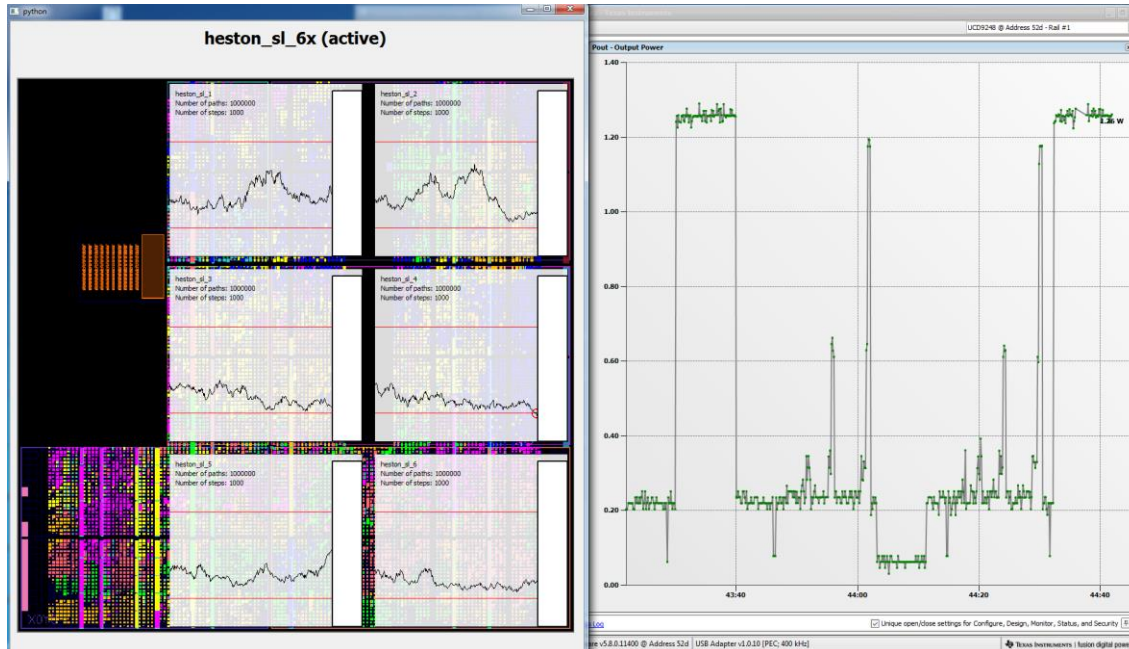


Bringing Flexibility to FPGAs with HyPER: Exotic Derivative Pricing on Zynq

Participant: Christian Brugger, Supervisor: Norbert Wehn,
University of Kaiserslautern, Germany, XIL-90398



YouTube: <https://www.youtube.com/watch?v=vC4c3mrf4dA>

GitHub: <https://github.com/tukl-msd/finance.zynqpricer.hls>

Introduction

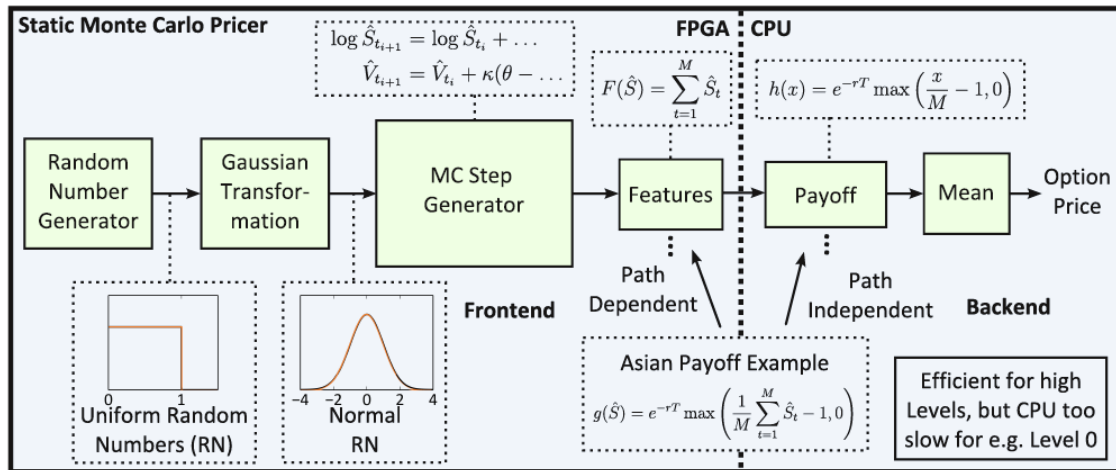
Derivatives present one of the biggest market in the world. To trade them financial institutions utilize more and more complex and demanding methods to price them. FPGAs have shown to be superior early on in this field. However, due to being perceived as being hard to use, they only slowly enter the financial business. This is especially true for derivatives, where new products and changes to the models are daily business. This project address this issue by fully exploiting newest Xilinx technology, in particular the Xilinx Zynq All Programmable SoC and Vivado High-Level Synthesis. The result of this project is the HyPER platform. HyPER makes use of reusable building blocks, an automatic platform optimizer and a domain specific language build on top of Vivado HLS. With that HyPER allows financial practitioners to easily model exotic derivatives with only minimal implementation knowledge and so-far unseen performance. Once the derivatives have been described by simple equations, the HyPER optimizer automatically creates several bitstreams that perfectly match the target device. HyPER explicitly make use of the dynamic reconfiguration capabilities of the Zynq on several levels. On a higher level the accelerator architecture will change for different products and on lower level even for different parts of the algorithm, to always perfectly balance all system resources. A central point of HyPER is to see the Zynq as a software central system running Linaro Ubuntu. In this view HyPER deploys simply as a

Linux software packaged that can be maintained as any other library in the datacenter. With so many things happening at so many levels one part of this project is to create a live visualization that shows what is happening in each part of the system. As the result of this project HyPER is expected to deliver two orders of magnitude improvements in energy efficiency compared to state-of-the-art CPU libraries, while providing the same of ease of use.

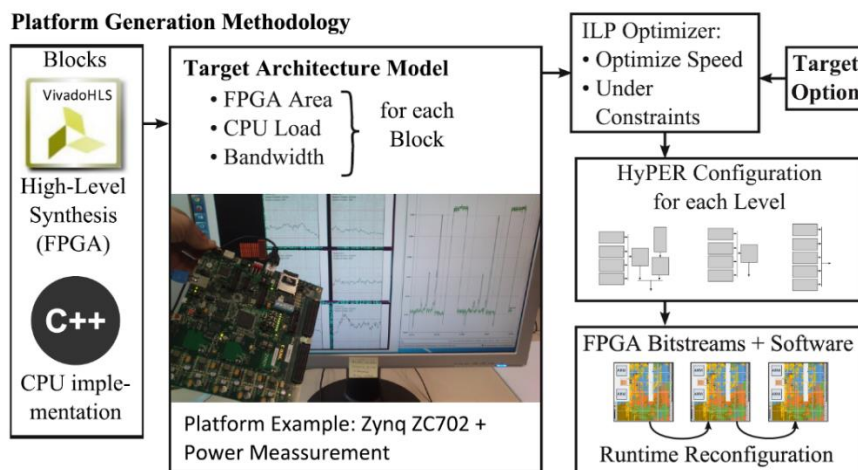
Design

The pricing architecture has the following structure. The final price is calculated in the statistics block. Depending on which algorithm is used and which option is priced different kernels are selected. We have two setups:

- Single-level Barrier Call: Uses Single-level Kernel, Barrier and Value at timepoint features and Call/Put payoff.
- Multilevel Barrier Call: Uses Single-level Kernel, Barrier and Value at timepoint features, Call/Put payoff and the ML-Diff kernel.



The kernels are then partitioned across the HW/SW of the Zynq, depending based on what makes best sense. The following platform generation methodology is used.



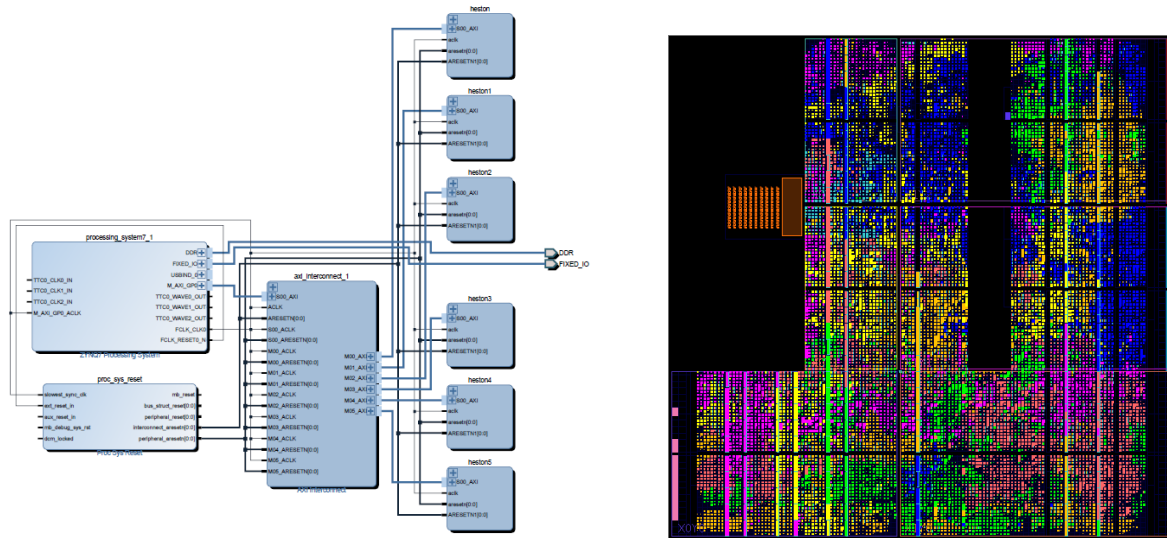
The kernels are all written in HLS (for HW = FPGA) and CPU (for SW = ARM). Each block is characterized based on FPGA area, CPU load and Bandwidth for the specific platform used. For the Xilinx Zynq 7020, the numbers are given in the following table. They are post place&route numbers of the HLS IP cores.

Building Blocks	LUT	FF	BRAM	DSP	CPU ns/val.
Increment Generator:					
Mersenne Twister	301	323	4	0	–
ICDF	451	592	4	1	–
Antithetic Core	228	258	0	0	–
Path Generators:					
Single-Level Kernel	4 153	4 241	2	38	–
Multilevel Kernel	5 607	5 326	6	43	–
Payoff Features F_i :					
Barrier	180	158	0	0	–
Payoff h :					
Call/Put	440	396	0	2	6
Backend:					
Feature					
Serializer $k \times 1$	$30k+65$	$65k+45$	0	0	–
Exponential	900	384	0	7	250
Multilevel Difference	372	355	0	2	5
Statistics II=1	2 170	1 612	4	9	6
Statistics II=2	1 454	1 164	2	6	3
Com. Interface Ψ					
FPGA \rightarrow CPU	LUT	FF	BRAM		Bandwidth in MB/s
Config-Bus $1 \times k$	$30k+50$	$2k+40$	0		< 1
Streaming-Fifo	654	611	4		20
DMA-Core	1 864	3 122	4		350
Hybrid Chip \mathcal{F}	LUT	FF	BRAM	DSP	ARM
Xilinx Zynq 7020	53 200	106 400	280	220	2 cores
Synthesis weight α	0.8	0.5	1	1	

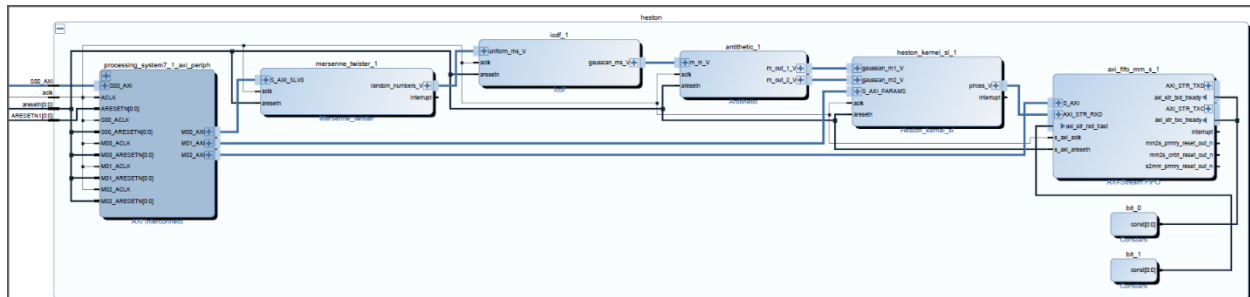
With this information a perfect configuration can be found via optimization. This configuration is then synthesized and we get different bitstreams. The multilevel algorithm has multiple levels and we might get different bitstreams for different levels, since they have different bandwidth requirements between the blocks for different levels.

This optimization process via ILP can be found in `\finance.zynqpricer.hls/ results/ 2013.12.02_platform_ILP`. The HLS IP cores in `\finance.zynqpricer.hls/hls`. The C++ software in `\finance.zynqpricer.hls/software`.

Once the optimal configuration is known they, they are currently generated by hand in Vivado and then synthesized. Different bitstreams have been synthesized in the folder `\finance.zynqpricer.hls/bitstream`. One example is the Single-Level bitstreams also shown in the video. In color the six Heston cores.



Each of the Heston cores has the following structure:



All of these cores are HLS ip cores, packaged in `\finance.zynqpricer.hls/ip`.

On the ARM cores, a full Linaro Ubuntu is running. During bootup an empty bitstream is loaded. The bootfiles are located in `finance.zynqpricer.hls\linux\boot_files`. How to install the Linaro Ubuntu is described in `finance.zynqpricer.hls\linux\full_setup.pdf`.

After bootup, you can login via ssh. The application then works like any other software. You can check it out from github:

```
git clone https://github.com/tukl-msd/finance.zynqpricer.hls.git
```

Then you can compile it and run the program via

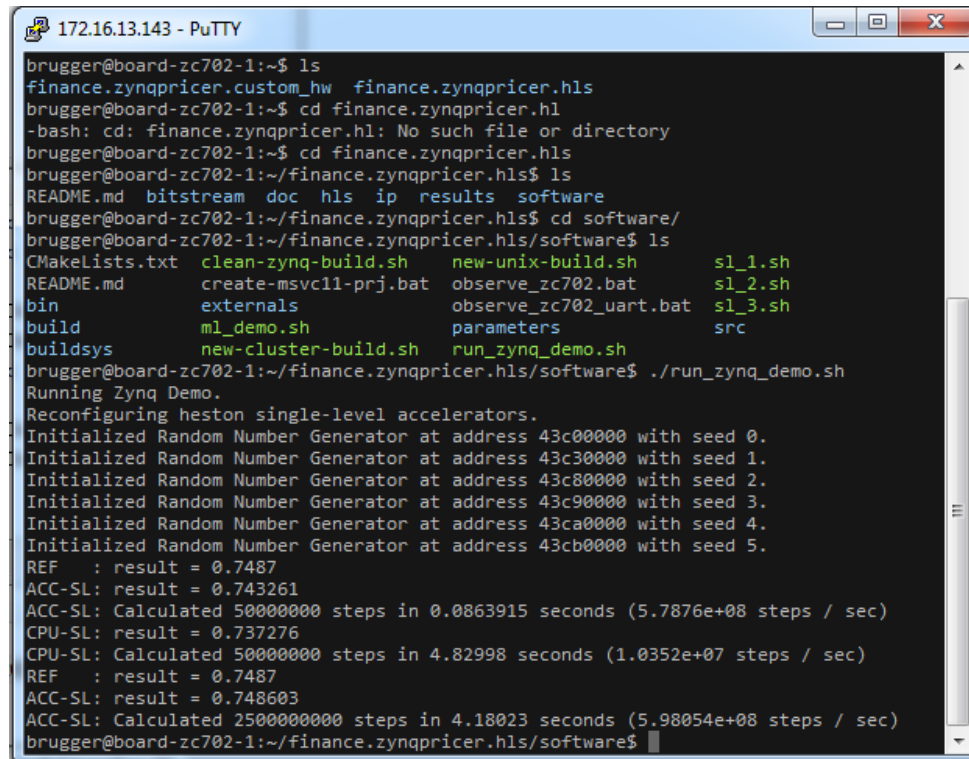
```
cd finance.zynqpricer.hls/software
./clean-zynq-build.sh
./run-zynq-demo.sh
```

Internally the last script will run the Heston executable in the following way:

```
bin/run_heston -sl -both params.json ../bitstream/heston_sl_6x.json
```

This will run the FPGA cores with the given parameter file. The second files specifies the configured cores of the bitstreams, including address ranges and the number of cores. With that the software also

knows which parts to run. In the Single-Level example these are accumulating the result and calculating the final option price via a correction formula. Internally the script configures the FPGA via dynamic configuration. Sets up the random number generators and configures all the cores. Once the result is ready, it is reported on the screen.

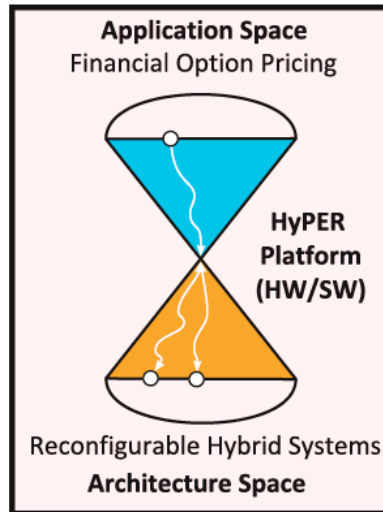


```
172.16.13.143 - PuTTY
brugger@board-zc702-1:~$ ls
finance.zynqpricer.custom_hw  finance.zynqpricer.hls
brugger@board-zc702-1:~$ cd finance.zynqpricer.hls
-bash: cd: finance.zynqpricer.hls: No such file or directory
brugger@board-zc702-1:~$ cd finance.zynqpricer.hls
brugger@board-zc702-1:~/finance.zynqpricer.hls$ ls
README.md  bitstream  doc  hls  ip  results  software
brugger@board-zc702-1:~/finance.zynqpricer.hls$ cd software/
brugger@board-zc702-1:~/finance.zynqpricer.hls/software$ ls
CMakeLists.txt  clean-zynq-build.sh  new-unix-build.sh  sl_1.sh
README.md       create-msvc11-prj.bat  observe_zc702.bat  sl_2.sh
bin             externals             observe_zc702_uart.bat  sl_3.sh
build           ml_demo.sh           parameters          src
buildsys        new-cluster-build.sh  run_zynq_demo.sh
brugger@board-zc702-1:~/finance.zynqpricer.hls/software$ ./run_zynq_demo.sh
Running Zynq Demo.
Reconfiguring heston single-level accelerators.
Initialized Random Number Generator at address 43c00000 with seed 0.
Initialized Random Number Generator at address 43c30000 with seed 1.
Initialized Random Number Generator at address 43c80000 with seed 2.
Initialized Random Number Generator at address 43c90000 with seed 3.
Initialized Random Number Generator at address 43ca0000 with seed 4.
Initialized Random Number Generator at address 43cb0000 with seed 5.
REF : result = 0.7487
ACC-SL: result = 0.743261
ACC-SL: Calculated 50000000 steps in 0.0863915 seconds (5.7876e+08 steps / sec)
CPU-SL: result = 0.737276
CPU-SL: Calculated 50000000 steps in 4.82998 seconds (1.0352e+07 steps / sec)
REF : result = 0.7487
ACC-SL: result = 0.748603
ACC-SL: Calculated 250000000 steps in 4.18023 seconds (5.98054e+08 steps / sec)
brugger@board-zc702-1:~/finance.zynqpricer.hls/software$
```

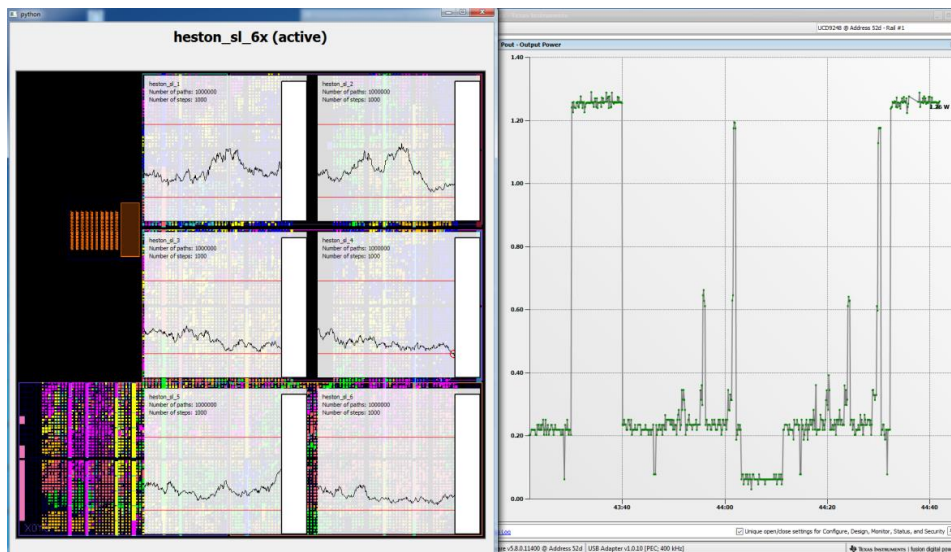
The software works on all boards with a Zync 7020 chip out of the box like this. And the whole framework was built to provide portable performance among all Zynq boards. For bigger boards more cores are generated and the hardware / software split might be different, but always optimal for the given platform.

The framework is also flexible from an application point of view. Adding new option products only require adding a few more cores to the platform. Based on the cores written, also simple templates can be created, where application people just enter the formula of the contract, what is called the payoff formula. So HLS help us to create a very simple domain-specific language. Once that is in place, finance application people can use FPGA without ever using complex tools, since the platform find always the best implementation by its own.

So we didn't just have a Heston pricer implementation here, but a platform that allows us to generate efficient implementations for all Zynq platforms and all European options. As visualized below.



To visualize the results a Desktop program has been created that sends the different commands to the Zynq via SSH. The script is located in `finance.zynqpricer.hls\software\src\observe_board.py`. It communicates with the Heston Executable. On start it receives the hardware configuration as json. Based on that the floorplan is visualized. Once the program starts running, the parameters and paths are send to the GUI, which are then visualized. The visualization is live and at the same time the power can be visualized with the TI Fusion Digital Power Designer. Based on that all the measurements have been made.



Complexity and IP Ownership

All application specific IP cores for the FPGA have been written by myself. (all code in `\finance.zynqpricer.hls\hls`) AXI-Interconnect cores have been used from Xilinx. All in all these are 1689 lines of code.

For the software all application code has been written by myself, while using libraries for Mersenne Twister (random number generator), Ziggurat (Gaussian Transformation) and JSON reader. These are

included in the folder `\finance.zynqpricer.hls\software\externals`. All other code in `\finance.zynqpricer.hls\software` has been written by myself. All in all these are 4060 lines of code.

Design Reuse

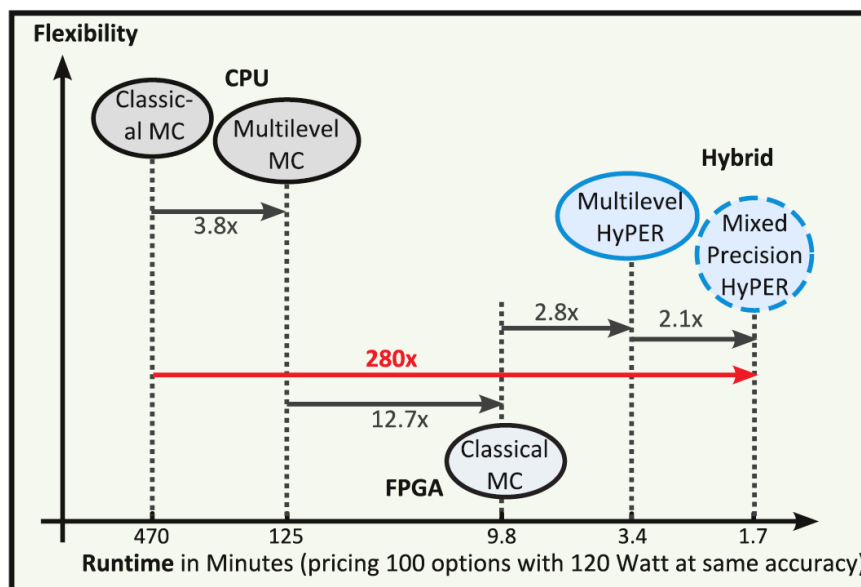
All parts of the design have been made available (Open Source) for reuse through GitHub [1]. This includes the source files for the HLS IP cores, the C++ software running on the Zynq, visualizations scripts, instructions on how to compile and setup the project and even the setups and results to generate the data for the paper.

The ideas have been published in a conference paper [2].

Especially the random number generator (a Mersenne Twister) is interesting for reuse.

Results

Financial institutions use big clusters to do their calculation. Our goal was to investigate whether FPGAs can bring innovation to this market. Being in an interdisciplinary team with mathematicians we first optimized the classical Monte Carlo algorithms by using more advanced multilevel schemes. In our setup this resulted in 3.8x speedups already. A previous FPGA implementation in VHDL of the classical Monte Carlo was published by Christian De Schryver [3]. It showed that FPGAs bring an additional 12.7x speedup, however they provide no flexibility. Adding new products is tedious and could only be done by experts, taking several days. With HyPER we brought two innovations, flexibility and efficiency. We use multilevel Monte Carlo and get another 2.8x and we created the platform to allow application practitioners to add new products. All optimizations like the hardware/software split are then done automatically. We further showed that by using reduced precision formats, instead of single-precision floating points as on the clusters and our FPGA implementations, another 2.1x speedup can be achieved [4]. However this is not part of the project, more as an outlook. The results:



HyPER is written with Vivado HLS, while the classical Monte Carlo implementation in [3] is written in VHDL. A comparison showed that 8.9x lines of code were necessary to realize the same hardware IP cores, while providing an even slightly smaller hardware, by saving Flip-flops through better scheduling.

Conclusion

With this work, we showed the financial community the potential of FPGAs, bringing two orders of magnitude speedups while providing a similar flexibility. This was only possible by using newest technology like hybrid devices like the Zynq, high-level languages like Vivado HLS and innovating on how we write software for FPGAs. Not by creating fixed designs, but creating portable platforms.

References

- [1] Source code of the project on GitHub, available at: <https://github.com/tukl-msd/finance.zynqpricer.hls> last-access: 2015-06-30
- [2] Brugger, Christian, Christian de Schryver, and Norbert Wehn. "HyPER: A runtime reconfigurable architecture for Monte Carlo option pricing in the Heston model." Field Programmable Logic and Applications (FPL), 2014 24th International Conference on. IEEE, 2014.
- [3] De Schryver, Christian, et al. "An energy efficient FPGA accelerator for Monte Carlo option pricing with the heston model." Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on. IEEE, 2011.
- [4] Brugger, Christian, et al. "Mixed precision multilevel Monte Carlo on hybrid computing systems." Computational Intelligence for Financial Engineering & Economics (CIFEr), 2104 IEEE Conference on. IEEE, 2014.