EPJ Web of Conferences **302**, 04007 (2024) https://doi.org/10.1051/epjconf/202430204007
SNA + MC 2024

XSBench on FPGAs using Intel oneAPI

Oldřich Pecák[1], *Zdeněk* Matěj[1], and *Václav* Přenosil[1]

[1]Faculty of Informatics, Masaryk University, Botanická 68a, Brno

Abstract. Field-Programmable Gate Arrays (FPGAs) are becoming an interesting component for heterogeneous computing systems in the post-Moore era thanks to their reconfigurable nature. The current generation of FPGAs includes specialized hard blocks for floating point operations, making them attractive for scientific computing. FPGA programming has historically been done in hardware description languages, which required a deep understanding of hardware design. Emerging high-level synthesis tools, such as Intel oneAPI and AMD Vitis™ , provide a more common programming environment for FPGAs. In this paper, we explore the capabilities of FPGAs for acceleration in the context of nuclear particle transport simulators. As a case study, we implement XSBench in Intel oneAPI targeting FPGAs, including basic optimizations. We then compare the performance of Intel Stratix10 FPGA and Intel Xeon CPU-based systems and evaluate the viability of FPGA use in heterogeneous systems.

1 Introduction

Modern high-performance computing relies more and more on heterogeneous systems due to the end of Moore's law and practical limits on power density [1]. High-performance systems are nowadays compromised of CPUs working together with GPUs, and exascale supercomputers like Frontier gain most of their FLOPS from GPUs [2].

Field-Programmable Gate Arrays (FPGAs) are becoming another interesting component for heterogeneous systems. They offer fully reconfigurable digital logic, enabling users to create custom hardware. They recently started gaining large amounts of floating point capabilities, and a recent offering from Intel, Agilex 7 offers up to 38 TFLOPs [3] of compute in a single chip. This makes them attractive for scientific high-performance computing. They still have an order of magnitude less raw floating point performance than modern GPU accelerators - Nvidia Hopper achieves up to 756 TFLOPs [4]. Although less powerful, FPGAs provide a different execution model to fixed architecture devices like CPUs and GPUs.

FPGAs are notoriously hard to program, as until recently, they used exclusively hardware description languages. This approach requires deep hardware design knowledge and a large effort to produce high performance. The emergence of high-level synthesis tools can provide an alternative to this approach and also make FPGAs a more common element of heterogeneous systems.

In this paper, we evaluate the relatively new Intel oneAPI toolkit with its DPC++ language for use on FPGAs. We take an existing oneAPI-compatible implementation of XSBench and test it on FPGAs. We try to achieve competitive performance with as little optimization work as possible and also try to keep FPGA-specific optimizations to a minimum.

The rest of the paper is structured in the following way. We give the background in section 2, including FPGAs, XSBench, oneAPI and related work. We describe our implementation of XSBench in section 3. The testing and results are described in section 4, and the paper is concluded in section 5.

2 Background

2.1 Field-Programmable Gate Arrays

FPGAs are a family of reconfigurable integrated circuits. They do not have fixed functionality; rather, they can be configured at runtime to implement any digital hardware circuit [5, 6]. They implement a matrix of configurable logic blocks, digital signal processing blocks and memory blocks. The interconnect can be programmed to create a digital circuit out of these basic building blocks. Modern FPGAs also usually include hard blocks. These blocks usually include memory controllers (i.e. DDR4), PCIe controllers, fully featured CPUs, ethernet transceivers and other commonly used features. Figure 1 shows a generic block schematic of modern FPGA.

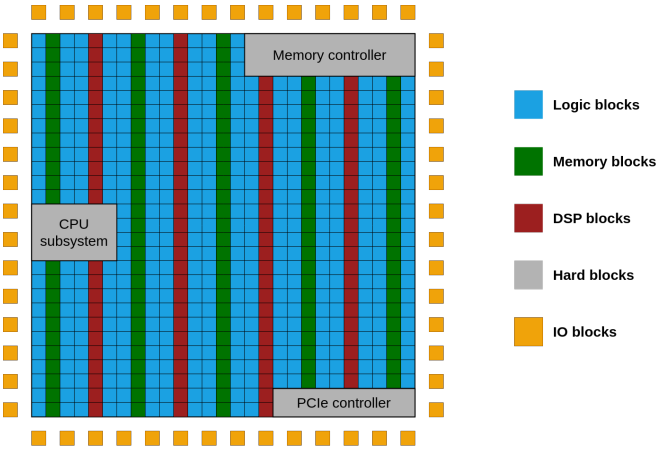

Figure 1. Modern FPGA high level block diagram

FPGAs have been traditionally used for application-specific integrated circuit(ASIC) verification and emulation. They are also often used in low-volume products, where custom ASICs would be economically unviable. As they implement pure digital logic circuits, they are also used where low latency and deterministic delays are required, such as signal processing [6].

The traditional way of programming FPGAs uses a hardware description language such as Verilog or VHDL. This approach requires a deep understanding of the hardware you want to create, and thus, it is often relegated to dedicated hardware engineers. This presents a skill and knowledge wall, making FPGAs challenging to use and evaluate.

Recently, high-level synthesis (HLS) tools such as Intel oneAPI FPGA toolkit or AMD Vitis™ started gaining traction. These tools enable a direct translation from regular code (usually C/C++ based) to FPGA hardware [7]. This makes FPGAs more available for people with little FPGA experience, enabling them to use the potential benefits of FPGAs in various high-performance computing applications. Thanks to the HLS tools, there has been an uptick in the usage of FPGAs in high-performance computing, such as neural network accelerators [8], molecular dynamics simulations [9] or astrophysics [10].

There are several limitations currently present. FPGAs have limited resources, often much smaller than equivalent GPU or CPU. Thus, a larger computation kernel might not fit on an FPGA. Full compilation for FPGA hardware usually takes several hours, being extremely long compared to other platforms. There are emulators provided by the vendors for functional verification, but these are also limited. One notable example is that they do not run the possibly interacting parallel tasks in parallel but sequentially, leading to possible mismatches between emulation and physical hardware [11].

From a high-level view, FPGAs provide a different programming model from the common data and task parallelism. They are a spatial architecture [12, Chapter 17] where the whole program maps to the hardware. This results in a very pipeline-focused style of acceleration, where the acceleration comes from keeping the pipeline fed with data and executing every pipeline step at once in parallel. Figure 2 shows this idea visually. This pipeline-focused model can be beneficial for workloads that do not work well with the data parallel model of acceleration.

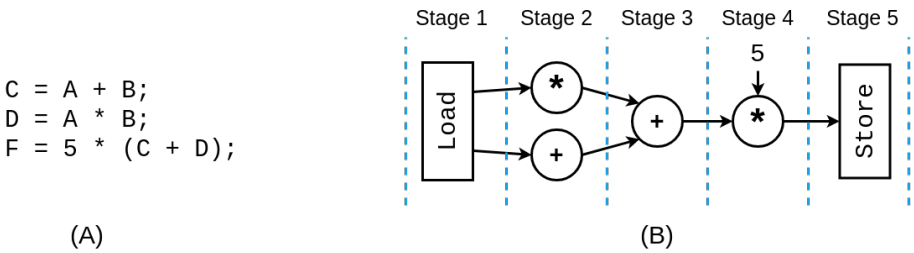

Figure 2. Visual representation of pipeline focused acceleration. (A) Simple sequential code. (B) Same code, mapped to 5 stage spatial pipeline, where stages can be executed in parallel, if we keep the pipeline fed with data.

2.2 XSBench

XSBench is a simple proxy application that simulates the most computationally intensive part of Monte Carlo nuclear particle transport simulators [13]. That is the macroscopic cross-section lookup, which happens at every particle collision, and it takes up to 85% of the total runtime of the simulator. This cross-section depends on the incoming particle's energy and collision type. The cross-section data is saved in an energy grid. Depending on the nuclide, the table with the data can have up to hundreds of thousands of different energy entries, resulting in large arrays. XSBench does this lookup using a binary search on the energy grid, including optimizations such as unionized energy grid [14]. The model included in the application typically uses around 5.6 GB of memory during runtime. XSBench picks random energy E and material m (a collection of nuclides in defined rations) for each lookup.

XSBench currently has implementations for several frameworks, including OpenMP, CUDA, SYCL(DPC++) and more. It has been shown to accurately mimic the simulators, so a good performance with XSBench should translate to a good simulator performance [13].

2.3 Intel oneAPI

Intel oneAPI is a software toolkit for developing applications for multiple architectures [11]. It provides a unified programming model for various accelerators. The oneAPI toolkit also

includes all required compilers, libraries, and development tools. The FPGA support is separated into an addon, called Intel FPGA Addon for oneAPI [15].

The unified programming model is provided by Data Parallel C++ (DPC++) [16]. It is an implementation and extension of SYCL [17], a Khronos Group standard defining a high-level programming model for heterogeneous architectures.

2.4 Related work

Several papers testing the capability of FPGAs for nuclear particle transport have been published. Jin et al. tested several OpenCL FPGA ports of XSBench and the similar RSBench benchmark [18–21]. They focus on the OpenCL version of XSBench, implement various optimizations of the look-up kernel and then evaluate the effort required for optimization using HLS tools. They rate the effort as adequate but not entirely suitable for HPC yet, as the optimizations still require quite a lot of FPGA knowledge, and the compile times limit the development speed. According to their results, FPGAs have efficiency between CPU and GPU implementations, achieving higher performance per watt than CPUs but lagging behind GPUs.

3 XSBench oneAPI implementation

XSBench already has a DPC++ implementation, which we use as a starting point. This implementation is written with CPUs and GPUs in mind, and as such, it is unsuitable for FPGAs.

During optimization efforts, we used the tools provided by the oneAPI toolkit. The most useful is the FPGA Optimization report. This report shows the compiler's analysis of loops and bottlenecks and other useful analyses, such as schedule view and FPGA area usage. It also generates most of its output after only the initial compilation stage, so we do not need to run the whole hardware compilation flow, which can take up to 10 hours in our case. We implemented all our optimizations based on the output from the reports.

Our first optimization was changing the data movement from implicit (which uses DPC++ buffers and accessors) to explicit, where we manually copy data to the FPGA. We did this based on running Intel VTune and analyzing the CPU-FPGA interactions, which indicated many memory transfers, as the compiler decided to keep some of the read-only data in CPU memory. This resulted in higher kernel f_{MAX}, which is the maximum frequency at which the pipeline can process data.

The second optimization we implemented is loop unrolling. This was again aided by the optimization report, in which we identified several loops that can be unrolled, as they always have a constant number of iterations. These loops often access arrays in a cycle-independent manner, so they are naively parallelizable. The unrolling was implemented using a compiler pragma, a single line of code. See listing 1 for example. The loop unrolling lowered the pipeline's f_{MAX}. Although the frequency is proportional to performance, this optimization enabled the pipeline to be fully parallel and resulted in orders of magnitude higher performance, see section 4. As a second stage of this optimization, we also unrolled (or partially unrolled, where applicable) nested loops.

The last optimization we did was multiplying the pipeline several times. In the optimization reports, we noticed that the final pipeline takes up a relatively small area (see table 1). The idea was to duplicate the pipelines, leading to performance uplift, as multiple pipelines are now working at once. We tried two approaches - partially unrolling the top-level loop over all lookups (thus leading to multiple lookups happening at once) and manually creating multiple pipelines. This optimization did not improve performance; see section 4.

Listing 1. Example of unrolled nested loop using pragmas

```
#pragma unroll
for (int i = 0; i < 12; i++) {
  double running = 0;
  #pragma unroll 12
  for (int j = i; j > 0; j--)
    running += distribution[j]
  if (roll < running)
    return i;
}
```

We did not do any optimizations to the binary search itself and we use the basic implementation. This is non-optimal, as there are parallelizable versions of the binary search algorithm. Some of them are implemented and tested in the work by Jin et al.[20].

The final kernel area and achieved f_{MAX} for the various optimization stages is in table 1. The limiting factor on a number of pipelines was the maximum amount of flip-flops available in the FPGA. Most optimizations lead to an increase in area usage and decrease of f_{MAX}. A significant amount of the FPGA is also used for the static portion of the kernel - the interfacing to host and kernel dispatching.

Table 1. FPGA Area usage for various kernel versions from the Optimization report

Type	ALUTs	FFs	RAMs	DSPs	f_{MAX}
Static area	25 %	25 %	26 %	22 %	
v0 - Base kernel	9 %	19 %	25 %	2 %	269 MHz
v1 - Explicit memory	6 %	16 %	16 %	1 %	293 MHz
v2 - Basic unrolling	9 %	17 %	11 %	2 %	187 MHz
v2.5 - Nested loop unrolling	7 %	16 %	11 %	2 %	185 MHz
v3 - Partially unrolled main loop	18 %	65 %	31 %	5 %	249 MHz
v4 - 4 parallel pipelines	24 %	64 %	42 %	7 %	182 MHz

4 Testing results

We compare all of the optimization variants of the kernel on FPGAs and compare it to a classic CPU implementation. Additionally we test the baseline oneAPI version on CPU. We also measure the power consumption of the FPGA using the card's power management system. We could not measure the exact CPU power usage as we did not have sufficient access privileges for the system used. Each version was tested on the default model size (around 5.6GB of data) in XSBench, with 17 million look-ups.

As an FPGA we use Intel Stratix 10 SX 2800 on Intel FPGA PAC D5005 PCIe card. It has 2753000 logic blocks, 244 Mb of embedded memory and 5760 digital signal processing blocks, which support variable precision and floating point operations [22]. The CPU used in benchmarks is Intel Xeon Platinum 8256. It has 4 cores, 8 threads at 3.8 GHz with 16.5 MB of Cache and a maximum TDP of 105 watts [23].

We ran each version of the XSBench 5 times, taking an average of the results to minimize the effect of caching. The average achieved lookups per second are in figure 3.

The basic DPC++ implementation available in XSBench performs worse than the OpenMP version, achieving about half the performance of OpenMP. This is expected, as OpenMP is a more mature system, and the oneAPI overhead is probably more significant than OpenMP's. On FPGA, the basic implementation is several orders of magnitude slower than the CPU. This is due to the pipeline in the kernel being unparallelized, resulting in sequential execution at

low speed - the compiled FPGA kernel clock is around 270 MHz, according to tooling reports, compared to the 3.8GHz of the CPU.

The kernel versions with loop unrolling all achieve similar performance, except for the partially unrolled main loop (v3). Their absolute performance is similar to the CPU implementation between the OpenMP and oneAPI. This uplift in the performance is caused by the compiler parallelizing the pipeline due to loop unrolling.

The two versions that try to increase the number of pipelines (v3 and v4) do not give the expected performance uplift. This is primarily due to the limited number of memory controllers in the FPGA. According to optimization reports and profiling, the pipelines are stalling while waiting for access to the controllers' memory bus.

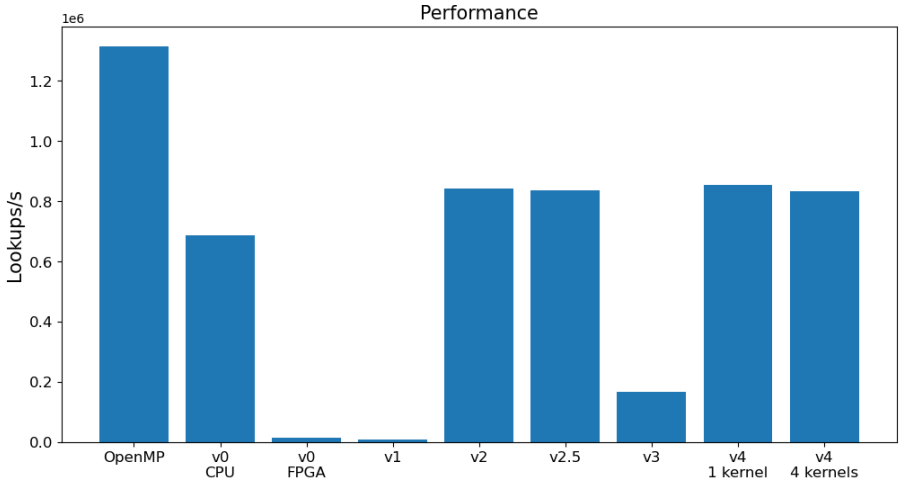

Figure 3. Achieved performance per version

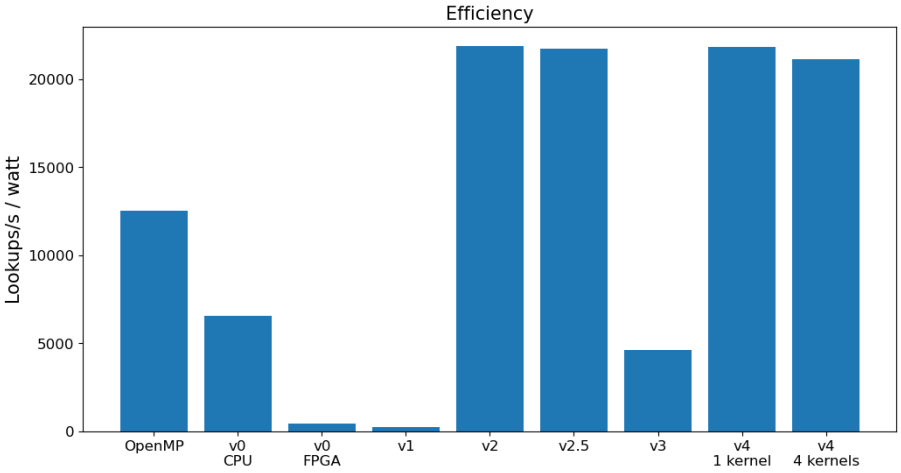

Figure 4. Achieved performance per watt per version

The FPGA power consumption was measured as a maximum through the benchmark run. As we could not measure the CPU power, we estimated using the maximum TDP as the CPU power consumption. We make this assumption because most CPUs run near TDP when under heavy load, and XSBench fully utilizes all available cores. This does not fully reflect typical CPU behaviour, but it is a good enough approximation to compare it with the coarse FPGA power consumption measurement.

The achieved lookups per watt are in figure 4. The FPGA implementation is more power efficient than the CPU one. We can achieve up to an order of magnitude better power efficiency than the CPU. Interestingly, the idle power consumption of the FPGA card is around 30 watts and around 35 watts while running the kernel. The dynamic power increase caused by the kernel is thus very small, possibly leading to even higher actual power efficiency, depending on how the power efficiency is calculated.

5 Conclusion

Considering that we have achieved such results in a short time and with relatively low effort, oneAPI and other HLS tools significantly improve the typical FPGA development flow. They enable scientists and researchers to evaluate FPGA performance without gaining a deep understanding of FPGAs and hardware description languages. This makes it possible for a wide area of high-performance computing to evaluate FPGA use quickly. The long compile times still hinder a fast development cycle, but oneAPI tooling makes it easier to identify potential optimization possibilities. Our FPGA implementation took only several hours of actual development time (not counting compile time). It required only around 50 lines of modified or added code to the CPU-focused DPC++ implementation, including the basic optimizations.

Our testing has shown that our FPGA implementation is about 60 % as performant as normal CPU implementation while being two times more power efficient. Our results agree with previous results from Jin et al. [18–21]. This makes FPGAs a viable component of heterogenous systems due to their power efficiency. There is still room for improvement, as we implemented only the most basic optimizations possible.

We are interested in further testing XSBench, possibly getting more into device-specific optimizations that exploit the capabilities of FPGAs. A great future direction is also exploring a design of fully custom accelerator architecture for nuclear particle transport simulators, as the custom architecture potential of FPGAs remains unexplored. Writing a new simulator from scratch designed for FPGAs is also an exciting possibility.

Acknowledgment

Benchmarks and FPGA compilation were done on infrastructure provided by Intel Corporation via their Intel Developer Cloud[1].

[1]https://www.intel.com/content/www/us/en/developer/tools/devcloud/services.html

References

[1] C.B. Ciobanu, A.L. Varbanescu, D. Pnevmatikatos, G. Charitopoulos, X. Niu, W. Luk, M.D. Santambrogio, D. Sciuto, M.A. Kadi, M. Huebner et al., *EXTRA: Towards an Efficient Open Platform for Reconfigurable High Performance Computing*, in *2015 IEEE 18th International Conference on Computational Science and Engineering* (2015), pp. 339–342, https://ieeexplore.ieee.org/abstract/document/7371394

[2] S. Atchley, C. Zimmer, J. Lange, D. Bernholdt, V. Melesse Vergara, T. Beck, M. Brim, R. Budiardja, S. Chandrasekaran, M. Eisenbach et al., *Frontier: Exploring Exascale*, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Association for Computing Machinery, New York, NY, USA, 2023), SC '23, pp. 1–16, ISBN 9798400701092, https://dl.acm.org/doi/10.1145/3581784.3607089

[3] *1. Overview of the Intel Agilex® 7 FPGAs and SoCs*, https://www.intel.com/content/www/us/en/docs/programmable/683458/current/overview-of-the-fpgas-and-socs.html

[4] W. Luo, R. Fan, Z. Li, D. Du, Q. Wang, X. Chu, *Benchmarking and Dissecting the Nvidia Hopper GPU Architecture* (2024), arXiv:2402.13499 [cs], http://arxiv.org/abs/2402.13499

[5] A. Boutros, V. Betz, IEEE Circuits and Systems Magazine **21**, 4 (2021), conference Name: IEEE Circuits and Systems Magazine

[6] S. Gandhare, B. Karthikeyan, *Survey on FPGA Architecture and Recent Applications*, in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN)* (2019), pp. 1–4, https://ieeexplore.ieee.org/abstract/document/8899550

[7] H.M. Waidyasooriya, Y. Iimura, M. Hariyama, *Benchmarks for FPGA-Targeted High-Level-Synthesis*, in *2019 Seventh International Symposium on Computing and Networking (CANDAR)* (2019), pp. 232–238, iSSN: 2379-1896, https://ieeexplore.ieee.org/document/8958473

[8] K. Guo, S. Zeng, J. Yu, Y. Wang, H. Yang, *A Survey of FPGA-Based Neural Network Accelerator* (2018), arXiv:1712.08934 [cs], http://arxiv.org/abs/1712.08934

[9] H.M. Waidyasooriya, M. Hariyama, K. Kasahara, *Architecture of an FPGA accelerator for molecular dynamics simulation using OpenCL*, in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)* (IEEE, Okayama, Japan, 2016), pp. 1–5, ISBN 978-1-5090-0806-3, http://ieeexplore.ieee.org/document/7550743/

[10] R. Kashino, R. Kobayashi, N. Fujita, T. Boku, *Multi-hetero Acceleration by GPU and FPGA for Astrophysics Simulation on oneAPI Environment*, in *International Conference on High Performance Computing in Asia-Pacific Region* (ACM, Virtual Event Japan, 2022), pp. 84–93, ISBN 978-1-4503-8498-8, https://dl.acm.org/doi/10.1145/3492805.3492817

[11] *Intel® oneAPI Programming Guide*, https://www.intel.com/content/www/us/en/docs/oneapi/programming-guide/2024-0/overview.html

[12] J. Reinders, B. Ashbaugh, J. Brodman, M. Kinsner, J. Pennycook, X. Tian, *Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL* (Apress, Berkeley, CA, 2021), ISBN 978-1-4842-5573-5 978-1-4842-5574-2, http://link.springer.com/10.1007/978-1-4842-5574-2

[13] J.R. Tramm, A.R. Siegel, T. Islam, M. Schulz (2014)

[14] J.A. Walsh, P.K. Romano, B. Forget, K.S. Smith, Computer Physics Communications **196**, 134 (2015)

[15] *Intel® oneAPI FPGA Handbook*, https://www.intel.com/content/www/us/en/docs/oneapi-fpga-add-on/developer-guide/2024-0/intel-oneapi-fpga-handbook.html

[16] B. Ashbaugh, A. Bader, J. Brodman, J. Hammond, M. Kinsner, J. Pennycook, R. Schulz, J. Sewall, *Data Parallel C++: Enhancing SYCL Through Extensions for Productivity and Performance*, in *Proceedings of the International Workshop on OpenCL* (Association for Computing Machinery, New York, NY, USA, 2020), IWOCL '20, pp. 1–2, ISBN 978-1-4503-7531-3, https://dl.acm.org/doi/10.1145/3388333.3388653

[17] R. Reyes, G. Brown, R. Burns, M. Wong, *SYCL 2020: More than meets the eye*, in *Proceedings of the International Workshop on OpenCL* (Association for Computing Machinery, New York, NY, USA, 2020), IWOCL '20, p. 1, ISBN 978-1-4503-7531-3, https://dl.acm.org/doi/10.1145/3388333.3388649

[18] Z. Jin, H. Finkel, *Nuclear Reactor Simulation on OpenCL FPGA: a Case Study of RSBench*, in *Proceedings of the International Workshop on OpenCL* (ACM, Oxford United Kingdom, 2018), pp. 1–9, ISBN 978-1-4503-6439-3, https://dl.acm.org/doi/10.1145/3204919.3204921

[19] Z. Jin, H. Finkel, *Nuclear Reactor Simulations on OpenCL FPGA Platform*, in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Association for Computing Machinery, New York, NY, USA, 2019), FPGA '19, p. 304, ISBN 978-1-4503-6137-8, https://doi.org/10.1145/3289602.3293983

[20] Z. Jin, H. Finkel, *Performance Evaluation of the Vectorizable Binary Search Algorithms on an FPGA Platform*, in *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)* (2020), pp. 63–67, https://ieeexplore.ieee.org/abstract/document/9407263

[21] Y. Luo, X. Wen, K. Yoshii, S. Ogrenci-Memik, G. Memik, H. Finkel, F. Cappello, *Evaluating irregular memory access on OpenCL FPGA platforms: A case study with XSBench*, in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)* (2017), pp. 1–4, iSSN: 1946-1488, https://ieeexplore.ieee.org/abstract/document/8056827

[22] *Intel® Stratix® 10 SX 2800 FPGA Product Specifications*, https://www.intel.com/content/www/us/en/products/sku/210308/intel-stratix-10-sx-2800-fpga.html

[23] *Intel® Xeon® Platinum 8256 Processor (16.5M Cache, 3.80 GHz) - Product Specifications*, https://www.intel.com/content/www/us/en/products/sku/192467/intel-xeon-platinum-8256-processor-16-5m-cache-3-80-ghz/specifications.html