization space, some key observations were made regarding the effect and the possible improvement in terms of performance.

Workgroup Configuration. For any OpenCL kernel, the recommend work group size should be equal to the SIMD width. Therefore, for Intel Xeon Phi and float data type, the kernel width should be in multiplies of 16. This structure exploits the auto vectorization module in an optimum way while for non multiplies of 16, the items are packaged in a traditional scalar way [21].

Compiler Optimizations. In most GPGPU architectures, several hardware specific optimizations are available by the compiler. These optimizations may have the form of specific "expensive" mathematical functions such as square roots. In the same manner OpenCL allows a certain number of such flags for allowing better exploitation of the hardware. The optimizations chosen relevant to the nature of the model where *-cl-fast-relaxed-math*, *-cl-no-signed-zeros* and *-cl-denorms-are-zero*.

Data Layout. Data layout can have significant impact in an applications performance. Memory access patterns of the kernel can be converted from array of structures (AoS) to structure of arrays (SoA). This conversion results to a more cache friendly layout which can be benefited by the vectorization module. [22] Thus, the resulted performance can be improved with the used of a more SIMD friendly layout like the SoA [19]. The vectorization module transforms scalar data type operations on adjacent work-items into an equivalent vector operation. If vector operations already exist in the kernel source code, the module scalarizes (breaks into component operations) and revectorizes them.

Constant Memory. The use of constant memory can allow all compute units of the device to have access on the same data. Any constant memory element can be accessible on the same time by all work-items. Although, use of constant memory is strongly relative to the nature of the problem and work-group dimensions. Moreover, constant memory is expected to effect performance only for small problem sizes, where data can fit in the small constant memory.

4 Results

In this section, the results obtained after applying various optimizations will be presented. All the experiments performed in an Intel i7 GPP and Intel Xeon Phi co-processor.

4.1 Performance Impact of the Optimizations

In Table 1, the individual and relevant impact of each optimization are presented. The final performance yield a speed-up in magnitude of 109 times compared with our initial scalar implementation.

Initially, a naive OpenCL implementation was tested on the Phi. The out of the box performance was x21 faster than the original scalar code. This extend of improvement in performance was satisfying, but still not any specific architecture or programming features were exploit.

Further results demonstrate that while choosing the optimum workgroup configuration, the impact in performance can be significant. By tuning the application for a global size of 10240 over 1D range, demonstrate an extensive effect in performance. For the optimum work-group/work-item arrangement (128×80), the overall speed-up increases to x80.1, while the relative speed-up compared to the naive OpenCL solution increases by a factor of x3.8.

Enabling the compiler flags mentioned in Sect. 3.2, increase the relative speed-up by just x1.06. On the other hand, converting the data access patterns to structure of arrays (SoA) gives almost 20 times faster performance in comparison with the original version. In addition, using constant memory intead of global for the different work-items to have access to independent scenario weights, gives an additional x1.05 speed-up. Although, for larger number of scenarios (more than 10240), data cannot fit in constant memory.

Table 1. Single precision OpenCL implementation: speed-up and relative speed-up for various optimizations and input of 10240 scenarios

Version	Time [s]	Speed-up	rSpeed-up
Scalar	3.1245	1	-
Naive	0.1500	20.8	1
Workgroup dim	0.0390	80.1	3.8
Compiler flags	0.0368	85	1.06
SoA	0.0304	103	1.21
Constant memory	0.0287	109	1.05

4.2 Speed-Up Scalability

After evaluating the peak performance under a specific knob of optimizations, we evaluate the scalability of these results under different number of scenarios. For each number of scenarios, the optimum work group configuration was determined and used.

In Table 2, the results obtained from our novel OpenCL implementation compared to the scalar baseline are presented. These results provide us with enough information to evaluate the potential benefits of using Intel Xeon Phi. Firstly, we note that in 2/3 of the simulation time for 1024 scenarios, we were able to simulate 80 times more scenarios. In addition, we shown that for very large number of scenarios, we were able to achieve speed-ups, up to x135 compared to our scalar implementation running on a GPP.

Finally, we verified that for larger scenario inputs, we achieved the best performance while using work groups in multiplies of 16 [8]. This behaviour was due to the fact that SIMD, deploys the work-group items in groups of 16. On the other hand, for very small group scenarios, the work-group parallelism couldn't exploited in it's fullest potential and thus, smaller speed-ups were achieved.

Table 2. Single precision execution time results: Intel Xeon Phi vs Intel i7-5600U

Scenarios	Execution time (s)		Speed-up
	Intel i7-5600U	Intel Xeon Phi	
1024	0.2853	0.01719	x17
4096	1.1381	0.01339	x85
8192	2.1431	0.02547	x86
10240	3.1245	0.02873	x109
40960	9.8645	0.09031	x112
81920	26.06165	0.19205	x135

5 Conclusion and Future Work

Due to the continuous need for faster and more accurate models, the financial sector offers a broad range of applications in need for acceleration. Therefore, we chose a scenario based ALM application, where speed and increase in accuracy are in particular needs. We proposed a novel OpenCL implementation of the Scenario based ALM and we tested on Intel Xeon Phi co-processor. We evaluate its out of the box performance and the effects of different optimizations.

In general, we proved that utilizing Intel Xeon Phi and OpenCL for scenario based ALM simulations, can yield to significant improvements in performance (up to x135). Also, we clarify that for application in which extensive branching is present, Intel Xeon Phi expected to offer a more efficient solution compared to a GPU. In addition, we shown that when optimizations are applied, the out of the box performance can be increased up to four times.

In terms of future, work we are working on investigating the performance portability for our OpenCL scenario based ALM solution. This investigation will focus on the OpenCL portability among different platforms, as well as the individual effects of different optimizations in every platform. This study will aim to find a minimum set of optimization knobs, for which a certain level of performance can be kept among different platforms.

References

1. Intel Math Kernel Library: Reference Manual. Intel Corporation, Santa Clara (2009). ISBN 630813-054US
2. Cramwinckel, J., Singor, S., Varbanescu, A.L.: FiNS: a framework for accelerating nested simulations on heterogeneous platforms. In: Hunold, S., et al. (eds.) Euro-Par 2015. LNCS, vol. 9523, pp. 246–257. Springer, Cham (2015). doi:10.1007/978-3-319-27308-2_21
3. Fang, J., Sips, H., Zhang, L., Xu, C., Che, Y., Varbanescu, A.L.: Test-driving intel xeon phi. In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ICPE 2014, pp. 137–148. ACM, New York (2014). http://doi.acm.org/10.1145/2568088.2576799
4. Gaikwad, A., Toke, I.M.: Parallel iterative linear solvers on GPU: a financial engineering case. In: 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, pp. 607–614, February 2010
5. Giles, M.: From CFD to computational finance and back again, November 2009. https://people.maths.ox.ac.uk/gilesm/talks/princeton.pdf
6. Giles, M., Lszl, E., Reguly, I., Appleyard, J., Demouth, J.: GPU implementation of finite difference solvers. In: 2014 Seventh Workshop on High Performance Computational Finance (WHPCF), pp. 1–8, November 2014
7. group, K.: The open standard for parallel programming of heterogeneous systems, January 2016. https://www.khronos.org/opencl/
8. Intel: Work-group size considerations for intel xeon phi coprocessors (2015). https://software.intel.com/en-us/node/540512
9. Intel: Intel xeon phi co-processor. April 2016. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

10. Jeffers, J., Reinders, J.: Intel Xeon Phi Coprocessor High Performance Programming, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (2013)
11. Liu, R.S., Tsai, Y.C., Yang, C.L.: Parallelization and characterization of garch option pricing on GPUS. In: 2010 IEEE International Symposium on Workload Characterization (IISWC), pp. 1–10, December 2010
12. Dempster, M.A.H., Medova, E.A..: Asset liability management for individual households. Br. Actuar. J. 405–439 (2011)
13. Nichols, B., Buttlar, D., Farrell, J.P.: Pthreads Programming. O'Reilly & Associates Inc., Sebastopol (1996)
14. OpenMP Architecture Review Board: OpenMP application program interface version 3.0, May 2008. http://www.openmp.org/mp-documents/spec30.pdf
15. Ortec-Finance: Goal-based financial planning, April 2016. http://www.ortec-finance.com/Private-Wealth/Online-Financial-Services.aspx
16. Robison, A.D.: Composable parallel patterns with intel cilk plus. Comput. Sci. Eng. **15**(2), 66–71 (2013)
17. Rocki, K., Suda, R.: Large-scale parallel monte carlo tree search on GPU. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum (IPDPSW), pp. 2034–2037, May 2011
18. Heinecke, A., Pflüger, D., Budnikov, D., Klemm, M., Narkis, A., Shevtsov, M., Zaks, A., Lyalin, S.: Demonstrating performance portability of a custom opencl data mining application to the intel r xeon phi (2013). http://dx.doi.org/10.13140/2.1.4212.6084
19. Smelyanskiy, M., Sewall, J., Kalamkar, D.D., Satish, N., Dubey, P., Astafiev, N., Burylov, I., Nikolaev, A., Maidanov, S., Li, S., Kulkarni, S., Finan, C.H., Gonina, E.: Analysis and optimization of financial analytics benchmark on modern multi- and many-core IA-based architectures. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, pp. 1154–1162 (2012). http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6495921
20. Stone, J.E., Gohara, D., Shi, G.: Opencl a parallel programming standard for heterogeneous computing systems. IEEE Des. Test **12**(3), 66–73 (2010). http://dx.doi.org/10.1109/MCSE.2010.69
21. Tian, X., Saito, H., Preis, S.V., Garcia, E.N., Kozhukhov, S.S., Masten, M., Cherkasov, A.G., Panchenko, N.: Practical SIMD vectorization techniques for intel® xeon phi coprocessors. In: Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and Ph.D. Forum, IPDPSW 2013, pp. 1149–1158 (2013). http://dx.doi.org/10.1109/IPDPSW.2013.245
22. Zhang, Y., Sinclair, M., Chien, A.A.: Improving performance portability in OpenCL programs. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) ISC 2013. LNCS, vol. 7905, pp. 136–150. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38750-0_11