Evaluating FPGA Accelerator Performance with a Parameterized OpenCL Adaptation of Selected Benchmarks of the HPCChallenge Benchmark Suite

Marius Meyer*, Tobias Kenter† and Christian Plessl‡
Department of Computer Science and Paderborn Center for Parallel Computing (PC^2)
Paderborn University, Paderborn, Germany
Email: *marius.meyer@uni-paderborn.de, †tobias.kenter@uni-paderborn.de, ‡christian.plessl@uni-paderborn.de

***Abstract*—FPGAs have found increasing adoption in data center applications since a new generation of high-level tools have become available which noticeably reduce development time for FPGA accelerators and still provide high-quality results. There is, however, no high-level benchmark suite available, which specifically enables a comparison of FPGA architectures, programming tools, and libraries for HPC applications.**

To fill this gap, we have developed an OpenCL-based open-source implementation of the HPCC benchmark suite for Xilinx and Intel FPGAs. This benchmark can serve to analyze the current capabilities of FPGA devices, cards, and development tool flows, track progress over time, and point out specific difficulties for FPGA acceleration in the HPC domain. Additionally, the benchmark documents proven performance optimization patterns. We will continue optimizing and porting the benchmark for new generations of FPGAs and design tools and encourage active participation to create a valuable tool for the community.

***Index Terms*—FPGA, OpenCL, High Level Synthesis, HPC benchmarking**

I. Introduction

In High Performance Computing (HPC), benchmarks are an important tool for performance comparison across systems. They are designed to stress important system properties or generate workloads that are similar to relevant applications for the user. Especially in acquisition planning they can be used to define the desired performance of the acquired system before it is built. Since it is a challenging task to select a set of benchmarks to cover all relevant device properties, benchmark suites can help by providing a pre-defined mix of applications and inputs, for example SPEC CPU [1] and HPC Challenge (HPCC) [2].

There is an ongoing trend towards heterogeneity in HPC, complementing CPUs by accelerators, as indicated by the Top 500 list [3]. From the top 10 systems in the list, seven are equipped with different types of accelerators. Nevertheless, to get the best matching accelerator for a new system, a tool is needed to measure and compare the performance across accelerators. For well-established accelerator architectures like Graphics Processing Units (GPUs), there are already standardized benchmarks like SPEC ACCEL [4]. For Field Programmable Gate Arrays (FPGAs), that are just emerging as accelerator architecture for data centers and HPC, existing benchmarks do not focus on HPC and miss to measure highly relevant device properties.

Similar to the compiler for CPU applications, the High Level Synthesis (HLS) framework consisting of Software Development Kit (SDK) and Board Support Package (BSP) takes a very important role to achieve performance on an FPGA. The framework translates the accelerator code (denoted as *kernel*), most commonly from Open Computing Language (OpenCL), to intermediate languages, organizes the communication with the underlying BSP, performs optimizations and synthesizes the code to create executable FPGA configurations (bitstreams). Hence, the HLS framework has a big impact on the used FPGA resources and the maximum kernel frequency, which might vary depending on the kernel design. An HPC benchmark suite for FPGAs should capture this impact and, for comparisons, must not be limited to a single HLS framework.

In this paper, we propose *HPCC FPGA*, an FPGA OpenCL benchmark suite for HPC using the applications of the HPCC benchmark suite. The motivation for choosing HPCC is that it is well-established for CPUs and covers a small set of applications that evaluate important memory access and computing patterns that are frequently used in HPC applications.

Specifically, we make the following contributions in this paper:

1) We provide FPGA-adapted OpenCL kernel implementations along with corresponding host code for setup and measurements for a selection of HPCC benchmark applications.
2) We provide configuration options for the OpenCL kernels that allow adjustments to resources and architecture of the target FPGA and board without the need to change the code manually.
3) We evaluate the execution of these benchmarks on different FPGA families and boards with Intel and Xilinx FPGAs and show the benchmarks can capture relevant device properties.
4) We make all benchmarks and the build system available as open-source on GitHub to encourage community contributions.

The remainder of this paper is organized as follows: In Section II, we give an overview of existing FPGA benchmark suites. In Section III, we introduce the benchmarks in HPCC

978-1-6654-1592-7/20/$31.00 ©2020 IEEE
DOI 10.1109/H2RC51942.2020.00007

FPGA in more detail and briefly discuss the contained benchmarks and the configuration options provided for the base runs. In Section IV we build the benchmarks for different FPGA architectures and evaluate the results to show the potential of the proposed configurable base runs. Finally, in Section V, we draw conclusions and outline future work.

II. Related Work

There already exist several benchmark suites for FPGAs and their HLS frameworks. Most OpenCL benchmark suites like Rodinia [5], OpenDwarfs [6] or SHOC [7] are originally designed with GPUs in mind. Although both GPU and FPGA can be programmed using OpenCL, the design of the compute kernels has to be changed and optimized specifically for FPGA to achieve good performance. In the case of Rodinia this was done [8] for a subset of the benchmark suite with a focus on different optimization patterns for the Intel FPGA (then Altera) SDK for OpenCL. In contrast, to port OpenDwarfs to FPGAs, Feng et al. [6] employed a research OpenCL synthesis tool that instantiates GPU-like architectures on FPGAs. With Rosetta [9], there also exists a benchmark suite that was designed targeting FPGAs using the Xilinx HLS tools from the start. It focuses on typical FPGA streaming applications from the video processing and machine learning domains. The CHO [10] benchmark targets more fundamental FPGA functionality and includes kernels from media processing and cryptography and the low-level generation of floating-point arithmetic through OpenCL, using the Altera SDK for OpenCL.

The mentioned benchmarks often lack possibilities to adjust the benchmarks to the target FPGA architecture easily. Modifications have to be done manually in the kernel code, sometimes many different kernel variants are proposed or the kernels are not optimized at all, making it difficult to compare results for different FPGAs. A benchmark suite that takes a different approach is Spector [11]. It makes use of several optimization parameters for every benchmark, which allows modification and optimization of the kernels for a FPGA architecture. The kernel code does not have to be manually changed, and optimization options are restricted by the defined parameters. Nevertheless, the focus is more on the research of the design space than on performance characterization.

To our best knowledge, there exists no OpenCL benchmark suite for FPGA with a focus on HPC characteristics at the point of writing. In some of the benchmarks, the investigated input sizes are small enough to fit into local memory resources of a single FPGA.

III. HPC Challenge Benchmarks for FPGA

The HPCC benchmark suite [2] consists of seven benchmarks. Three of them are synthetic benchmarks that measure the memory performance for successive accesses (STREAM [12]) and random updates (RandomAccess) as well as the effective network bandwidth of a system (b_eff). Moreover, it contains four applications of varying complexity: Matrix transposition (PTRANS), 1D Fast Fourier Transformation (FFT), matrix multiplication (GEMM) and High Performance LINPACK (HPL).

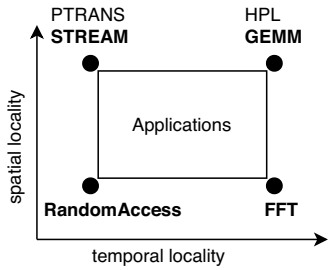

Fig. 1. Visualization of the spatial and temporal locality of memory accesses for the HPCC benchmarks based on the illustration in [2].

A central concept of the HPCC benchmark suite is to characterize the performance of HPC applications by their memory access patterns as it is shown in Figure 1. The applications were chosen to represent border cases of spatial and temporal locality in memory accesses such that the performance for all other applications can be estimated based on these border cases. The achieved performance in CPU systems is thus highly dependent on the memory and cache architecture. In this paper we will focus on four of the benchmarks to represent and discuss all four border cases: STREAM, RandomAccess, FFT and GEMM.

When benchmarking FPGA boards with this concept, it is crucial to note that even for a given board, the memory hierarchy is not fixed. While the off-chip memory itself and typically parts of the memory controllers are fixed hardware components, local buffers or caches, address generators, prefetchers, and data buses inside the FPGA are provided by the BSP or generated by the SDK depending on the benchmark implementation. Thus, in contrast to the CPU version of HPCC, HPCC FPGA does not only measure hardware properties of a given memory interface but rather also the ability of the tools to optimize the memory hierarchy and compute pipelines of a kernel for the specific pattern to provide good performance of the calculation on an FPGA.

Another aspect of the HPCC benchmark suite is the distinction between two different runs:

- *Base runs* are done with the provided reference implementations.
- *Optimized runs* are done with implementations that use architecture-specific optimizations.

The goal of HPCC FPGA, as presented in this article, is to provide implementations for the base runs that are reasonably optimized for Intel and Xilinx FPGAs and their SDKs. With current FPGA execution models, such optimizations necessarily include adaptations to the available resources, in particular the number of physical memory interfaces and configurable local memory. Therefore, the base implementation exposes defined configuration parameters for such customization, without pre-empting the full flexibility that is reserved for optimized runs with manual code changes and target architecture- or

TABLE I
COMPARISON OF THE HPCC BENCHMARKS AND THEIR MEMORY ACCESS PATTERNS ON CPU AND FPGA

Benchmark	CPU	FPGA	
		Global Memory	Local Memory
STREAM	linear	linear	linear
RandomAccess	random	random	linear
FFT	strided, T	linear	strided, T
GEMM	blocked, linear, T	blocked, linear	strided, linear, T

T = temporal locality

Listing 1
SIMPLIFIED KERNEL LOGIC OF THE STREAM BENCHMARK

```
for(uint i=0; i<array_size; i+=BUFFER_SIZE){
    float buffer[BUFFER_SIZE];
    for (uint k=0; k<BUFFER_SIZE; k++)
        buffer[k] = scalar * in1[i + k];
    if (second_input)
        for (uint k=0;k<BUFFER_SIZE; k++)
            buffer[k] += in2[i + k];
    for (uint k=0;k<BUFFER_SIZE; k++)
        out[i + k] = buffer[k];
}
```

SDK-specific designs. The presented adjustable parameters for the base runs are a subject of discussion and might be changed with the evolution of the FPGA architectures and toolchains.

In Table I the memory access patterns for the benchmarks contained in the first version of *HPCC FPGA* are given for CPU and the FPGA base implementations proposed in this paper. For FPGA, the memory is further divided into global and local memory representing the DDR RAM or Block RAM (BRAM) and registers, respectively. Spatial locality is represented by the *linear* and *blocked* access pattern, while temporal locality is separately indicated with a *T*. Since it is possible to partially define the memory hierarchy used for the application, the FPGA designs attempt to move the strided memory accesses with low spatial locality into the local memory and increase the temporal locality if possible using blocked algorithms. The global memory again is accessed in a blocked fashion to increase spatial locality and decrease temporal locality.

The benchmark suite is open source and publicly available on GitHub[1].

A. Common Build Setup

The HPCC FPGA benchmark suite is set up to create one host binary and FPGA bitstream per benchmark, with source code structure, build process, and benchmark execution very similar for all benchmarks. For usability, HPCC FPGA adopts the following approaches:

- Usage of the same build system for all benchmarks (CMake) offers a unified user experience during the build process and for the modification of the configuration parameters.
- Integrated tests allow checking functional correctness of the configuration using emulation before actual synthesis.

Every benchmark can be configured using several parameters that are specified during the build configuration with CMake's -D flag. They directly affect the performance of the kernel and thus need to be given together with the benchmark results to allow a more meaningful comparison. These parameters are summarized in Table II for all four benchmarks. An essential parameter for all benchmarks is `NUM_REPLICATIONS` which allows to automatically create copies of the kernels. It is heavily used by the four presented implementations to scale the benchmarks to the available resources of the target boards.

B. STREAM Benchmark

The goal of the STREAM benchmark is to measure the sustainable memory bandwidth in $\mathrm{GB/s}$ of a device using four different vector operations: *Copy*, *Scale*, *Add* and *Triad*. All kernels are taken from the STREAM benchmark[2] v5.10 and thus slightly differ from the kernels proposed in the HPCC article [2]. The benchmark will sequentially execute the operations given in Table III. *PCI Read* and *PCI Write* are no OpenCL kernels but represent the read and write of the arrays to the device memory. The benchmark will output the maximum, average, and minimum times measured for all of these operations. For the calculation of the memory bandwidth, the minimum times will be used.

The arrays A, B, and C are initialized with a constant value over the whole array. This allows us to validate the result by only recalculating the operations with scalar values. The error is calculated for every value in the arrays and must be below the machine epsilon $\epsilon < ||d - d'||$ to pass the validation.

A simplified version of the code is given in Listing 1. The OpenCL kernel of the benchmark combines all four described compute kernels in a single function. Since on FPGAs the source code is translated to spatial structures that take up resources on the device, a single combined kernel allows for best reuse of those resources. The computation is split into blocks of a fixed length, which makes it necessary that the arrays have the length of a multiple of the block size. In the first inner loop, the input values of the first input array `in1` are loaded into a buffer located in the local memory of the FPGA. While they are loaded, they are multiplied with a scaling factor `scalar`. This allows the execution of the *Copy* and *Scale* operation. In the case of *Copy*, the scaling factor is set to 1.0. In the second loop, the second input `in2` is only added to the buffer, if a flag is set. Together with the first loop, this makes it possible to recreate the behavior of the *Add* and *Triad* operation. In the third loop, the content of the buffer is stored in the output array `out` located in global memory.

`DATA_TYPE` and `VECTOR_COUNT` can be used to define the data type used within the kernel. If `VECTOR_COUNT`

[1]https://github.com/pc2/HPCC_FPGA

[2]https://www.cs.virginia.edu/stream/

TABLE II
THE CONFIGURATION OPTIONS THAT ARE EXPOSED TO THE USER TO MODIFY THE KERNEL GENERATION OF THE BENCHMARKS

Benchmark	Parameter	Description
All benchmarks	NUM_REPLICATIONS	Replicates the benchark kernels the given number of times.
STREAM	DATA_TYPE	Data type used for host and device code.
	VECTOR_COUNT	If VECTOR_COUNT > 1, OpenCL vector types of the given size are used in the device code instead of scalar values. Can be used to match the data size with the width of the memory interface.
	GLOBAL_MEM_UNROLL	Loop unrolling factor for the inner loops in the device code. Similar effect to VECTOR_COUNT.
	DEVICE_BUFFER_SIZE	Number of values that are stored in the local memory. Implicitly affects size of global memory bursts.
RandomAccess	DEVICE_BUFFER_SIZE	Number of values that are stored in the local memory. Can be used to relax dependencies between subsequent accesses to global memory in trade-off with a higher error rate.
FFT	LOG_FFT_SIZE	Logarithm of the FFT size that is calculated. Higher FFT sizes will consume more resources but achieve higher compute performance.
GEMM	DATA_TYPE	Data type used for host and device code.
	BLOCK_SIZE	Size of the symmetric matrix block that will be stored in local memory. Should have a sufficient size to allow full utilization of global memory read and write bursts and the calculation pipeline.
	GEMM_SIZE	Size of the symmetric matrix block that will be stored in registers. This will affect the amount of Digital Signal Processors (DSPs) used in the implementation and the actual calculation performance.
	GLOBAL_MEM_UNROLL	Used to specify the amount of unrolling done in loops that access the global memory to match the width of the memory interface.

TABLE III
THE STREAM BENCHMARK KERNELS THAT ARE CALCULATING ON THREE ARRAYS A, B AND C AND THE PCIE TRANSFERS LISTED IN ORDER OF EXECUTION

Name	Kernel Logic
PCI Write	write arrays to device
Copy	$C[i] = A[i]$
Scale	$B[i] = j \cdot C[i]$
Add	$C[i] = A[i] + B[i]$
Triad	$A[i] = j \cdot C[i] + B[i]$
PCI Read	read arrays from device

is greater than 1, OpenCL vector types of the given length are used. Since some of the STREAM operations show an asymmetry between reads and writes from global memory, the best performance can be achieved if a single bank is used for all three arrays. DDR memory bandwidth is not bidirectional, so with a sufficient memory interface width, each of the three phases can fully utilize all available bandwidth of the global memory bank. Intermediate results are stored in a local memory buffer that can be modified in size over a configuration parameter. This makes STREAM bandwidth bound by the global memory and also slightly resource bound by the BRAM. The kernel has to be replicated to fully utilize all available memory banks and at the same time the local memory buffer size has to be increased to achieve larger burst sizes and decrease the impact of memory latency.

C. RandomAccess Benchmark

The random access benchmark measures the performance for non-consecutive memory accesses expressed in the performance metric Giga Updates Per Second (GUPS). It updates values in a data array $d \in \mathbb{Z}^n$ such that $d_i = d_i \oplus a$ where $a \in \mathbb{Z}$ is a value from a pseudo random sequence. n is defined to be a power of two. As data type, $64\,\text{bit}$ integers are used. Since only operations in a finite field are used to update the values, the correctness of the updates can easily be checked by executing a reference implementation on the host side on the resulting data using the same pseudo random sequence. The incorrect items are counted, and the error percentage is calculated with $\frac{error}{n} \cdot 100$. An error of $< 1\%$ has to be accomplished to pass the validation. Hence, update errors caused by concurrent data accesses are tolerated to some degree. The performance of the implementation is mainly bound by the memory bandwidth and latency. So the kernel should be replicated to utilize all available memory banks.

The benchmark requires $4 \cdot n$ updates on the array of length n using two pipelines. The first pipeline reads data items from global memory. Therefore, it uses a pseudo random sequence generator that is implemented in every kernel replication. The index of the value will be extracted from the random number. Since every replication of the kernel is in charge of a distinct range of addresses that is placed in their memory bank, the kernel has to check if the calculated address is actually within its range. Only then the value will be loaded from global memory and stored in a local memory buffer which size can be configured over the DEVICE_BUFFER_SIZE parameter. In the second pipeline, the data in local memory is used to update the values in global memory. With this approach, read after write dependencies will be relaxed which prevents

pipeline stalls caused by pending load operations. On the one hand, this will improve the performance of the kernel. On the other hand, this mechanism will lead to errors if the same memory address is loaded multiple times into the buffer. Since the benchmark allows a small amount of errors, changing the buffer size can be used as a trade-off between benchmark error and performance.

D. Fast Fourier Transformation Benchmark (FFT)

In the FFT benchmark, a batch of 1D FFTs of a predefined size – that can be specified using the `LOG_FFT_SIZE` parameter – is calculated. A batch of FFTs is used to increase the overall execution time of the benchmark to decrease measurement errors and to better utilize the kernel pipeline. The benchmark kernel is based on a reference implementation for the Intel OpenCL FPGA SDK included in version 19.4.0 and slightly modified to also allow execution on Xilinx FPGAs. The calculation is done with complex single precision floating point values. All stages of the FFT calculation are fully unrolled and the current data is stored in a shift register that is shared between the stages. This allows the implementation of the algorithm in a single pipeline but also limits the maximum size of the FFT by the available FPGA resources and SDK capabilities. So the benchmark implementation is memory as well as resource bound. Larger FFT sizes require more DSPs for the calculations, and logic resources and BRAM to implement the shift register. Since reads and writes from global memory are symmetric, the kernel offers the best performance if the input and output data can be stored in different memory banks. So the kernel replications should be set at most to half of the available memory banks. Also, the best trade-off between kernel replication and FFT size has to be used to achieve the best performance.

For the performance metric GFLOP/s, the floating point operations of this benchmark are calculated as $5 \cdot n \cdot ld(n)$ for an FFT of dimension n. The result of the calculation is checked by calculating the residual $\frac{||d-d'||}{\epsilon \cdot ld(n)}$ where ϵ is the machine epsilon, d' the result from the reference implementation and n the FFT size.

E. Matrix Multiplication Benchmark (GEMM)

The GEMM benchmark implements a matrix-matrix multiplication similar to the GEMM routines in the BLAS library. It calculates $C = \alpha \cdot A \cdot B + \beta \cdot C$ where $A, B, C \in \mathbb{R}^{n \times n}$ and $\alpha, \beta \in \mathbb{R}$. For the performance metric GFLOP/s, the floating point operations of this benchmark are calculated as $2 \cdot n^3$. The result is verified by calculating the residual $\frac{||C-C'||}{\epsilon \cdot n \cdot ||C||_F}$ where ϵ is the machine epsilon and C' the result of the reference implementation. The implementation is based on a matrix multiplication design for Intel Stratix 10 [13] and simplified to make it compatible with a broader range of devices. The kernel creates a memory hierarchy by doing the following data movements:

- A blocked matrix multiplication in global memory
- A blocked matrix multiplication in local memory (BRAM)
- A fully unrolled matrix multiplication in registers

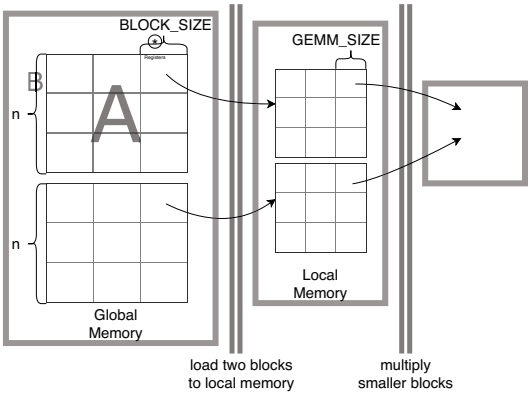

Fig. 2. Visualization of the data movements that are performed by the GEMM kernel.

A visualization of the data movements is given in Figure 2. The matrices are divided into blocks of a fixed size that can be specified with `BLOCK_SIZE` parameter. The kernel calculates the result of a single block in C by sequentially executing two pipelines. In the first pipeline, a single block of A and B is loaded into a local memory buffer. In the second pipeline the loaded blocks are used to calculate an intermediate result again with a blocked approach. This time the block size is defined by the parameter `GEMM_SIZE`. The matrix multiplication for this block will be fully unrolled which allows to initiate the multiplication of a whole block every clock cycle. Both pipelines will be executed sequentially to calculate a result of a single block of $A + B$. A third pipeline loads a block of C and calculates the final result for a block of the output matrix. So the data is loaded block-wise from global memory to local memory and from there in smaller blocks to registers to perform the actual computation. Writing the data back to global memory does not have a huge impact on the overall performance for huge matrices because this just has to be done every $\frac{n}{BLOCK\_SIZE}$ iterations.

The host code will divide the output matrix into rows of blocks of size `BLOCK_SIZE` and distribute the calculation over all kernel replications. Since the kernel is calculation bound, the second pipeline should be scaled as large as possible or the kernel should be replicated to increase the parallelism of the calculation. This makes the implementation resource bound in terms of DSPs for the calculation pipeline and BRAM for the local memory buffer. Loading and storing data from global memory just takes up a minor role in the implementation. Thus, the kernel replications should be used to fill the FPGA instead of utilizing all memory banks.

IV. Benchmark Execution and Evaluation

In the following, we synthesize and execute all four benchmarks, collect first benchmark results for the proposed base implementations, and evaluate the results to performance models.

A. Evaluation Environment and Configuration

The benchmarks were synthesized and tested on three different boards: *Nallatech 520N*, *Intel PAC D5005* and *Xilinx Alveo U280*. The Nallatech 520N boards are connected to the host node via x8 PCIe 3.0 and are equipped with Intel Stratix 10 GX2800 with access to four banks of DDR4 SDRAM x 72 bit with 8 GB per bank and a transfer rate of 2400MT/s. With the BSP version 19.4.0 and SDK version 19.4.0 the most recent versions available at the time of writing are used for synthesis and execution.

The Intel PAC D5005 board hosts a Stratix 10 SX2800 FPGA and is connected to the host with x16 PCIe 3.0. The board contains a DDR4 memory infrastructure similar to the other boards, which is however, not used for these experiments. Instead, a reference design BSP (18.1.2_svm) was used, that offers direct access to the host's memory using Shared Virtual Memory (SVM) by building upon the the Intel Acceleration Stack (IAS) [14] version 1.2. The OpenCL kernel compilation was performed with the SDK version 19.4.0.

The Alveo U280 boards are equipped with the XCU280 FPGA. The board is connected to an Intel Xeon Gold 6234 CPU over x8 PCIe 4.0 and equipped with two banks of DDR4 SDRAM x 72 bit with 16 GB per bank and a transfer rate of 2400 MT/s. Moreover the FPGA is equipped with $8\,\mathrm{GB}$ of High Bandwidth Memory 2 (HBM2) split over 32 banks. Xilinx Vitis is used in version 2019.2 and the shell version is 2019.2.3. The host code is compiled with GCC 7.4.0 and CMake 3.18 is used for the configuration and build of the benchmarks. The CPU has access to $108\,\mathrm{GB}$ of main memory.

For the Intel FPGAs, the host node is a two-socket system equipped with Intel Xeon Gold 6148 CPUs and $192\,\mathrm{GB}$ of DDR4-2666 main memory. The host codes are compiled with GCC 8.3.0 and CMake 3.15.3 is used for the configuration and build of the benchmarks.

The used synthesis parameters for all benchmarks are given in Table IV and the resulting resource usage for each benchmark is given in Table V. A straight forward comparison of the resource usage between the FPGA boards is not possible because hardware and software architectures are different. Nevertheless, in the table a general overview of the resource usage is given by looking at the basic resource elements of an FPGA: The Lookup Tables (LUTs), Registers, BRAM and DSPs. The table only takes into account the resources directly used for the kernels. Next to the absolute value, the percentage of the used resources relative to the available resources is given. Absolute values and ratios of the resource usage are taken directly from the reports generated by the HLS tools.

B. STREAM and RandomAccess

The measured performance results for STREAM and RandomAccess are given in Table VI. Single precision floating point numbers are used in the STREAM benchmark. For the size of the STREAM data array we used 2^{29} items, which corresponds to $2\,\mathrm{GB}$ of data per data array. This is the largest power of two that fits into the $8\,\mathrm{GB}$ HBM2 of the Alveo U280 board.

The theoretical peak performance for DDR memory is $19.2\,\mathrm{GB/s}$ per bank for both the 520N and the U280 boards. That said, the STREAM benchmark achieves an efficiency between 87.2% and 90.1% of the theoretical peak performance for both devices for all four operations. The HBM2 of the U280 board offers a theoretical peak bandwidth of $460\,\mathrm{GB/s}$, so the benchmark achieves an efficiency of 82.4% on this board. One reason for the lower efficiency is the use of smaller local memory buffers because of the high number of kernel replications. This will also lead to shorter read and write bursts which directly affect the memory bandwidth.

The full-duplex PCIe connection of the PAC SVM board eliminates the need for a large local memory buffer. Instead, it is set to the same size than the unrolling (`GLOBAL_MEM_UNROLL`) which fully unrolls the three inner read- and write- pipelines and results in a single pipeline that allows to simultaneously read and write from global memory. For the *Copy* and *Scale* operation the kernel achieves more than $20\,\mathrm{GB/s}$. The *Add* and *Triad* operations need two input parameters, which leads to the PCIe read channel being the bottleneck and only half of the write channel's bandwidth can be utilized.

For the Random Access benchmark, a data array of 2^{29} items, which is a total of $4\,\mathrm{GB}$, is equally split onto the available memory banks. All kernels have to calculate all addresses and only update the value if it is placed within the range in the data array they are assigned to. This leads to a compute-bound implementation for a high number of kernel replications since the pipeline stalls caused by memory accesses per kernel will decrease. Still, the amount of random numbers that have to be generated stays the same. So the maximum achievable updates per second are limited by the kernel frequency. The measured performance will show a higher gap to the maximum performance for designs with only a small amount of replications because the random memory accesses will consume a considerable amount of time. The RandomAccess benchmark shows huge performance differences between the used boards. One reason for that is the difference in the kernel design that allows the creation of a single pipeline in case of the 520N board using the Intel-specific *ivdep* pragma. This optimization also slightly decreased the calculation error that is introduced by the buffer. Nevertheless, for the PAC SVM tests this optimization had to be disabled because with 98% the error drastically exceeds the allowed range. For Xilinx there is to our knowledge no optimization flag similar to *ivdep*.

C. FFT

The FFT benchmark calculates the 1D FFT of different sizes specified in the configuration in a batched manner. The batch size is chosen such that the total size of the data is $2\,\mathrm{GB}$. This allows the kernel to fill the pipeline and hide the latency of the calculation. Eight values are loaded and stored to global memory per clock cycle and the calculation is implemented in a single pipeline. With every value, `LOG_FFT_SIZE` complex floating-point multiplications are calculated, which

TABLE IV
SYNTHESIS CONFIGURATIONS OF ALL BENCHMARKS

Benchmark	Parameter	520N	U280 DDR	U280 HBM2	PAC SVM
STREAM	NUM_REPLICATIONS	4	2	32	1
	DATA_TYPE	float	float	float	float
	GLOBAL_MEM_UNROLL	1	1	1	1
	VECTOR_COUNT	16	16	16	16
	DEVICE_BUFFER_SIZE	4,096	16,384	2,048	1
RandomAccess	NUM_REPLICATIONS	4	2	32	1
	DEVICE_BUFFER_SIZE	1	1,024	1,024	1,024
FFT	NUM_REPLICATIONS	2	1	15	1
	LOG_FFT_SIZE	17	9	5	17
GEMM	NUM_REPLICATIONS	5	3	3	5
	DATA_TYPE	float	float	float	float
	GLOBAL_MEM_UNROLL	16	16	16	16
	BLOCK_SIZE	512	256	256	512
	GEMM_SIZE	8	8	8	8

TABLE V
RESOURCE USAGE OF THE SYNTHESIZED BENCHMARK KERNELS

Benchmark	Board	LUTs	Registers	BRAM	DSPs	Frequency [MHz]
STREAM	520N	176,396 (25%)	449,231 (25%)	4,029 (34%)	128 (2%)	316.67
	U280 DDR	20,832 (1.90%)	39,002 (1.39%)	558 (34.19%)	160 (1.78%)	300.00
	U280 HBM2	331,904 (20.69%)	574,976 (27.24%)	1,408 (77.70%)	2,560 (28.38%)	370.00
	PAC SVM	103,628 (12%)	244,354 (12%)	548 (5%)	32 (< 1%)	346.00
RandomAccess	520N	115,743 (18%)	253,578 (18%)	489 (4%)	14 (< 1%)	329.17
	U280 DDR	7,256 (0.65%)	11,716 (0.50%)	38 (2.23%)	14 (0.16%s)	446.00
	U280 HBM2	116,096 (10.68%)	187,456 (8.76%)	608 (33.55%)	224 (2.48%)	450.00
	PAC SVM	103,397 (12%)	225,293 (12%)	535 (5%)	0 (0%)	322.00
FFT	520N	276,676 (36%)	724,790 (36%)	4,177 (36%)	1,414 (25%)	413.34
	U280 DDR	83,494 (7.39%)	168,150 (7.19%)	39 (2.28%)	672 (7.46%s)	248.00
	U280 HBM2	602,125 (54.13%)	941,404 (42.18%)	405 (22.35%)	5,280 (58.58%)	254.00
	PAC SVM	192,189 (22%)	480,285 (22%)	2,147 (18%)	707 (12%)	348.00
GEMM	520N	275,754 (36%)	861,277 (36%)	8,860 (76%)	3,398 (59%)	160.42
	U280 DDR	568,558 (51.87%)	441,602 (19.43%)	666 (43.11%)	7,683 (85.23%)	100.00
	U280 HBM2	499,002 (42.64%)	920,127 (38.70%)	666 (36.71%)	7,683 (85.18%)	236.00
	PAC SVM	299,427 (33%)	829,802 (33%)	9,041 (77%)	3,398 (59%)	225.00

TABLE VI
MEASUREMENT RESULTS FOR STREAM AND RANDOMACCESS

Benchmark	520N	U280 HBM2	U280 DDR	PAC SVM
STREAM Copy [GB/s]	67.01	377.42	33.94	20.15
STREAM Scale [GB/s]	67.24	365.80	33.92	20.04
STREAM Add [GB/s]	68.90	374.03	34.58	15.04
STREAM Triad [GB/s]	68.90	378.88	34.57	15.12
STREAM PCIe read [GB/s]	6.41	6.66	5.68	–
STREAM PCIe write [GB/s]	6.32	6.03	5.47	–
RandomAccess [MUOP/s]	245.0	128.1	40.3	0.5
RandomAccess Error	0.0099%	0.0106%	0.0106%	0.0106%

TABLE VII
MEASUREMENT RESULTS FOR GEMM AND FFT

Benchmark	520N	U280 HBM2	U280 DDR	PAC SVM
FFT [GFLOP/s]	349.45	576.20	78.26	119.66
FFT [GB/s]	65.78	368.77	27.83	22.54
FFT Error	7.1e-01	5.4e-01	3.9e-01	7.1e-01
GEMM [GFLOP/s]	708.95	603.86	266.91	739.59
GEMM norm [GFLOP/s]	88.39	85.29	88.97	65.74
GEMM Error	6.0e-07	2.0e-06	2.0e-6	6.0e-07

corresponds to five single-precision floating-point operations. Eight values are loaded from global memory per clock cycle so the expected performance of the kernel can be modelled with the Equation 1.

$$p_{FFT} = 5 \cdot s \cdot f \cdot r \cdot 8 \tag{1}$$

where s is LOG_FFT_SIZE, r is NUM_REPLICATIONS and f the minimum of the kernel frequency and the memory interface frequency.

The model shows, that the FFT size and the number of kernel replications have a similar impact on the expected performance. Since every kernel exclusively needs two memory banks, the kernel replications are chosen after the number of available memory banks on the FPGA. In a second step the FFT size is increased to fill the FPGA. The configuration for the U280 board with DDR is an exception in this regard because it does not fully utilize the FPGA resources which again shows a weakness of the current FFT implementation: The implementation makes use of a large shift register to store and delay data that is read from global memory. For Xilinx, the largest supported shift register size is 1024, which is too small for FFT sizes for 2^{10} and beyond. Also the achieved kernel frequencies are too low to fully utilize the memory on the Xilinx board which has to be considered when comparing the results to STREAM.

Besides that, the kernel shows a high efficiency of more than 85% for all FPGAs which is close to the STREAM results. Also the PAC board shows a very high memory bandwidth for the kernel. This indicates that large memory bursts have a huge impact on the achieved memory bandwidth over SVM.

D. GEMM

Table VII contains the results for the GEMM and FFT benchmark. We used single precision floating point as data type because this data type is well supported among all used devices. The GEMM benchmark uses 4096×4096 matrices for the calculation. This is the biggest matrix size that fits into a single HBM2 bank. The actual calculation is done on 8×8 matrices for all boards defined by the `GEMM_SIZE` parameter. So the kernel can initialize the calculation of 1024 floating-point multiplications and additions per clock cycle. This is the largest value for `GEMM_SIZE` that fits on the FPGAs. The kernel replications and `BLOCK_SIZE` is chosen to utilize the FPGA resources as best as possible while still providing an acceptable kernel frequency.

Because the kernel contains three pipelines that are executed sequentially, the total execution time of the kernel can be modelled by the sum of the three pipelines as given in Equation 2.

$$t_{exe} = \frac{b^2}{u \cdot f_{mem}} + \frac{b^3}{g^3 \cdot f_k} + \frac{b^2}{u \cdot \frac{n}{b} \cdot f_{mem}} \tag{2}$$

where

b : `BLOCK_SIZE` r : `NUM_REPLICATIONS`
u : `GLOBAL_MEM_UNROLL` g : `GEMM_SIZE`

Moreover, n is the total matrix size, f_k the kernel frequency and f_{mem} the frequency of the memory interface.

For the platforms with on-board memory resources (DDR and HBM), the measured performance deviates not more than 1.8% from the model performance. Only the SVM platform performs 23% slower than the model. Simultaneous memory accesses of the different kernel replications as well as the decreased memory burst sizes caused by the blocked reads are likely bottlenecks during the global memory access phases. However, due to the higher clock frequency, the overall performance of the SVM design is still slightly higher than the very similar design on 520N DDR.

Since the GEMM benchmark is calculation bound, the performance is highly dependent on the kernel frequency and lesser on the memory interface. We also provide a performance result normalized to $100\,\mathrm{MHz}$ and a single kernel replication in the table to make the performance easier to compare between all boards. The normalized performance of the 520N and U280 is close together which shows that the tools for both boards are able to generate hardware from the code with a similar performance efficiency. For the U280 with DDR memory we had to set the target frequency to $100\,\mathrm{MHz}$ to achieve a successful synthesis. Reasons for that are routing problems and not satisfied timing constraints because of the additional logic needed for the DDR memory interface. In contrast, when using HBM2, place and route of three replications worked seamlessly and resulted in a considerably higher clock frequency.

V. Conclusion

In this paper, we proposed *HPCC FPGA*, a novel FPGA OpenCL benchmark suite for HPC. Therefore, we provide configurable OpenCL base implementations and host codes of four selected benchmarks of the well-established HPCC benchmark suite. We showed that the configuration options allow the generation of efficient benchmark kernels for Xilinx and Intel FPGAs using the same source code without manual modification. We executed the benchmarks on three FPGAs with four different memory setups and compared the results with simple performance models. The evaluation showed that, for most of the benchmarks, the measured performance is close to the modelled peak performance under resource or global memory constraints of the FPGA board. In these cases, the benchmarks can start to serve as a deployment criterion for FPGA accelerated systems. Nevertheless, the evaluation also showed that some of the base implementations are unable to fully utilize the available resources or produce low kernel frequencies which has to be improved through further optimization and changes in the design.

Hence, it is important to discuss the base implementations and configuration options with the community to create a valuable and widely accepted FPGA performance characterization tool for HPC. We made the code open-source and publicly available to simplify and encourage contributions to future versions of the benchmark suite.

Acknowledgements

The authors gratefully acknowledge the support of this project by computing time provided by the Paderborn Center for Parallel Computing (PC2). We also thank Xilinx for the donation of an Alveo U280 card, Intel for providing a PAC D5005 loaner board and access to the reference design BSP with SVM support, and the Systems Group at ETH Zurich as well as the Xilinx Adaptive Compute Clusters (XACC) program for access to their Xilinx FPGA evaluation system.

REFERENCES

[1] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, September 2006. [Online]. Available: http://doi.acm.org/10.1145/1186736.1186737

[2] J. J. Dongarra and P. Luszczek, "Introduction to the HPCChallenge Benchmark Suite," Defense Technical Information Center, Fort Belvoir, VA, Tech. Rep., Dec. 2004. [Online]. Available: http://www.dtic.mil/docs/citations/ADA439315

[3] TOP500.org, "TOP500 Supercomputer Sites," https://www.top500.org/lists/2019/11/, accessed: 2020-03-31.

[4] G. Juckeland, W. Brantley, S. Chandrasekaran, B. Chapman, S. Che, M. Colgrove, H. Feng, A. Grund, R. Henschel, W.-M. W. Hwu, H. Li, M. S. Müller, W. E. Nagel, M. Perminov, P. Shelepugin, K. Skadron, J. Stratton, A. Titov, K. Wang, M. van Waveren, B. Whitney, S. Wienke, R. Xu, and K. Kumaran, "Spec accel: A standard application suite for measuring hardware accelerator performance," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham: Springer International Publishing, 2015, pp. 46–67.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *2009 IEEE Int. Symp. on Workload Characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[6] W.-c. Feng, H. Lin, T. Scogland, and J. Zhang, "OpenCL and the 13 dwarfs: A work in progress," in *Proc. Int. Conf. on Performance Engineering (ICPE)*, ser. ICPE '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 291–294.

[7] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *Proc. of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ser. GPGPU-3. New York, NY, USA: Association for Computing Machinery, 2010, p. 63–74.

[8] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 409–420.

[9] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software-Programmable FPGAs," *Int. Symp. on Field-Programmable Gate Arrays (FPGA)*, Feb 2018.

[10] G. Ndu, J. Navaridas, and M. Luján, "Cho: Towards a benchmark suite for OpenCL fpga accelerators," in *Proceedings of the 3rd International Workshop on OpenCL*, ser. IWOCL '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2791321.2791331

[11] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An OpenCL FPGA Benchmark Suite," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, pp. 141–148.

[12] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," 1995.

[13] P. Gorlani, T. Kenter, and C. Plessl, "OpenCL Implementation of Cannon's Matrix Multiplication Algorithm on Intel Stratix 10 FPGAs," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 99–107.

[14] E. Luebbers, S. Liu, and M. Chu, "Simplify software integration for FPGA accelerators with OPAE," White Paper, Intel, 2019. [Online]. Available: https://01.org/sites/default/files/downloads/opae/open-programmable-acceleration-engine-paper.pdf