FiNS: A Framework for Accelerating Nested Simulations on Heterogeneous Platforms

Joris Cramwinckel[1](✉), Stefan Singor[1,2], and Ana Lucia Varbanescu[3]

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

Abstract. Insurers use advanced risk management models to, among other things, compute required capital for different sources of financial risks. In these models the application of nested simulations becomes increasingly important. To keep computation times within acceptable limits high-performance computing is required. In this work we present a framework designed to significantly improve the performance of nested simulations by using heterogeneous computing. Specifically, we use modern features from CUDA - streams, Hyper-Q, and Multi-Process Service - to take full advantage of the massive parallelism of modern GPUs. We manage to reduce the execution time of such simulations from several hours to tens of minutes.

Keywords: CPU-GPU heterogeneous computing · Asset & Liability Management · Nested simulations · CUDA Streams · CUDA Hyper-Q

1 Introduction

Insurance companies sell products like variable annuities and universal life insurance that include certain rights, such as guarantees and profit sharing. These rights may bring profit to the policy holders but cannot cause a loss. The valuation of these embedded options in insurance contracts is quite challenging, because insurance companies need to value their embedded options for many applications - e.g., Solvency II[1] reporting, monitoring, product pricing and Asset &Liability Management (ALM). For these various applications, one would ideally use the same valuation method in order to maintain consistency, transparency, and ease of interpretation. The preferred valuation method nowadays is risk neutral Monte Carlo simulation [7].

Determining the current ($t = 0$) market value of the embedded option is generally not a problem, since we can gather relevant market data, calibrate risk

[1] Solvency II is a new regulatory framework for insurance companies that officially starts as of January 1, 2016.

S. Hunold et al. (Eds.): Euro-Par 2015 Workshops, LNCS 9523, pp. 246–257, 2015.
DOI: 10.1007/978-3-319-27308-2_21

neutral models and perform Monte Carlo valuations to obtain option values. Performing these steps in a scenario simulation context for future time periods ($t > 0$) is more complicated. These so-called nested simulation valuations are, amongst others, required for computing the Solvency Capital Requirement (SCR) based on a 99.5 % Value at Risk (VaR) on a 1-year period[2], the Own Risk and Solvency Assessment, and ALM. Additionally, complementary models for calibration and validation purposes also use nested simulations.

Most nested simulation applications found in financial applications can easily run for days, an obvious bottleneck to their viability. Furthermore, these long-running simulations discourage any research on new models and new methodologies.

Our aim in this work is to significantly improve the performance of nested simulations, making them feasible for both production and more empirically-driven research. Given that Graphical Processing Units (GPUs) are a proven technology in finance for performing Monte Carlo valuation simulations ([1,10]), we aim to make use of these massively parallel architectures to accelerate financial nested simulations. The main challenge here is efficiency, because the multiple layers of parallelism of nested simulations require a tight collaboration of the CPU and the GPU.

Our work introduces a *Financial Nested Simulations* (FiNS) framework, i.e. a CPU-GPU heterogeneous solution for improving the performance of nested simulations in financial applications. FiNS is driven by two important requirements: performance improvement and ease-of-use for financial specialists.

To address performance, FiNS makes extensive use of a set of advanced CUDA abstractions available in the latest NVIDIA architectures (Kepler and newer): CUDA streams [11], Hyper-Q [13], and MPS [14] are all used to efficiently offload simulations to the GPU. To address usability, FiNS is built as a skeleton that can be easily adapted to different applications.

To demonstrate both the performance and usability of FiNS, we build a mock-up model of an existing ALM tool, which emulates the behavior of a full nested simulation. An ALM tool uses many macroeconomic scenarios to provide users insight in future performance of, for example, an insures' or pension funds' balance sheet.

Our results show significant performance improvement over the sequential code, with speed-ups ranging between 26 and 6 for light and heavy cases, respectively. This significant gain is due to our efficient use of both the CPU and the GPU. Although the reference sequential code is by no means optimized, FiNS brings a significant improvement in the way nested simulations can be used in production and research.

Summarizing, the main contribution of this work is threefold:

1. We propose an original way to exploit streams for increasing the efficiency of heterogeneous CPU-GPU platforms in the case of applications with multiple layers of moderate parallelism;

[2] This represents the amount of capital the insurer must hold against unforeseen losses during a one-year period.

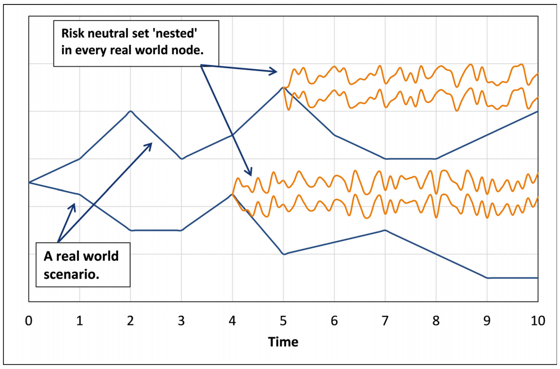

Fig. 1. Risk neutral simulations in a real world simulation. The figure shows two real world scenarios starting at t=0. For each real world scenario, a single nested simulation with two risk neutral scenarios is shown (at $t = 4$ and $t = 5$). In practice, thousands of risk neutral scenarios are used for valuation in each time step of a real world scenario.

2. We design FiNS, a generic framework for using heterogeneous platforms in financial nested simulations;
3. We demonstrate how FiNS can be used to accelerate ALM, a specific case of nested simulation used in risk management for institutional investors like insurance companies or pension funds.

2 Financial Background

For risk management, an insurer wishes to compute all relevant balance sheet components at each time step of a real world scenario. One generally starts with generating real world scenarios (the outer scenarios) for $t \geq 0$. To obtain the value of the embedded options at each time-step of a real world scenario, risk neutral scenarios (the inner scenarios) are required to perform a Monte Carlo valuation [7]. Within a Monte Carlo valuation the discounted pay-off cash flows resulting from the inner scenarios are averaged to obtain the option price. This concept of nested simulations is illustrated in Fig. 1.

While Monte Carlo valuation can easily be used for $t = 0$ applications, it is computationally expensive to apply it in a scenario environment. This problem emerges because Monte Carlo valuations need to be performed in each time step of a real world scenario.

Consider, for example, the case of $2,000$ real world scenarios with a horizon of 5 years (annual frequency), then $10,000 + 1$ (including $t = 0$) valuations are required in total. Assuming one valuation takes a few seconds[3], then the total computation time for valuation is ≈ 6 h, which is unacceptable in practice[4]. Therefore, high performance computing is required to reduce the overall computation time.

[3] As measured in production using sequential code on a state-of-the-art CPU.

[4] It should be noted that, in practice, multiple valuations are required at each time step for Solvency II and hedging. This results in even higher computation times.

3 GPU Background

GPUs[5] are massively parallel processing units, originally designed for graphics. A GPU has multiple streaming processors (SMs), each grouping tens of simple cores. With hundreds to thousands such cores, the performance to be achieved can easily reach a couple of TFLOPs *for applications with enough concurrency.* Additionally, GPUs have a layered memory system, including private memory per core, shared memory and L1 cache per SM, L2 cache and a global, off-chip memory. Memory bandwidth is typically significantly higher than for CPUs, but it is still the limiting factor for performance in many applications.

GPUs are working as accelerators - i.e., they are not stand-alone processing units, but require a *host* to manage their involvement in computation. Such host is typically a CPU; in such a CPU-GPU platform, the GPU is called a *device*. Note that the host and the device run separate codes: the host code is the main application from which parts are being offloaded for computation by kernels running on the device. Also note that the memory spaces of the CPU and GPU are separated, which means that any application that offloads computation kernels to the GPU might also need to copy data from host to device and/or the other way.

For programming these GPU kernels, the most popular solution is CUDA, a proprietary programming model from NVIDIA. While portable models like OpenCL and higher level models like OpenACC exist and can be successfully used for many applications, they are not suitable for this work because the special features we are using are not yet available in these models. We further describe these features in the paragraphs below.

3.1 CUDA Streams and Hyper-Q

A CUDA stream is an abstraction of a series of tasks run by the GPU. By tasks we mean (1) memory copies, (2) synchronization, and (3) kernels (i.e., computational tasks). The tasks in a single stream are ordered, but they are independent from tasks in different streams.

Using streams can improve the concurrency of an application. For example, within nested simulation we repeat a sequence of tasks for every outer scenario in every period. By embedding this sequence of tasks in a stream, and launching a new stream for each node in the outer simulation (as is displayed in Fig. 1), we have an elegant solution to launch multiple inner simulations that will not interfere with each other.

The most important features of streams necessary for this work are: first, stream launches can be asynchronous, allowing the CPU to compute while the GPU is running. Furthermore, tasks for different hardware engines within streams can run concurrently, i.e. computational tasks (kernels), executed by

[5] In this work we focus on NVIDIA GPUs and we make heavy use of CUDA concepts. In theory all other GPUs have the required features, yet programming them remains a challenge which is beyond the scope of this work.

the GPU SMs and memory copies (D2H and H2D), executed by the GPU copy engines. And finally, streams can run concurrently on the device. We further detail the way we exploit these features in the following paragraphs.

CPU-GPU Concurrency. Stream launches are asynchronous by default. To ensure a asynchronous launch, care must be taken that memory copies are initialized with the asynchronous API and that the host memory allocations are pinned [13]. If synchronization between device and host is required, this can be accomplished by either several CUDA synchronize methods or by implicit synchronization. Unintentionally synchronizing the streams on the device is the main difficulty of working with streams.

Compute and Memory Accesses Overlap. Within a stream, it is possible to overlap the memory transfers with computational work by invoking the tasks with the asynchronous API. If this is crucial for the application performance, one needs to chunk the work such that the overlap is optimal. Furthermore, memory copies of a stream can overlap the computations of another stream.

We note that, implicitly, this solution increases application parallelism by decreasing the granularity of the tasks and making use of the engine parallelism in the hardware platform.

Concurrent Kernel Execution. The latest developments in NVIDIA cards increase the concurrency possibilities of the streams within a work queue. This is accomplished by NVIDIAs Hyper-Q feature [13].

Concurrent kernel execution is strictly bound by computational capacity and the device architecture. The latter is at the time of writing a dominant factor. The Kepler (and newer) cards support Hyper-Queuing, which has an important effect on performance.

To illustrate this effect, we present in Figs. 2 and 3 a comparison of the Fermi and Kepler architectures for streams concurrency. The streams contain two kernels: a large `generatePaths` kernel followed by the tiny `priceOptions` kernel. In Fig. 2 we observe a two-way concurrency; because the final (tiny) task of a stream is running concurrently with the first (much larger) task of the subsequent stream. Hardware utilization in this case is low. On the contrary, Fig. 3 displays the benefits of the hyper-Q allowing higher concurrency between streams, resulting in higher hardware utilization.

3.2 Multi Processing Service

Another way to increase the utilization of GPUs is to share the device for kernels from different (local) processes. In order to manage GPU sharing between processes, we used NVIDIA's Multi Processing Service (MPS) [14]. This software layer provides a context manager to handle work launched from different processes. MPS is exclusively available on Linux and is only provided with

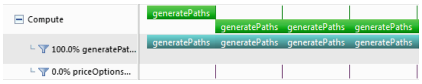

Fig. 2. Stream concurrency on fermi architecture

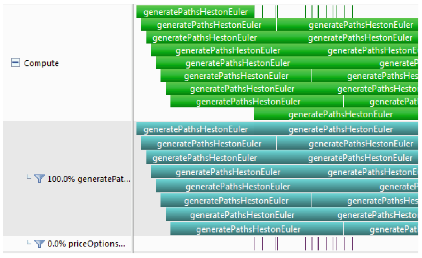

Fig. 3. Stream concurrency on kepler architecture

NVIDIA Tesla cards with compute capability 5 or higher. Although these restrictions limit applicability, it is a relatively cheap way to explore the concept of kernel offloading from multiple processes to a single GPU. This is an important feature when a single host process cannot generate sufficient work for the GPU, as it allows multiple cores or even multiple machines to collaborate in keeping a single device busy.

We note that previous work on offloading streams in a multi-threaded or multiprocessing environment [20] showed significant GPU utilization in benchmark cases. We implemented the same ideas as [20] for local Python processes. Section 5 provides more detail on how this feature affects our FiNS framework.

4 Framework Architecture

An ideal scenario is for the outer and inner scenarios to run in parallel. In this case, a perfect overlap provides optimal performance. Specifically, this means that within the duration of a real world simulation step (typically tens to hundreds of milliseconds), we must complete a full risk neutral simulation. This requirement demands a heterogeneous solution, which matches CPU + GPU architectures quite well, as seen in Fig. 7. Our work therefor focuses on building a framework that significantly outperforms existing solutions for nested simulations, but is flexible enough to support multiple types of such applications, where the analysis and end-results of the inner and outer simulations can vary in complexity.

The key to efficient heterogeneous programming is in designing the right solution to utilize the available hardware efficiently. In our case, the main challenge is GPU utilization: one inner simulation offloaded to the GPU can not, for most applications, fully utilize a GPU on its own. In order to increase GPU utilization multiple inner simulations will have to run concurrently. Without the concept of streams a custom implementation is needed, and it can become quite complex as one needs to build an aggregated kernel, as an artificial concatenation of kernels, which limits their flexibility in accepting different data sources or data types. When using CUDA streams these kernels can remain independent - thus flexible

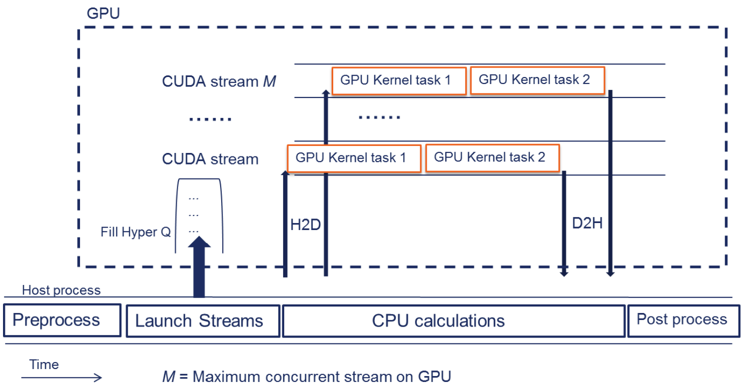

Fig. 4. Using streams in FiNS. The CPU and all GPU streams can run concurrently. The level of concurrency achieved in practice is determined by the actual hardware.

and fully reusable - and the task of concurrently executing them is offloaded to the device itself. From the perspective of flexibility this is an ideal solution for a generic framework, even if it comes with a small performance penalty (below 10 % according to our results [5]).

4.1 The Framework

FiNS is a development framework, that offers a skeleton-like infrastructure for the designers of nested simulation applications. Essentially, we provide a high-concurrency template that needs to be instantiated for a specific application. Using FiNS a developer needs to focus only on the implementation of the outer and inner simulation functionality. The framework will make sure that the mapping of these tasks on the real heterogeneous platform will be optimized for massive parallelism and efficient usage for both the CPU and the GPU.

To achieve this high level of flexibility we make use of CUDA streams, as seen in Fig. 4, displaying the concurrency between host and a number of streams running on the device. The CPU prepares the workload for the GPU and launches the work in streams to the device. The device receives the streams and stacks the work in a Hyper-Q. Note here that the issue order of the streams is not necessarily the order of execution, which clearly requires the computations in streams to be completely independent. If this is not the case, FiNS cannot be used. As mentioned earlier, it is important to have all host memory allocations page-locked or else the CPU-GPU concurrency will break when memory transfers are initialized. Furthermore, the Hyper-Q takes care of optimal hardware utilization by scheduling the streams concurrently. All streams are launched asynchronously, so that the GPU computations can overlap with the CPU tasks. These tasks can consist of, for example, calculating real-world simulation steps, calculating statistics or memory flushes to the hard drive.

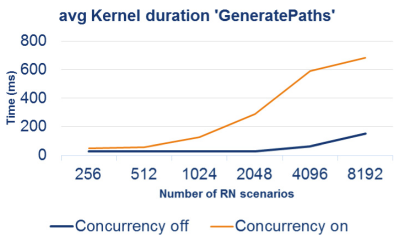

Fig. 5. Average kernel duration

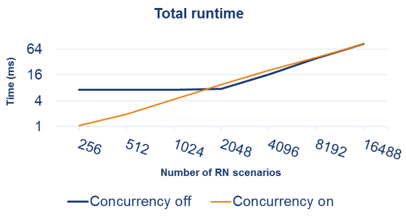

Fig. 6. Total runtime

4.2 Streams in Practice

To show the performance behavior of kernel concurrency with streams, we set up a sample workload of 256 streams for the `generatePaths` kernel. We evaluated different sizes of offloaded work, determined by the number of risk neutral scenarios. To understand the performance gain for concurrency, we launched the sample both with and without kernel concurrency. We observe that the average duration of the kernel increases when streams are running concurrently (Fig. 5). However, for a lower number of scenarios (till around 2,000), the concurrency contributes such that it is faster than the non-concurrent variant (Fig. 6). Moreover, for larger numbers of scenarios there is no performance penalty. This behavior indicates that the CUDA Work Scheduler assigns more resources to single sequential streams than it does to concurrent streams. As a result, kernel concurrency in streams only benefits performance when the offloaded kernels are not large enough to fully utilize the GPU, whereas for larger cases performance remains equal.

5 Nested Simulation for ALM Tooling: A Case Study

Section 2 already described the need for nested simulations in practice. Due to performance reasons the available ALM software is not equipped with the nested simulation features. Instead, we use analytical methodologies, often less accurate. This also means that we have no real reference code to compare against. Therefore, we build a mock-up model of an ALM tool. In this mock-up model outer simulations are emulated by a sleep statement. Since we are interested in the impact on the user-time performance (wall-clock performance) for this application, we assumed three benchmark cases. They differ in the duration of a real world simulation step per scenario per period. We assumed normal distributed duration with means 75, 150 and 300 ms for respectively a light, medium and a heavy case and a 5 ms standard deviation.

5.1 Application Description

Figure 7 displays the concept of offloading risk neutral calculations to the GPU in comparison with a sequential version. With this concept the goal is to perform

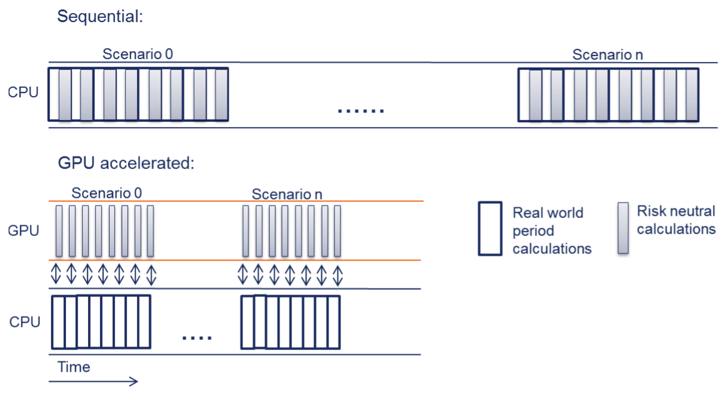

Fig. 7. Overlapping real and nested simulations. Note that all RNs are *different*, yet independent, thus they can be executed concurrently.

all risk neutral tasks within the runtime of a real world period. This is important since ALM models are used in a decision-making process. Any additional runtime on current methodologies is unwanted by its users.

In contrast, with Fig. 7 modern ALM tools perform real world simulations in a multiprocess or multi-threading setting. This makes it more challenging finishing the offloaded risk neutral simulation within the duration of its parent real world simulation step, since the GPU will be receiving tasks from all processes simultaneously.

5.2 Using the Framework

The framework presented in Sect. 4 serves as the greatest common divisor of several FiNS applications. For the ALM applications we extended the framework with multiprocessing support. This way, each host process is launching streams for its risk neutral calculation tasks to a daemon process hosted by the MPS Sect. 3.2. This daemon process manages all GPU requests from its slave processes and queues all received streams in a single hyper-Q. Using a single hyper-Q results in concurrent stream execution over the different processes.

5.3 Evaluation

Consider, again, the case of $2,000$ real world scenarios with a horizon of 5 years (annual frequency), which are common dimensions for an insurer's ALM study. In a single process setting, simulations for the benchmark cases take resp. 12.5, 25 and 50 min. Note that we assumed perfect performance scaling for higher numbers processes. We measured that a single inner simulation of $1,000$ with a horizon of 100 years and a $\frac{1}{120}$ time steps per year takes 1.875 s on a state-of-the-art CPU. These simulation dimensions are representative for current CPU models. Given that the inner simulations are run $10,000 + 1$ (including $t = 0$)

Table 1. Runtime full nested simulation on GPU framework

Case # Cores	Light	Medium	Heavy
	(in minutes)		
1	12.3	24.9	51.3
2	6.6	12.8	25.3
4	3.7	6.5	12.7
8	3.7	3.9	6.4
16	3.3	3.4	3.9

Table 2. User time speed ups *(maximum theoretical speed up)*

Case # Cores	Light	Medium	Heavy
	(26.0)	*(13.5)*	*(7.25)*
1	26.46	13.53	7.07
2	24.68	13.17	7.16
4	22.03	12.95	7.12
8	11.01	10.83	7.05
16	6.10	6.15	5.76

times for the complete run, the inner simulations represent a workload of over 5 h in a single process setting.

For evaluating the use case on FiNS we used a NVIDIA K20 GPU. Table 1 displays the measured runtime of the mockup model with the heterogeneous framework. We observe that for the lower number of cores the GPU is keeping up with the offloaded work. For a larger number of cores offloading work to the GPU, we observe that tasks stack in the hyper-Q and the CPU has to wait for the GPU results to be finished. Note that the single core results are below the theoretical reference for the Light and Medium case; this is caused by the assumption of normal distributed benchmark timings for the outer simulation.

The maximum speed up of the heterogeneous framework versus the theoretical sequential runtime is bound by the fraction of the tasks to be offloaded. Amdahl's law tell us that maximum achieved speed up is defined by $\frac{1}{B}$ where B represents the fraction of time the models is strictly serial. This results in theoretical maximum speedups of 26.0, 13.5 and 7.25 for all processes in resp. the Light, Medium and Heavy benchmark case. Table 2 displays the speedup of the GPU accelerated model versus the theoretical CPU runtime. The results show that for most of the cases a near to maximum speed up is reached.

The results in Table 1 indicate that, if resources can be scaled (usage of multiple GPUs), the proposed architecture would in theory be able to close in to the theoretical maximum speed up for every case defined in Table 2. Although scalability of the architecture is not implemented yet, we can conclude that the proposed architecture is most promising, since streams are easily distributable over multi GPUs and MPS has multi GPU support. To reveal the importance of NVIDIA MPS we also run the same tests with MPS disabled. We observed that speeds ups as displayed in Table 2 were up to 40 % lower.

6 Related Work

The utility of GPUs in Monte Carlo valuation methods is becoming a proven technology in finance [10]. Such work is focusing strictly on improving the performance of risk neutral simulation. In [1], a CPU-GPU performance comparison is made, achieving up to 10x speedup for both European and American contracts. Additionally, a lot of research has been dedicated to GPU-accelerated solutions for several risk neutral models [4,6,8,18,19]; the observed speedups range between 4 and 150. In our work, we focus on simulation models that require the cooperation of the CPU and the GPUs towards efficient nested simulations.

Financial nested simulations are increasingly important due to new regulations, so their performance becomes a production-level concern. Therefore, [3] describes several numerical methods for reducing the computational intensity of Solvency Capital Requirement (SCR) calculation, making it computationally feasible. Complementary to their study, our work demonstrates that the FiNS framework can render the same simulations feasible by using CPU-GPU heterogeneous computing.

Using heterogeneous computing for large scale simulations is already established as a feasible solution to improve performance for many classes of applications. Systems such as Glinda, Qilin, or Insieme [9,12,16,17] focus on static partitioning of one workload to multiple devices, under the assumption that the GPU is overloaded. Such systems are not suitable for our nested simulations, because in our scenarios the GPU is "underloaded". An alternative is to use a runtime-based system for heterogeneous computing, such as OmpSS or StarPU [2,15]. However, none of these approaches supports sharing devices by multiple kernels, which is an essential performance booster for FiNS.

7 Conclusion and Future Work

Nested simulation applications become increasingly important for insurance companies. Due to the compute-intensive nature of such simulations, CPU-only implementations lack the ability to provide sufficient performance, while GPU-only solutions are unable to efficiently utilize the hardware and lead to disappointing results. In this work we proposed FiNS, a flexible heterogeneous framework which, based on modern technologies such as CUDA Streams, Hyper-Q and NVIDIAs MPS, is able to utilize both the CPU and the GPU to accelerate nested simulations. We demonstrated the use of FiNS for an ALM application, which required multiple CPU processes to offload calculations to the same GPU. To tackle this challenge, we customized the MPS functionality to handle local Python processes and achieved concurrency between streams owned by different local processes. Our results demonstrate that ALM as implemented using FiNS achieves very good parallel efficiency.

We have four main objectives for the future. First, we will focus on running our applications in a multi-GPU environment, to fully overlap the CPU and GPU execution. Second, we expect that the concept of FiNS could be implemented on any many-core architecture, like Intel MIC, but we need to test this. Third, we plan to investigate more applications [5] and the effort needed to implement them in FiNS. Last but not least, we will investigate the possibility of providing an intuitive front-end for this framework together with a computational infrastructure (e.g., in the cloud), enabling financial specialists to use it as a computational service.

References

1. Abbas-Turki, L., Vialle, S., Lapeyre, B., Mercier, P.: Pricing derivatives on graphics processing units using monte carlo simulation. Concurr. Comput.: Pract. Exp. **26**(9), 1679–1697 (2014). http://dx.doi.org/10.1002/cpe.2862
2. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr. Comput.: Pract. Exp. **23**(2), 187–198 (2011)
3. Bauer, D., Reuss, A., Singer, D.: On the calculation of the solvency capital requirement based on nested simulations. Astin Bull. **42**(02), 453–499 (2012)
4. Bernemann, A., Schreyer, R., Spanderen, K.: Accelerating exotic option pricing and model calibration using gpus (2011). http://ssrn.com/abstract=1753596
5. Cramwinckel, J.: GPU Accelerated framework for financial nested simulations. Master's thesis. VU University Amsterdam (2015)
6. Fernández, J., Ferreiro, A., García-Rodríguez, J., Leitao, A., López-Salas, J., Vázquez, C.: Static and dynamic sabr stochastic volatility models: calibration and option pricing using gpus. Math. Comput. Simul. **94**, 55–75 (2013)
7. Glasserman, P.: Monte Carlo Methods in Financial Engineering, vol. 53. Springer, New York (2004)
8. Joshi, M.S.: Graphical asian options. Wilmott J. **2**(2), 97–107 (2010)
9. Kofler, K., Grasso, I., Cosenza, B., Fahringer, T.: An automatic input-sensitive approach for heterogeneous task partitioning. In: ICS 2013 (2013)
10. Lee, A., Yau, C., Giles, M.B., Doucet, A., Holmes, C.C.: On the utility of graphics cards to perform massively parallel simulation of advanced monte carlo methods. J. Comput. Graph. Stat. **19**(4), 769–789 (2010)
11. Luitjens, J.: CUDA streams - best practices and pitfalls, Presentation GTC 2014 (2014)
12. Luk, C.K., Hong, S., Kim, H.: Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: MICRO 2009 (2009)
13. NVIDIA: Kepler GK110 architecture whitepaper, v1.0 (2012)
14. NVIDIA: Sharing a GPU between MPI processes: Multi-process service (MPS) overview (2013). Technical Brief TB-06737-003
15. Planas, J., Badia, R.M., Ayguadé, E., Labarta, J.: Self-Adaptive ompss tasks in heterogeneous environments. In: IPDPS 2013 (2013)
16. Shen, J., Varbanescu, A.L., Sips, H.: Look before you leap: using the right hardware resources to accelerate applications. In: HPCC (2014)
17. Shen, J., Varbanescu, A.L., Zou, P., Lu, Y., Sips, H.: Improving performance by matching imbalanced workloads with heterogeneous platforms. In: ICS (2014)
18. Tian, Y., Zhu, Z., Klebaner, F.C., Hamza, K.: Option pricing with the sabr model on the gpu. In: 2010 IEEE Workshop on High Performance Computational Finance (WHPCF). IEEE (2010)
19. Tian, Y., Zhu, Z., Klebaner, F.C., Hamza, K.: Pricing barrier and american options under the sabr model on the graphics processing unit. Concurr. Comput.: Pract. Exp. **24**(8), 867–879 (2012)
20. Wende, F., Steinke, T., Cordes, F.: Multi-threaded kernel offloading to gpgpu using hyper-q on kepler architecture. ZIB-Rep. 14–19 June 2014 (2014)