IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 August 2022, accepted 25 August 2022, date of publication 1 September 2022, date of current version 14 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3203566

APPLIED RESEARCH

FER: A Benchmark for the Roofline Analysis of FPGA Based HPC Accelerators

ENRICO CALORE AND SEBASTIANO FABIO SCHIFANO
Università degli Studi di Ferrara, 44121 Ferrara, Italy
INFN Ferrara, 44122 Ferrara, Italy

Corresponding author: Enrico Calore (enrico.calore@fe.infn.it)

This work was supported in part by the European Union's H2020 Research and Innovation Program through the EuroEXA under Agreement 754337, and in part by the COKA and COSA INFN Projects.

ABSTRACT Nowadays, the use of hardware accelerators to boost the performance of HPC applications is a consolidated practice, and among others, GPUs are by far the most widespread. More recently, some data centers have successfully deployed also FPGA accelerated systems, especially to boost machine learning inference algorithms. Given the growing use of machine learning methods in various computational fields, and the increasing interest towards reconfigurable architectures, we may expect that in the near future FPGA based accelerators will be more common in HPC systems, and that they could be exploited also to accelerate general purpose HPC workloads. In view of this, tools able to benchmark FPGAs in the context of HPC are necessary for code developers to estimate the performance of applications, as well as for computer architects to model that of systems at scale. To fulfill these needs, we have developed FER (FPGA Empirical Roofline), a benchmarking tool able to empirically measure the computing performance of FPGA based accelerators, as well as the bandwidth of their on-chip and off-chip memories. FER measurements enable to draw Roofline plots for FPGAs, allowing for performance comparisons with other processors, such as CPUs and GPUs, and to estimate at the same time the performance upper-bounds that applications could achieve on a target device. In this paper we describe the theoretical model on which FER relies, its implementation details, and the results measured on Xilinx Alveo accelerator cards.

INDEX TERMS Accelerator, benchmark, FPGA, HPC, performance, roofline.

I. INTRODUCTION

In the context of HPC (High Performance Computing) systems the use of highly parallel hardware accelerators to boost the performance of applications is nowadays a very common option, adopted by a large and increasing share of HPC systems listed in the Top500 ranking [1]. In this sector, GPUs are definitively the most common accelerators, whereas FPGAs are hardly, or even not, used at all. Despite this, some data centers have recently started to adopt FPGAs to speed-up network interconnects [2], and specific workloads [3] such as Machine Learning (ML) inference algorithms [4], [5]. Given the rapidly increasing use of ML methods in several application fields, and the interest in reconfigurable architectures, which is rising in the HPC community since several years [6], [7], [8], we may expect FPGAs to become a more common option, as accelerators, for next generations of HPC systems.

In the past, several reasons have prevented this. First, FPGAs were not designed to provide high floating-point (FP) computing performance [7], whereas typical HPC workloads usually require double-precision (DP) and single-precision (SP) FP computations. Secondly, FPGA programming could be a very time consuming process, requiring the use of specific hardware programming skills and the use of programming languages not common among HPC developers communities [9]. Thirdly, the code written for one FPGA could hardly run across different devices without a complete re-design, causing serious portability problems not acceptable for a wide set of HPC applications, for which even the porting to GPUs had been a long and suffered process [10].

The associate editor coordinating the review of this manuscript and approving it for publication was Shadi Alawneh.

However, more recently, these barriers started to fade thanks to improvements in hardware architectures and programming frameworks. In fact, latest generations of FPGAs integrate thousands of programmable DSPs (Digital Signal Processors) able to implement SP- and DP-FP operations [11], [12], [13], and may also embed custom FP DSP blocks. This is leading to devices able to reach a performance in the same order of magnitude as commodity HPC processors (i.e., TFLOP/s), and in some cases able to deliver a better energy-efficiency [14], [15]. At the same time, the recent improvements of synthesis tools, and the development of new programming approaches such as HLS (High Level Synthesis) [16], allow programmers to develop codes using high level languages. As an example, OpenCL [15] could be used, as well as plain C/C++ annotated with *pragma* directives to guide the compiler to automatically map the code onto FPGA hardware resources [17]. These approaches are very similar to those (e.g., OpenMP and OpenACC) commonly used by HPC developers to target multi-core CPUs and other accelerators, which are also able to guarantee a fair level of code portability [18].

All the above improvements combined with the urging quest for higher energy-efficiency and lower latency interconnects in exascale HPC systems, are leading to a significant increase in the interest towards heterogeneity and specialized computing in the form of reconfigurable accelerators [19]. This makes the use of FPGAs very attractive as they allow to scale-out resources by enabling distributed computing, and can be programmed to be network-capable processors implementing custom interconnects featuring low-latency communications without involving the CPU control [20].

First prototypes of FPGA accelerated HPC systems are already being designed and deployed. One example is the Alveo FPGA Cluster installed at ETH Zurich in the context of the Xilinx Adaptive Compute Clusters (XACC) initiative, using commodity hardware to support novel research in adaptive compute acceleration for HPC. Another example is the EU-H2020 EuroEXA Project, which has developed a HPC system prototype with custom hardware, adopting FPGA based accelerators for both computing and networking [20].

Consequently, as a future scenario we may expect next generations of HPC systems to be equipped with FPGA-based accelerators, probably alongside other accelerators, such as GPUs, being programmed with high level languages, possibly based on pragma directives, allowing to address several kind of different accelerators in a uniformed way [18].

In this context, application developers need to estimate the performance achievable on target FPGAs, to decide whether an application kernel is worth to be ported, or which FPGA better fits its computing requirements. At the same time, system architects and engineers need to estimate the performance of a single FPGA, to feed performance models to tune, balance and optimize the performance at system level [19]. These are in fact the needs that arose, for example, while evaluating the performance of the EuroEXA FPGA-based accelerators and the optimization level of HPC applications ported to its custom architecture.

To tackle these needs, we have developed FER (FPGA Empirical Roofline), a benchmarking tool we have released as *Free Software* [21]. FER is based on a similar principle as ERT (Empirical Roofline Toolkit) [22], but it is specifically designed to target FPGA based accelerators, using C/HLS, to determine their hardware characteristics, allowing for the Roofline analysis of their architectures. It is able to measure an empirical estimate of the peak compute performance achievable on a given FPGA, as well as its on-chip and off-chip memories bandwidth. The choice of C/HLS is strategic, since it allows to measure performance upper-bounds using the same high level programming approach that we expect to be used by most HPC developers, thanks to its approach based on pragma directives, but at the same time it allows for fairly low level optimizations.

This work is an extension of our previous conference paper where we presented a preliminary implementation of the FER benchmark focusing only on measuring the compute performance, and off-chip DDR (Double Data Rate) DRAM bandwidth, of a Xilinx Alveo U250 [23]. Here, we present an improved version of FER allowing to benchmark also on-chip memories, providing a more comprehensive description of its implementation details and of the theoretical model on which it relies. Moreover, we also show how FER could be used on accelerators embedding off-chip HBM (High Bandwidth Memory) [24], running it also on the Xilinx Alveo U50 and U280 data center cards, and show how close third party application kernels can get to the performance upper-bound measured by FER.

As far as we know, FER is the first tool, developed using a *pragma* directives based high level programming paradigm, able to benchmark FPGA based accelerators in the framework of the Roofline Model [25], and the first able to take into account also on-chip memories.

The rest of the paper is organized as follow: in the next section we give an overview of related works; in Sec. III we present the FER benchmark, including the theoretical model on which it relies and its implementation details; in Sec. IV we describe the FPGA based accelerators adopted; in Sec. V we present the experimental results provided by FER, running on such accelerators; and finally in Sec. VI we give our concluding remarks.

II. RELATED WORKS

Several research works have investigated FPGAs performance when used as hardware accelerators, mostly using synthetic benchmarks to estimate the bandwidth of off-chip memories [26], [27], [28], and OpenCL kernels to measure the FPGA computing performance [29], [30], [31]. However, only few tools utilize the Roofline Model, and none assess also the on-chip memories bandwidth.

In [26] is presented the *Shuhai* Verilog benchmark, used to characterize the performance of HBM and DDR off-chip memories embedded in the Xilinx Alveo U280.

This benchmark is mainly meant to guide Verilog and VHDL designers in the implementation of applications, and allows also to make a fine low-level characterization of the off-chip memory sub-systems. On the Alveo U280, it measures a bandwidth of 425 GB/s and 36 GB/s, respectively for the 32-channels HBM and the 2-channels DDR4 memories.

In [30] is presented an OpenCL implementation of the HPCChallenge Benchmark Suite, reporting the results for different FPGAs. In particular, they show that on the Alveo U280 the GEMM benchmark achieves a performance of 202.6 GFLOP/s in SP, using ≈28% of the available DSPs, whereas the STREAM benchmark achieves a bandwidth of ≈34.6 GB/s using the 2-channels DDR4 memory banks.

In [31] is reported a C/HLS implementation of the HPCG Benchmark targeting FPGAs. Interestingly, in this case the Roofline Model has been used, but only to assess the optimization level of the specific application, with respect to theoretical estimations.

In fact the Roofline Model has already been used in the past to evaluate the performance of specific applications [32], being ported to FPGAs. But few works provide a generic application-independent extension of this model for these architectures, mainly due to the difficulty in defining the maximum compute performance for a reconfigurable device.

A first comprehensive work extending the Roofline Model to FPGAs has been presented in [33], but the authors focus mainly on aiding developers to explore the design space options, moreover, not taking into account FP operations. They estimate the expected performance of algorithms run on FPGAs by evaluating the hardware resources required to implement a single PE (Processing Element) of the specific algorithm, and deriving the maximum number of PEs to fill the device.

Building on the same principle, more recently, in [34] and in its extended version [35], a semi-automated performance optimization methodology based on the Roofline model for FPGAs has been proposed. In this case the authors, aim for a tool to explore the design space, whereas in our case we aim to provide a benchmarking tool.

The first work proposing a methodology for the performance analysis of FPGAs allowing to make Roofline plots and cross-architectural comparisons, has been reported in [36]. In this case, the authors use OpenCL as programming language to provide mini-apps, such as SHOCL0, LINPACK and STREAM, to measure the computing performance and the memory bandwidth of the off-chip memory.

In [37] we have reported the first C/HLS benchmark tool able to provide empirical Roofline plots for FPGAs. It is a C kernel annotated with OmpSs@FPGA [18] pragmas directives, and it has been used to assess the FP performance of the Xilinx ZU9 FPGA in the context of the EuroEXA Project. Later, we have extended this initial tool porting it also to the Xilinx Vitis workflow to allow for a wider adoption [23].

Also an OpenCL version of the ERT benchmark has been reported to run on FPGAs in [38] and in its extension [39]. In these works the authors focus on the performance and energy-efficiency evaluation of two FPGAs. Roofline plots are presented for an Intel Arria 10 FPGA and the Xilinx Alveo U280, and for the latter they report for the SGEMM and DGEMM kernels respectively ≈ 500 and 150 GFLOP/s. This research work is that with the most similar aims to ours. The main differences are that FER adopts a directive based C/HLS programming approach, focusing on a lower level characterization, allowing to strictly correlate its results with hardware resources by means of a theoretical model; and that it takes into account also on-chip memories bandwidth.

The work we present here extends the above works in several directions. We focus on application-agnostic performance assessment of FPGA based accelerators, aiming to a comprehensive machine characterization, allowing for cross-architectural comparisons and for performance estimations of generic HPC kernels on a given device. To this aim, FER is able to measure both the computing peak performance of FPGAs, and the bandwidths of on-chip and off-chip memories. It is based on the Roofline Model and it is implemented having at its core a directives annotated C/HLS kernel, with tunable operational intensity and hardware resources usage. Moreover, it relies on a theoretical model aiming to strictly link the performance results to the hardware resources. The choice of C/HLS allows at the same time to expose to the users low level fine tuning knobs, as well as to use a high-level programming paradigm that can easily be used by the HPC user community for development and porting. At the moment FER has been used both with the proprietary Xilinx Vitis workflow, as well as with OmpSs@FPGA [40].

III. THE FPGA EMPIRICAL ROOFLINE

The Roofline Model [25] is a visual performance model used to provide performance upper bounds for compute kernels, or more complex applications, when run on a target architecture. The most basic Roofline plot shows the maximum computational performance (e.g., FLOP/s) achievable by an application running on a specific processor as a function of the application *arithmetic intensity* $I = O/D$, where O is the number of operations performed (e.g., FLOPs), and D the amount of data exchanged with memory (e.g., Bytes).

In general, each computing architecture is characterized by a specific *machine balance* [41] $M_b = C/B$, where C is the peak compute performance (e.g. the maximum theoretical FLOP/s), and B is the peak memory bandwidth (e.g. the maximum theoretical Byte/s). The performance reached by an application with arithmetic intensity I, running on a processor with machine balance M_b, is then limited either by the peak compute performance or by the peak memory bandwidth, respectively if I is greater or lower than M_b. Kernels with an arithmetic intensity lower than the *machine balance* are said *memory-bound* for the specific processor architecture, otherwise they are said *compute-bound*.

The Roofline plot for a specific processor architecture can be drawn using its theoretical peak compute performance and memory bandwidth, but real applications are commonly able to exploit just a fraction of those values. For this reason,

to have more realistic Roofline *ceilings* in the plot, it is useful to exploit empirical values, measured running appropriate benchmarks. For FPGAs this is even more important, given that on these devices a theoretical estimation of the peak performance can lead to values which are quite higher than what could actually be achieved by real applications [42].

A first challenge in extending the Roofline Model to FPGAs, is given by the difficulty to define their compute performance before configuring them with a bitstream [33]. In fact, their peak performance strongly depends on the operations being implemented and on how these operations get mapped onto the hardware resources available in the FPGA [34]. This impacts on the number of implementable compute cores, as well as on the maximum clock frequency, clearly affecting the theoretical peak performance.

Actually, due to the growing complexity of CPUs and GPUs, also on these devices the theoretical compute performance is not anymore so trivial to estimate. In fact, according to the compute operations performed, and the kind of resources exploited (e.g., vector instructions, Fused-Multiply-Add instructions, etc.) different compute performance peaks can be theoretically estimated, and also empirically measured. Moreover, in recent processors also the maximum reachable clock frequency depends on the type of instructions being executed (e.g., if vector instructions are used, or not), and additionally, it could also be lowered due to thermal throttling. For these reasons, the Roofline Model for CPUs and GPUs has been extended to take into account different ceilings [43] for different sets of resources, and software tools have been developed to measure empirically these ceilings, such as the ERT benchmark [22].

For FPGAs, the higher degree of dimensions in the design space, increases the complexity in dealing with the different possible ceilings. Despite of this, focusing on the use of high level synthesis tools, and with the help of a simple theoretical model, we aim to show that the Roofline Model for FPGAs can still be useful to understand their compute performance and guide the optimization of HPC applications to exploit them. In the following we first describe the theoretical model on which the FER benchmark relies, and then we describe its implementation.

A. THEORETICAL MODEL FOR FPGAs

To produce the Roofline plot for a target FPGA, we need to estimate its *machine balance* $M_b = C/B$, where C is the peak computational performance, as operations per second, and B the memory bandwidth, as bytes per second.

We can start to define the relation between C and the hardware resources available in the FPGA. Assuming to implement a core performing the operation *op* (e.g., a FP addition), which requires R_{op} hardware resources, and assuming that R_{av} of these resources are available on the target FPGA: the ratio R_{av}/R_{op} gives the number of multiple *op*-cores that could be implemented on the device, potentially executing the *op* operation in parallel. If each core can be operated at a maximum clock frequency f_{op}, the corresponding peak theoretical compute performance C in op/s for the FPGA is:

$$C = f_{op} \times \left(\frac{R_{av}}{R_{op}} \right). \tag{1}$$

This equation is valid under the assumption of using all FPGA hardware resources to implement multiple instances of the same computing core, each able to start to perform one new *op* operation per clock cycle.

Since on FPGAs are commonly available different types of resources, such as LUTs (Look Up Tables), FFs (Flip-Flops), DSPs, etc., Eq. 1 can be extended to take into account all of them as following:

$$C = f_{op} \times \min_k \left(\frac{R^k_{av}}{R^k_{op}} \right), \tag{2}$$

where R^k_{av} is the amount of resources of type k, available on the target FPGA, whereas R^k_{op} is the amount of resources of type k used to implement the selected compute core.

In practice, it is slightly more complicated, since on reconfigurable devices the same operation can be implemented in several different ways, trading-off one resource type for another. However, focusing only on using high level synthesis tools, we may assume that common operations are implemented using pre-defined RTL (Register Transfer Level) cores picked from an IP (Intellectual Property) library, which are placed and wired for the target FPGA by the synthesis tool. As a consequence of this, the set of available cores is limited, and the resource number R^k_{op} used for each of them is commonly available in the library documentation. In practice, we rely on the same idea presented in [33], but we adopt as a PE an elementary RTL core from the IP library, performing an elementary operation (i.e., a FP addition, or a FP multiplication), not related to any specific algorithm, but commonly used by generic HPC kernels.

Theoretical models similar to that reported in Eq. 2, have already been used by FPGA manufacturers to publicize their devices peak performance, but some limitations should be taken into account to get realistic results [42], especially when large fractions of FPGA resources are used. In particular, the assumption to be able to exploit all the available resources of one specific type is rather optimistic, as well as the assumption to reach, for an actual application, the maximum theoretical clock frequency claimed by manufacturers for one single RTL core.

A more accurate performance model should take into account the implemented design clock frequency f_{imp} achieved when many resources are in use, and a factor $u < 1$ for each resource corresponding to the fraction that can be successfully placed and routed in real designs. This leads to a slightly more complex model equation:

$$C = f_{imp} \times \min_k \left(\frac{R^k_{av}}{R^k_{op}} \times u_{R^k} \right). \tag{3}$$

Implementation frequency and resource utilization factors, introduced in Eq. 3, are dependent on FPGA architectures,

synthesis tools, and the kind of operations to implement. Consequently to obtain those values we can consider two approaches. The first is to make empirical measurements with specific benchmarks, such as FER. While the second, is to use values provided by the hardware manufacturers, which could be available in the documentation as recommendations to avoid timing issues. Anyhow, the latter approach can be applied only if such recommendations are available, and anyway the former approach is the only viable one to validate the latter. For example, in the case of Xilinx Alveo cards, as will be discussed in Sec. IV, f_{imp} could be set to the *nominal frequency* or *default frequency* [44], cited in the devices datasheet. While the different u_R could be set to the recommended maximum resource utilization to avoid timing closure issues, mentioned in [45].

The other quantity needed to compute the *machine balance* is the peak memory bandwidth B. FPGAs have on-chip memories, such as Block-RAM (BRAM) and Ultra-RAM (URAM), but they may also be connected to off-chip memories, such as DDR and/or HBM (High Bandwidth Memory) banks. The Roofline plot can show different ceilings according to the different memory levels usable by an application, as commonly done for CPUs cache levels.

For on-chip memories the theoretical maximum bandwidth can be estimated with a similar approach as for the peak performance. Assuming that the clock frequency of the on-chip memory is the same as that of the kernel, the bandwidth can be computed as:

$$B_{ram} = f_{imp} \times W_{ram} \times R_{av}^{RAM} \times u_{R^{RAM}}, \tag{4}$$

where W_{ram} is the bit-width of the on-chip memory banks (e.g., BRAMs or URAMs), R_{av}^{RAM} is the amount of available on-chip memory banks and $u_{R^{RAM}}$ is the maximum utilization factor (i.e., $u_{R^{RAM}} < 1$). If the on-chip memory works at a different clock frequency (e.g., Intel BRAMs can work at twice the kernel frequency), f_{imp} can be multiplied by a constant and the equation is still valid. Interestingly enough, keeping data in on-chip memories, M_b becomes frequency independent, corresponding to the ratio:

$$M_b = \frac{\min_k \left(\frac{R_{av}^k}{R_{op}^k} \times u_{R^k} \right)}{W_{ram} \times R_{av}^{RAM} \times u_{R^{RAM}}}. \tag{5}$$

On the other side, when using off-chip memories (such as DDR or HBM), two different clock frequencies are actually involved: f_{imp} corresponding to that of the user kernel; and f_{ext} corresponding to the clock frequency of the off-chip memory. The maximum bandwidth B_{ext} attained is then given by the minimum between the on-chip and off-chip interfaces bandwidths:

$$B_{ext} = \min(f_{imp} \times W \times Ch,\ f_{ext} \times W_{ext} \times Ch_{ext}), \tag{6}$$

where W is the bit-width of the on-chip interface (e.g., an AXI interface), and W_{ext} for the off-chip memory (e.g., DDR or HBM), whereas Ch is the number of available on-chip channels, and Ch_{ext} the off-chip ones. Different channels (e.g., multiple AXI interfaces) are commonly used to connect to different memory banks.

B. FER IMPLEMENTATION

The first aim of the FER benchmark is to be able to measure the peak compute performance, and max bandwidths of the different memory levels, of a target FPGA-based architecture, allowing to produce a Roofline plot. In addition, it also aims to empirically establish realistic values for the maximum implementation frequency f_{imp} and utilization factors u_R, to be used in Eq. 3.

The first version of FER has been implemented as a plain C kernel decorated with Vivado HLS directives [46], and managed using the OmpSs@FPGA [18], [40] programming model, required to handle kernel launching and data movements between host and device memories. Preliminary results of this version have been reported in [37], running a single instance of the FER kernel on the FPGA embedded in a Xilinx Zynq ZU9 MPSoC. Later, the host-side of the benchmark has been implemented also using the Xilinx Vitis workflow [23] to manage kernel launches and data transfers, allowing for a wider adoption and to target more FPGA devices.

The latest version of FER, with a significant set of new features, is meant to have an user configurable computational intensity I, applying one or more times a given *op* operation to each of the elements of an input array, and writing for each of them a corresponding output one in an output array. FER can be configured to allocate a single or multiple CU (Compute Unit) instances, each instance operating on a chunk of the input buffer. This allows to easily map CUs on different SLRs (Super Logic Regions) avoiding costly inter-region long paths, and also to exploit different memory channels, if available on the target FPGA.

The FER top-level kernel include three different functions, two I/O functions to load and store data, and one compute function to process it. The kernel can be configured to run in two different ways, namely the *dataflow* and *datalocal* modes. These are actually two different kernels that can be used alternatively, providing the same interface. The former is suitable to measure peak values reachable keeping data in off-chip memories (e.g., on-board DDR or HBM memory banks), whereas the latter keeping it in on-chip memories (e.g., BRAM or URAM). In fact, to obtain the Roofline ceilings corresponding to on-chip memory levels, as commonly done for CPU cache levels, in the case of FPGAs it is required to explicitly program data movements to/from on-chip memories, and thus different kernels are needed.

When FER runs in *dataflow* mode, see Fig. 1, the computation is arranged as a task level pipeline defined by the HLS *dataflow* pragma directive [17], as shown in Listing 1. Each stage of this coarse pipeline is one of the three functions, namely `read()`, `computeDflow()` and `write()`; and data between stages is passed through FIFO queues `inFifo` and `outFifo`, getting implemented on on-chip memories. This allows to decouple I/O and compute functions, enabling to execute each stage concurrently, overlapping computations

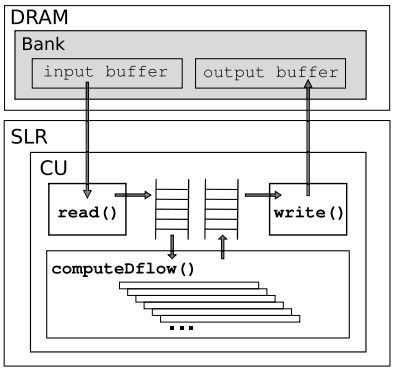

FIGURE 1. Schematic architecture of the FER Kernel for *dataflow* mode. The kernel is divided in three task level pipeline stages, reading and writing elements from the buffers stored onto the FPGA off-chip DRAM. Data movements and computations are overlapped and FIFO buffers are used to move data between *input*, *compute* and *output* stages.

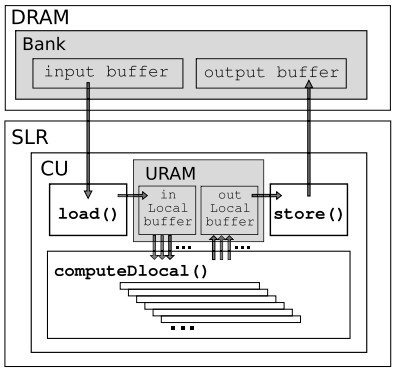

FIGURE 2. Schematic architecture of the FER Kernel for *datalocal* mode. In this case, *load* and *store* functions are run just once. The *load* function load the input buffer onto on-chip memories (such as URAM); the *compute* function reads and writes from/to on-chip memories for multiple iterations, and eventually the *store* function save results on external DRAM.

```
1
2  void fer( const data_v *input, data_v *output ) {
3    #pragma HLS interface ...
4    #pragma HLS dataflow
5    hls::stream<data_v> inFifo, outFifo;
6    #pragma HLS stream variable=inFifo
7    #pragma HLS stream variable=outFifo
8    read(input, inFifo);
9    computeDflow(inFifo, outFifo);
10   write(output, outFifo);
11 }
```

Listing 1. Organization of the main kernel function for *dataflow* mode.

```
1
2  void fer( const data_v *input, data_v *output ) {
3    data_v inLocal[DIM], outLocal[DIM];
4    #pragma HLS bind_storage variable=inLoc
5                               type=RAM_2P impl=URAM
6    #pragma HLS bind_storage variable=outLoc
7                               type=RAM_T2P impl=URAM
8    #pragma HLS array_partition variable=inLoc cyclic ...
9    #pragma HLS array_partition variable=outLoc cyclic ..
10   loadData(inLoc, input);
11   for ( int iter=0; iter < NITER; iter++ ) {
12     computeDlocal(inLoc, outLoc);
13   }
14   dumpData(outLoc, output);
15 }
```

Listing 2. Organization of the main kernel function for *datalocal* mode.

and I/O operations, hiding the latency to access off-chip memories.

Conversely, when FER runs in *datalocal* mode, see Fig. 2 and Listing 2, the three functions are executed in a serial fashion. In this case, data are moved by I/O functions between off-chip and on-chip memories and vice-versa, just once, at the beginning and at the end of the kernel execution, whereas the compute function is executed for several iterations. This allows the compute function to operate only on on-chip data, measuring the corresponding on-chip memories bandwidth. Multiple iterations are performed to increase the benchmark execution time and reliably measure it.

In both modes, I/O functions read and write data elements as SIMD (Single Instruction Multiple Data) vectors of length V from/to off-chip memories. Consequently, V can be set to select the optimal bit-width to fully utilize memory buses. Burst operations are then used to move data between off-chip and on-chip memories.

As shown in Listing 1, for the *dataflow* mode, SIMD vectors are passed to the compute function through FIFOs, whereas in Listing 2, for the *datalocal* mode, they are passed trough local buffers. Such local buffers (see Listing 2, Line 3-7), are explicitly requested to be implemented using on-chip dual port memory blocks (in this case URAMs), to allow for two reads/writes per clock cycle per memory block. Actually, for the output local buffer a True Dual Port (T2P) has to be requested (see Listing 2, Line 7), since the kernel will need to write in parallel, during the same clock cycle, on the the two ports of each memory block. On the other hand, for the input local buffer a Simple Dual Port (2P) is enough (see Listing 2, Line 5) to read in parallel on the two ports [47]. Then, both buffers need to be partitioned [17] across multiple memory blocks (see Listing 2, Line 8-9), to allow to read/write from/to the widest possible number of memory blocks, during the same clock cycle, to reach the maximum bandwidth.

The two compute functions, namely `computeDflow` and `computeDlocal` are outlined respectively in Listing 3, and in Listing 4.

Concerning the `computeDflow` function, for the *dataflow* mode, outlined in Listing 3, the outer loop at line 3 is pipelined with an *Initiation Interval* of II_c, and this requires the synthesis tool to automatically fully unroll the inner loops (in fact unroll directives for inner loops could be omitted). The loop at line 6 is horizontally unrolled [17] and thus vectorized into SIMD operations reading vectors of V data elements and applying the operation `op()` for O_e times, to each element of the SIMD vector. In fact, the loop at line 9 gets unrolled as well, translating into a chain of sequential operations to be applied to each vector element.

```
1  void computeDflow( hls::stream<data_v> &inFifo,
2                     hls::stream<data_v> &outFifo ) {
3    for (i = 0; i < DIM; i++) {
4      #pragma HLS pipeline II=IIc
5      data_v in = inFifo.read();
6      for (e = 0; e < V; e++) {
7        #pragma HLS unroll
8        data_t elem = in.elem[e];
9        for (o = 0; o < Oe; o++) {
10         #pragma HLS unroll
11         elem = op(elem);
12       }
13       out.elem[v] = elem;
14     }
15     outFifo.write(out);
16   }
```

Listing 3. Organization of the main compute function in *dataflow* mode.

```
1  void computeDlocal( data_v* inLoc, data_v* outLoc ) {
2    for (i = 0; i < (DIM/MEM/V); i++) {
3      #pragma HLS pipeline II=IIc
4      #pragma HLS dependence variable=outLoc
5                                inter WAW false
6      for (k = 0; k < MEM/V; k++) {
7        #pragma HLS unroll
8        j = i*(MEM/V) + k;
9        data_v in = input[i];
10       for (e = 0; e < V; e++) {
11         // Same as Listing 3, line 7-13
12       }
13       output[j] = out;
14     }
15   }
16 }
```

Listing 4. Organization of the main compute function in *datalocal* mode.

The loop at line 3 is actually translated to a hardware SIMD vector unit, able to start to process a new vector of V elements every II_c clock cycles. The amount H_c of cores implemented in hardware is then given by $(V \times O_e)/II_c$. In particular, with $II_c = 1$ the vector unit starts processing a new SIMD vector of width V at each clock cycle, whereas for $II_c > 1$ a fraction equal to $1/II_c$ of the SIMD vector length is processed per cycle. In fact, for example, setting $II_c = 2$ allows the synthesis tool to implement the hardware cores needed to start processing just half of a SIMD vector, at each clock cycle. As HLS tools try to optimize also the resources utilization, this translate to halving the hardware resources required.

The possibility to also set O_e, and II_c allows to configure FER with any number of operations per element O_e (i.e., the Arithmetic Intensity), independently from the hardware resources available on a target device. Clearly, configuring FER with parameters leading to an H_c greater than what allowed by the available hardware resources will lead to a not synthesizable configuration, but the user can increase II_c, to lower H_c, keeping O_e high enough to match the desired arithmetic intensity.

Once the three parameters are set, since C is limited by the number of cores implementable in hardware, as shown in Sec. III-A, the expected performance of each CU of the FER benchmark is given by equation:

$$C = f \times H_c = f \times \frac{V \times O_e}{II_c}. \tag{7}$$

Concerning the `computeDlocal` function, for the *datalocal* mode, outlined in Listing 4, the rationale is similar, apart from the facts that: `in` and `out` vectors are read/written from/to local arrays allocated on on-chip memories; and the width of the SIMD hardware unit which get implemented is a multiple of V, as wide as the number of used memory blocks allows. This is achieved by unrolling the loop at Line 6 in Listing 4, which automatically unroll all the inner loops, letting wider SIMD vectors to be read/written during the same clock cycle, using multiple on-chip memory blocks in parallel. In particular to write concurrently on the two ports, of the same memory block, it is needed to relieve the synthesis tool to enforce the WAW (Write-After-Write) output dependence, with the pragma at Line 4, taking the responsibility for writing at different addresses.

If R^{RAM} is the maximum number of usable on-chip memory blocks, half would be used for the input and half for the output buffer, but being dual port blocks, 2 elements can be read/written to/from each block per clock cycle, thus a R^{RAM} elements wide SIMD hardware unit could be implemented, operating on a new wide SIMD vector per clock cycle; assuming the single vector element does not exceed the memory block port width (e.g., 64-bit for Xilinx UltraScale+ URAM, when ECC is enabled).

Consequently, also for the `computeDlocal` function, for the *datalocal* mode, the same observations leading to Eq. 7 hold true, apart from the fact that V should be replaced by R^{RAM}. Anyhow, in this case the interest is not in C, but towards the reachable on-chip memories bandwidth, which can be directly computed using Eq. 4, when $II_c = 1$ in order to process one SIMD vector per clock cycle. Given that commonly $R^{RAM} \gg V$, the maximum number of implementable H_c could easily be reached, thus in this case it is also desirable to lower the operations per element (e.g., $O_e = 1$) to contain the final H_c.

IV. EXPERIMENTAL SETUP

In the context of HPC, the peak performance of CPUs and GPUs is commonly publicized using the DP-FP throughput, usually derived from the number of FMA (Fused Multiply Accumulate) instructions processed per second. For this reason, we initially configured FER to use a DP FMA as its core *op* operation, allowing for a direct comparison of its results against the ones obtained with similar benchmarks [22], [48] on commodity processors.

We remark that this is not the best option to show the highest FP performance achievable by FPGAs, but it is indeed for CPUs and GPUs, which are specifically designed and optimized to excel in FP FMAs throughput. In fact, this is commonly the only instruction accounting for two FLOPs per clock cycle when executed on CPUs FP units, whereas on most FPGAs its cost is approximately the same as an addition plus a multiplication. On the other side, this also translate to the fact that an actual FP intense application, when

not able to exploit FMAs, could reach at most half of the theoretical performance of an ordinary CPU, whereas it could theoretically reach the maximum performance measured by FER on a FPGA. In any case, FER can be easily configured to perform different *op* operations, also with a different numerical precision, as shown later.

In this work we have benchmarked different Xilinx Alveo Accelerator Cards: the U250 offering the highest number of hardware resources among the Alveos; the U50, a more recent board, with less resources compared to the U250, but embedding HBM2 high speed memory banks; and the U280, offering more resources than U50, together with both HBM2 and DRAM memory banks. The syntheses have been performed using the Xilinx Vitis version 2020.1, running on GNU/Linux CentOS 7 nodes of the COKA cluster hosted at INFN and University of Ferrara. Since Vitis allows for different synthesis and implementation strategies, we have selected those aiming to maximize the performance, and run multiple syntheses of our code for each test we have performed. In particular, we have used alternatively the *Performance_EarlyBlockPlacement*, the *Performance_ExploreWithRemap* or the *Performance_WLBlockPlacementFanoutOpt* strategies, targeting a FPGA clock frequency from 300 MHz up to 600 MHz, in steps of 50 MHz. All the results reported in the following refer to those settings and clock frequencies giving the best performance.

On the selected FPGAs, the Vitis synthesis tool implements FP operations using pre-defined RTL cores from a software library [49], although just a subset of them can be used when adopting high level synthesis [46]. In particular, in our case, the FMA operation is implemented using separate addition and multiplication cores, and the hardware resources they mostly utilize on the tested devices are DSPs and LUTs.

In the following sections, we first estimate the computing performance and bandwidths for each card using the theoretical models we have developed in Sec. III-A, and then we present and discuss the empirical results reporting our measurements achieved running the FER benchmark on each FPGA.

A. THE ALVEO U250

The Alveo U250 is a PCIe card embedding 4 × 16 GB DDR4 memory banks, for a total of 64 GB of on-board memory, and a XCU250 FPGA using Xilinx Stacked Silicon Interconnect (SSI) technology. The SSI allows for increased density by combining multiple physically separated Super Logic Regions (SLRs), four in the case of the XCU250, interconnected through Super Long Line (SLL) routes. However, the use of these connections should be limited to avoid timing issues that may prevent to synthesize the design at a high clock frequency. Therefore, the best strategy to fit FER (as any other application if possible) onto this architecture is to place 4 instances of the kernel, one per SLR, each using one different memory bank.

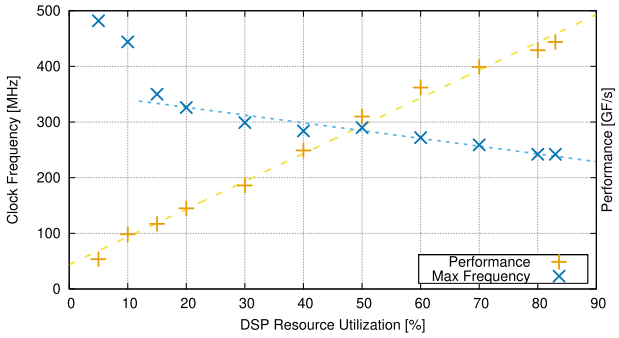

FIGURE 3. Maximum clock frequency (left axis) achieved synthesizing FER for the Alveo U250 and the corresponding DP-FP performance (right axis), as a function of the DSPs utilized by the implemented design. Linear fit lines are shown over experimental points.

The XCU250 accounts for $1.728 \cdot 10^6$ LUTs and 12288 DSPs, and is divided into two logical parts named *static region* and *dynamic region*. The first implements the *Shell* or *Platform* [44] provided by Xilinx, allowing for PCIe communications and bitstream loading into the FPGA, in the dynamic region. The latter is then used to implement the user designs. For the U250, we have used the *U250 XDMA 201830_2 Platform*, which leaves for the dynamic region $1.380 \cdot 10^6$ LUTs and 11508 DSPs.

In [49] is reported the number of resources used to implement DP addition and multiplication cores on this FPGA architecture, using this synthesis tool, and the corresponding maximum frequency at which they may operate. From the different available cores we selected the default ones used by the synthesis tool, which revealed to be also the best choice from a theoretical point of view, given the resources available on this device. For the addition operation are required: 616 LUTs and 3 DSPs, whereas for the multiplication: 172 LUTs and 8 DSPs. The maximum frequency at which both of the cores can operate is 694 MHz. Using Eq. 2 we can then derive the maximum number of FMAs we may fit into the FPGA and estimate a theoretical performance value:

$$\begin{aligned} C &= f \times \min\left(\frac{R_{av}^{\mathrm{LUT}}}{R_{add}^{\mathrm{LUT}} + R_{mul}^{\mathrm{LUT}}}, \frac{R_{av}^{\mathrm{DSP}}}{R_{add}^{\mathrm{DSP}} + R_{mul}^{\mathrm{DSP}}} \right) \\ &= 694\ \mathrm{MHz} \times \min\left(\frac{1.380 \cdot 10^6}{616 + 172}, \frac{11508}{3 + 8} \right) \\ &= 726 \cdot 10^9\ \mathrm{FMA/s} \\ &= 1.45\ \mathrm{TFLOP/s}. \end{aligned} \tag{8}$$

As claimed in [42], this approach may lead to not realistic values. In fact, synthesizing FER for the U250 using the estimated maximum number of mul- and add-cores leads to a not routable design. While, reducing the number of cores to allow for a successful routing, clearly shows that the frequency of 694 MHz is not reachable.

To make this evident, we have synthesized FER multiple times targeting the U250, for an increasing number of FMAs cores, keeping the benchmark in the compute-bound

region (i.e., the performance is never limited by the bandwidth), up to fit as much as possible the hardware resources with FMA cores. In Fig. 3 we report the clock frequency achieved by our syntheses and the corresponding performance as a function of the DSPs utilization, since this is the main resource limiting the number of FMA cores that can be implemented. As we see, to reach clock frequencies significantly higher than 300 MHz less than 20% of the available DSPs should be used, whereas using more than 80% of DSPs rapidly leads to a not routable design. An interesting result highlighted by Fig. 3 is that despite the lower frequency reached, exploiting as much resources as possible pays in terms of performance. Moreover, as shown by the fit, a linear relation links the amount of used DSPs with the performance: $C \approx 5 \times u_{R^{DSP}} + 44$. And when $u_{R^{DSP}} \geq 15\%$, another linear relation also links the amount of used DSPs with the maximum reached frequency: $f_{imp} \approx -1.4 \times u_{R^{DSP}} + 354$.

Interestingly, using the conservative values suggested by Xilinx best practices for designers, namely a clock frequency of 300 MHz and an utilization factor lower than 70% for LUTs and 80% for DSPs [44], [45] (with respect to the total amount of resources, not just the user available ones), Eq. 3 estimates a more realistic theoretical performance:

$$\begin{aligned} C &= 300\ \text{MHz} \times \min\left(\frac{1.728 \cdot 10^6}{616+172} \times 0.7,\ \frac{12288}{3+8} \times 0.8\right) \\ &= 536\ \text{GFLOP/s}, \end{aligned} \tag{9}$$

resulting approximately 20% higher with respect to the maximum performance we measured empirically with FER, and reported in Fig. 3.

Concerning the on-chip memories, such as URAMs, we can use a similar approach to estimate their maximum bandwidth. Using the conservative values suggested by Xilinx best practices, in this case 300 MHz of clock frequency and 80% as utilization factor, Eq. 4 gives:

$$\begin{aligned} B_{uram} &= 300\ \text{MHz} \times 64\ \text{bit} \times 2 \times 1280 \times 0.8 \\ &= 4.91\ \text{TB/s}, \end{aligned} \tag{10}$$

where 1280 is the amount of available dual-port (thus we multiply by 2 their number) URAMs. Each URAM block is 72 bits wide, but with ECC (Error Correction Code) enabled it offers 64 bits wide protected data words. In this paper we always consider ECC to be enabled. The maximum bandwidth would be 6.1TB/s with a 100% utilization.

Concerning the off-chip memory bandwidth, assuming that this is not limited by the user design, the maximum value estimated by Eq. 6 for the 4 DDR4 banks results in:

$$\begin{aligned} B_{ddr} &= 2 \times 1.2\ \text{GHz} \times 64\ \text{bit} \times 4 \\ &= 76.8\ \text{GB/s}. \end{aligned} \tag{11}$$

The 1.2 GHz clock frequency is multiplied by 2, since the memory banks are DDR (i.e., Double Data Rate).

Off-chip memory is accessed from the programmable logic through AXI interfaces that can be configured with a maximum bus width of 512 bits. Then, implementing a 512 bit interface for each of the 4 DDR4 banks and running at the nominal frequency of 300 MHz we achieve a bandwidth of $0.3\ \text{GHz} \times 512\ \text{bit} \times 4 = 76.8$ GB/s perfectly balancing that of DDRs.

Having estimated the computing performance and the memory bandwidths, it is possible to estimate the theoretical *machine balance* for the U250, for DP-FP operations, and in particular: $M_b = 536/4910 \approx 0.1$ when data are stored onto on-chip URAM, whereas $M_b = 536/76.8 \approx 7.0$ when data are stored onto off-chip DDR4 memory.

B. THE ALVEO U50

The Alveo U50 is a PCIe card embedding 8 GB of HBM2 (High Bandwidth Memory v2) accessible through 32 pseudo-channels and a XCU50 FPGA featuring SSI technology. With respect to the U250, the U50 provides less hardware resources (i.e., $872 \cdot 10^3$ LUTs and 5952 DSPs), divided in just two SLRs, but the HBM2 memory allows for a much higher bandwidth compared to DDR. As shown in Fig. 4b, only the SLR#0 is connected to the off-chip HBM2 memory banks, and then in this case SLL routes can not be avoided for accessing data from logic allocated in SLR#1. For the U50 all tests have been performed with the *U50 Gen3 × 16 XDMA 201920_3 Platform*, leaving to the user $731 \cdot 10^3$ LUTs and 5340 DSPs in the dynamic region. Using Eq. 3, with the default *Platform* frequency and recommended utilization values by Xilinx (with respect to the total amount of hardware resources), the expected performance is:

$$\begin{aligned} C &= 300 \times \min\left(\frac{872 \cdot 10^3}{616+172} \times 0.7,\ \frac{5952}{3+8} \times 0.8\right) \\ &= 260\ \text{GFLOP/s}. \end{aligned} \tag{12}$$

To measure the HBM2 bandwidth we have allocated multiple kernel instances (i.e., multiple CUs) in order to use the 28 (out of 32) memory pseudo-channels exploitable by the user design with the current platform [44]. In particular, we have used 14 CUs, each using two different pseudo-channels, one for the input, and one for the output buffer. The HBM2 peak bandwidth on this device is limited by the power provided by the PCIe rails (i.e., 10W for the memory), thus being limited to 316 GB/s [50] out of the theoretical 460 GB/s if all HBM2 channels could be used without power limits. The theoretical *machine balance* for DP-FP operations when reading data from the HBM2 is then $M_b = 260/316 \approx 0.82$, suitable for memory-bound applications with low arithmetic intensity.

To estimate the URAM on-chip memory bandwidth, using again the conservative values of 300 MHz for clock frequency and 80% as URAM utilization factor, Eq. 4 gives for this device:

$$\begin{aligned} B_{uram} &= 300\ \text{MHz} \times 64\ \text{bit} \times 2 \times 640 \times 0.8 \\ &= 2.46\ \text{TB/s}, \end{aligned} \tag{13}$$

where 640 is the number of available URAM units, resulting in a machine balance $M_b = 260/2460 \approx 0.1$. Theoretical bandwidth would be 2.8 TB/s with 100% URAM utilization.

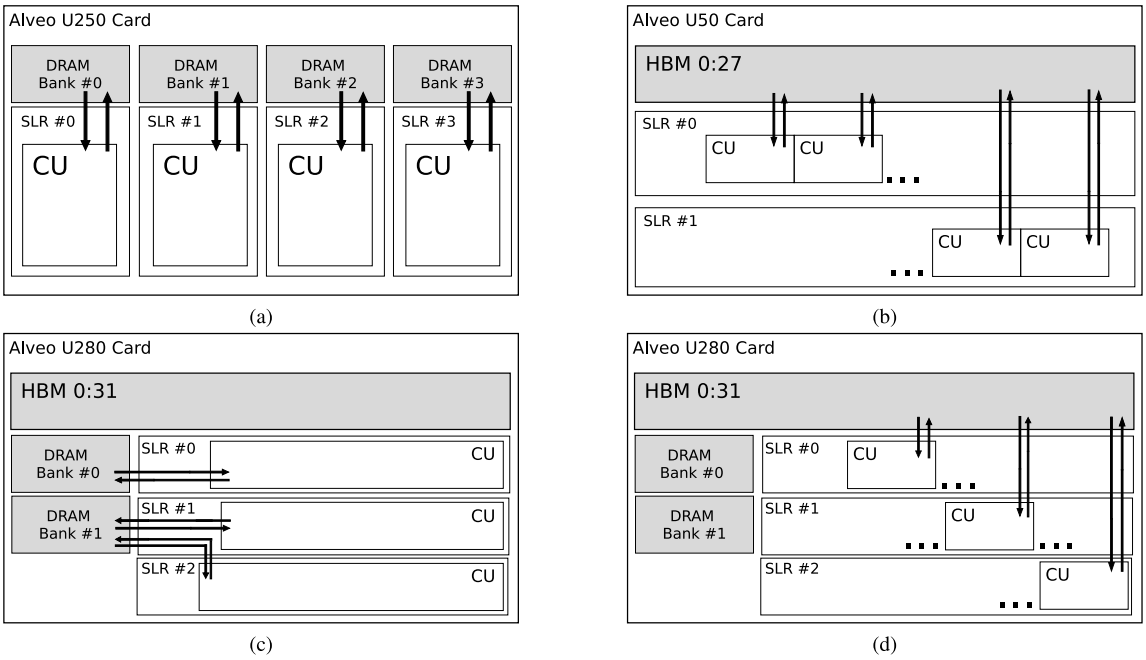

FIGURE 4. Schematic views of the FER CUs allocation on: Alveo U250 (Fig. 4a), one CU per SLR; Alveo U50 (Fig. 4b), multiple CU per SLR, to exploit all the available HBM memory pseudo-channels; Alveo U280 (Fig. 4c), one CU per SLR and Alveo U280 (Fig. 4d), multiple CU per SLR, to exploit all the available HBM memory pseudo-channels.

C. THE ALVEO U280

The Alveo U280 is a PCIe card embedding two HBM2 (High Bandwidth Memory v2) stacks, of a total size of 8 GB, accessible through 32 pseudo-channels; 32 GB of DDR memory; and a XCU280 FPGA featuring SSI technology. The U280 provides a set of features which is a mix of the ones that could be found in the U250 and the U50. It provides less hardware resources (i.e., $1304 \cdot 10^3$ LUTs and 9024 DSPs), with respect to the U250, but more than the U50. The programmable logic is divided in 3 SLRs, and can access both HBM2 memory banks, as well as ordinary DDR memory banks.

As shown in Fig. 4c, on the U280, only SLR#0 and SLR#1 are connected to the 2 off-chip DDR memory banks, and then in this case SLL routes can not be avoided for accessing DDRs from logic allocated in SLR#2. On the other hand, as shown in Fig. 4d, only the SLR#0 is connected to the off-chip HBM2 memory banks, and then in this case SLL routes can not be avoided for accessing data from logic allocated in SLR#1 and SLR#2.

For the U280, all tests have been performed with the *U280 XDMA 201920_3 Platform*, leaving to the user $1069 \cdot 10^3$ LUTs and 8490 DSPs in the dynamic region. Using Eq. 3, with the default *Platform* frequency and recommended utilization values by Xilinx (with respect to the total amount of hardware resources), the expected performance is:

$$\begin{aligned} C &= 300 \times \min\left(\frac{1304 \cdot 10^3}{616+172} \times 0.7,\ \frac{9024}{3+8} \times 0.8\right) \\ &= 394 \text{ GFLOP/s}. \end{aligned} \tag{14}$$

To measure the HBM2 bandwidth we have allocated multiple kernel instances (i.e., multiple CUs) in order to use all of the 32 memory pseudo-channels (16 per stack) exploitable by the user, as shown in Fig. 4d. In particular, we have used 16 CUs, each using two different pseudo-channels, one for the input, and one for the output buffer. The HBM2 peak bandwidth on this device is not limited by the power (as for the U50), thus its theoretical bandwidth, can be estimated by Eq. 6:

$$\begin{aligned} B_{hbm} &= 2 \times 0.9 \text{ GHz} \times 64 \text{ bit} \times 32 \\ &= 460 \text{ GB/s}. \end{aligned} \tag{15}$$

The theoretical *machine balance* for DP-FP operations when reading data from the HBM2 is then $M_b = 394/460 \approx 0.86$, suitable for memory-bound applications with low arithmetic intensity.

To estimate the URAM on-chip memory bandwidth, using again the conservative values of 300 MHz for clock frequency and 80% as URAM utilization factor, Eq. 4 gives for this device:

$$\begin{aligned} B_{uram} &= 300 \text{ MHz} \times 64 \text{ bit} \times 2 \times 960 \times 0.8 \\ &= 3.69 \text{ TB/s}, \end{aligned} \tag{16}$$

where 960 is the number of available URAM units (320 per SLR), resulting in a machine balance $M_b = 394/3690 \approx 0.1$. The theoretical bandwidth would be 4.6 TB/s under 100% URAM utilization.

V. RESULTS

In this section we report the results measured running the FER benchmark on the U250, U50 and U280 Alveo cards.

To measure the DP-FP peak computing performance we have run FER configured in *dataflow* mode (see Fig. 1, Listing 1 and Listing 3), using the FMA as main *op* operation, allowing for cross-architectural comparisons. For the off-chip memory bandwidths we have benchmarked both DDR4 and HBM2 where available, using again the *dataflow* mode, configuring the kernel for a low arithmetic intensity, to let it become memory-bound, while paying attention to exploit all the available memory channels on the different devices as shown in Fig. 4. To measure the on-chip URAM memory bandwidth, we have run FER configured in *datalocal* mode (see Fig. 2, Listing 2 and Listing 4), using as main *op* a single DP-FP addition, to reduce the arithmetic intensity to the lowest value of 1 FLOP per element, avoiding other resources than URAM to became limiting factors.

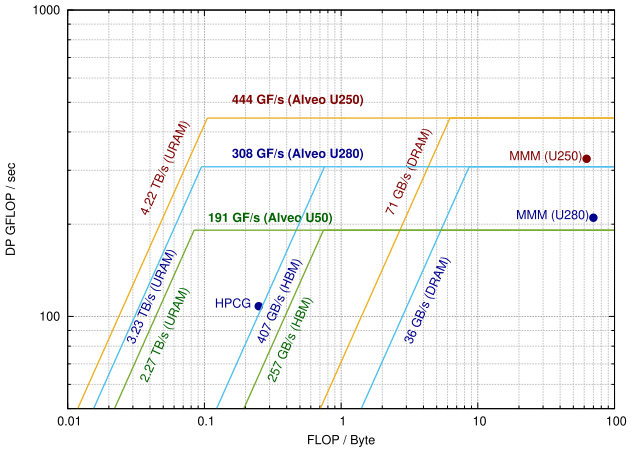

FIGURE 5. Empirical Roofline plot, showing DP-FP performance and both on-chip and off-chip memory bandwidths, obtained by FER in *dataflow* and *datalocal* modes, running on Xilinx Alveo cards. Points represents the performance of DP-FP third-party applications.

A. DATAFLOW MODE

On the Alveo U250, FER has measured a maximum compute performance of 444 GFLOP/s, using 83.18% of DSPs and 69.6% of LUTs. This is ≈20% less than the value estimated by Eq. 9. The maximum DDR4 bandwidth achieved by FER is 71 GB/s, i.e., 17.75 GB/s per channel, corresponding to ≈99% of the bandwidth per channel measured by the *Shuhai* Verilog benchmark [26] and ≈92% of the total 77 GB/s maximum theoretical bandwidth. As a reference, running on the U250 a highly optimized third-party HLS implementation of the compute-bound MMM (Matrix-Matrix-Multiplication) [51], computing 4 MMM in parallel, one kernel instance per SLR, we reached an aggregated performance of 327 GFLOP/s, as reported in Fig. 5.

On the U50, FER has measured a maximum compute performance of 191 GFLOP/s, corresponding to a ≈26% lower performance than estimated by Eq. 12. This is most likely due to the need for SLL paths crossing SLR boundaries, which are required to reach HBM memory banks from SLR#1. A slightly higher performance (≈10%) could be reached using just 2 CUs, one per SLR, but with such configuration FER could not exploit all of the HBM memory channels. For the HBM2 bandwidth, a peak of 257 GB/s (9.17 GB/s per pseudo-channel) has been reported, corresponding to ≈81% of the 316 GB/s maximum declared bandwidth. Interestingly, in a recent third party article [52], the FER benchmark (being already released as *Free Software*), has been used to evaluate the performance on the Alveo U50 of the Gridding Kernel, used for the acceleration of radio-astronomical imaging. This third-party application reaches ≈90% of the single-precision FP performance measured by FER on this device.

On the U280, FER has measured a maximum compute performance of 308 GFLOP/s. This is ≈78% of the value estimated by Eq. 14. The maximum DDR4 bandwidth achieved by FER is 35.6 GB/s, corresponding to ≈ 99% of the bandwidth measured by the *Shuhai* Verilog benchmark [26] and ≈94% of the total 38 GB/s maximum theoretical bandwidth. For the HBM2 bandwidth, a peak of 407 GB/s (12.7 GB/s per pseudo-channel) has been measured, corresponding to ≈96% of the bandwidth measured by the *Shuhai* [26] and ≈88% of the 460 GB/s maximum theoretical bandwidth. As a reference, running MMM [51] on the U280, we reached a performance of 210 GFLOP/s, as reported also in Fig. 5. Moreover, on this device has been recently ported a C/HLS version of HPCG [31], a memory-bound benchmark, able to exploit both the HBM memory channels and on-chip URAM blocks. We show in Fig. 5 the point corresponding to its computational intensity and performance, as reported in [31].

B. DATALOCAL MODE

Running FER also in *datalocal* mode, we have been able to add in the Roofline plots, reported in Fig. 5, the memory-bound ceilings corresponding to URAMs on-chip bandwidth.

In particular, for the U250, FER reports a bandwidth of 4.22 TB/s, which is ≈14% lower than the 4.91 TB/s estimated by Eq. 10. The synthesis achieved a maximum clock frequency of 245 MHz instead of 300 MHz, although a larger number (i.e., 90%) of the available URAM units have been used. Trying to use more URAMs in the attempt to further increase the bandwidth, leads to a not routable design. Higher URAM utilization may be achieved performing non FP operations, not involving the use of DSPs, and thus freeing additional routing resources.

For the U50 the bandwidth achieved is 2.27 TB/s using 80% of the URAMs corresponding almost exactly to what estimated by Eq. 13. In this case in fact, SSL path are used just to load the URAMs and store the results, at the beginning and at the end of the benchmark execution, thus their use does not impact on the on-chip memory performance.

For the U280 the bandwidth achieved is 3.23 TB/s using 90% of the URAMs corresponding almost exactly to what

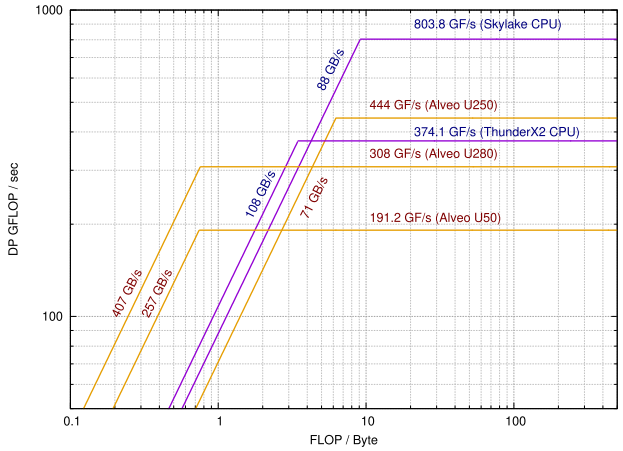

FIGURE 6. Empirical Roofline plot, for DP-FP, obtained by FER on Xilinx Alveo cards; for comparison, we also report the Roofline plots obtained by the ERT benchmark on an Intel Skylake CPU and by an Arm optimized version of ERT on a Marvell ThunderX2 CPU.

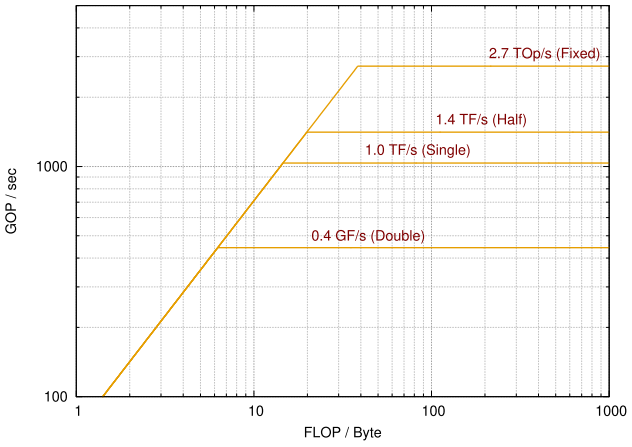

FIGURE 7. Empirical Roofline plot, for different numerical precisions (i.e., fixed-point; and floating-point in double- single- and half-precision), obtained by FER on a Xilinx Alveo U250.

estimated by Eq. 16. Also in this case the use of SSL paths does not impact on the local memory performance.

C. CROSS-ARCHITECTURAL COMPARISON

Using the DP-FP FMAs as main mathematical operation, for which the floating point accuracy is granted to be compliant with the IEEE-754 standard [53], we can also use FER results to compare FPGAs with commodity processors.

In Fig. 6 we compare the Roofline plots of U50, U250 and U280 FPGAs, with that of Intel Xeon Gold 6130 (based on Skylake micro-architecture) measured using the ERT benchmark [22], and that of Marvell ThunderX2 CPU (based on the Arm v8-A architecture) measured using an optimized ERT version we have developed [48]. Both processors appeared on the market in the same time frame as the Alveo architecture. As we see, the computing performance of the U250 is approximately 20% higher than the ThunderX2 CPU and a factor 2× lower than the Skylake CPU, whereas that of the U50 and the U280 are lower. Regarding the memory bandwidth, the U280 delivers approximately a factor 3.8× and 4.6× higher compared respectively to the ThunderX2 and Skylake.

As already mentioned, measuring the performance in terms of the FP-DP FMA throughput is extremely favorable to CPUs, but we believe it to be interesting since, in the context of HPC, architectures are commonly compared using this metric. We also remark that on FPGAs any mix of FP-DP additions and multiplications would lead approximately to the same DP-FP performance peak, whereas on CPUs this can be achieved just using only FMA instructions, otherwise the performance could easily be halved.

D. LOWER PRECISION DATA TYPES

Besides using DP for the FMA operation, we have also run FER on the U250 using SP, half-precision (HP) and fixed-point precision, to highlight one of the main features offered by reconfigurable accelerators.

For the fixed-point FMA, we have arbitrarily selected data types that maps efficiently onto the DSP48E2 units available on these devices. In particular, in this case, FMA operations are executed multiplying a 27-bit fixed-point operand by a 18-bit one, accumulating the result on another 27-bit one. For 27-bit operands we have used 15-bits for the integer part and 12-bits for the decimal one, whereas for 18-bit operands we have used respectively 9-bits and 12-bits; anyhow, this choice does not affect the computing performance achieved.

The results for the different data-types we have tested are shown in Fig. 7. Highlighting that using fixed-point operations leads to a much higher performance, achieving a peak value of approximately 2.7 TOP/s, almost doubling the HP performance, and improving respectively by a factor 6.7× and 2.7× the DP and SP ones.

E. DISCUSSION

In Tab. 1 we summarize the results measured by FER on the three Alveo cards. It can be noticed that FER is able to achieve almost the same off-chip bandwidths as reported by the *Shuhai* benchmark [26], despite HLS is a higher level programming language than Verilog. At the same time, FER is able to reach a higher compute performance than the OpenCL benchmarks cited in Sec. II, since HLS allows for lower level optimizations than OpenCL.

Tab. 1 compares also empirical results with theoretical estimations, showing that FER reaches ≈80% of the performance estimated by Eq. 3, and an even higher fraction of the on-chip and off-chip theoretical bandwidths. The lower performance with respect to the theoretical estimations of Eq. 3 is due to the use of f_{imp} and u_{R^k} values obtained from the Xilinx documentation of the devices, which result to be too optimistic compared to the real values measured empirically by the benchmark; in fact, as shown in Fig. 3 for the Alveo U250, the nominal frequency of 300 MHz can not be reached in practice, when a high fraction of resources is being used. As measured by FER, the clock frequencies reached are

TABLE 1. Comparison between theoretical estimations using Eq. 3, 4, 6 and empirical results measured by FER. Theoretical estimations assume the nominal clock frequency of 300 MHz and utilization factors ≤70% for LUTs and ≤80% for DSPs and URAMs.

	U250		U50		U280	
	Theor.	FER	Theor.	FER	Theor.	FER
DP Perf. [GF/s]	536	444	260	191	394	308
URAM [TB/s]	4.91	4.22	2.46	2.27	3.69	3.23
HBM [GB/s]	–	–	316[a]	257	460	407
DDR [GB/s]	77	71	–	–	38	36

[a] Alveo U50 HBM theoretical bandwidth can not be computed using Eq. 6, since it is limited by the memory power drain (i.e., 10 Watt) [50].

respectively 242 MHz for the U250, 290 MHz for the U50, and 273 MHz for the U280. Fig. 3 also highlights that using as much resources as possible pays in terms of performance, despite the lower frequency achieved. On the other hand, for devices having an uneven distribution of resources between SLRs and designs requiring the use of SLL paths, it may also be difficult to reach the utilization factors mentioned in the Xilinx documentation. In fact the maximum $u_{R^{DSP}}$ reached are respectively 83% for the U250, but 62% for the U50 and 70% for the U280. This explains the discrepancies with the theoretical values, in fact using the values of f_{imp} and u_{R^k} measured by FER into Eq. 3 leads to the same performance measured empirically.

Within the HPC community, there are currently few open source C/HLS applications available, but some of them have shown to nicely fit under the ceilings of the Roofline measured by FER. In particular, the compute-bound MMM [51] reaches ≈80% of the peak performance measured by FER on different cards; the Gridding Kernel, used for a radio-astronomical imaging application [52], reaches ≈90% of the SP peak performance measured by FER on the U50; and also the reported performance of the memory-bound HPCG benchmark [31] on the U280 lies between the HBM and the URAM ceilings reported by FER on this device.

These applications have been reported in research papers as success stories. However, we do not expect every application to be able to get so close to the Roofline ceilings, since FER provides only the performance upper-bound, highlighting some of the limits that applications can face, but not all of them. As an example, complex compute kernels can easily be limited by routing congestion, strongly reducing the amount of implementable hardware compute cores and consequently the corresponding performance. In our experience, compute-bound kernels can get quite close to the Roofline ceiling, if their main compute kernel is made of simple loops, possibly nested, that could be easily pipelined and unrolled [17]. On the other hand, complex codes (e.g., involving lot of conditional statements, complex dependencies, reduction operations, lot of partial results to be kept in memory, etc.), may hit other performance limits.

Off-chip bandwidth peaks can be reached taking care of using all of the available memory channels and of exploiting their full bit-width. The same holds true for on-chip memories, but in this case it could be more difficult for an actual application to reach the maximum bandwidth, in particular if several operations per element need to be applied. As an example, on the U250, FER in *datalocal* mode, using 90% of the URAMs and performing just 1 FLOP per element, required to use 29% of the available DSPs. For this reason we evaluated just the URAMs bandwidth, but not the BRAMs one. This could be done in principle, but the operation to be applied on each element should be elementary to avoid to hit other limits due to the lack of other resources.

VI. CONCLUSION

In this paper we have presented FER, a synthetic benchmark we developed using C/HLS, allowing to produce Roofline plots for FPGA-based accelerators, measuring their computing performance, as well as their on-chip and off-chip memory bandwidths.

As described in Sec. III-A, a Roofline plot could be estimated by theoretical models, but to obtain realistic values, empirical parameters are needed, such as the maximum clock frequency and utilization factors for the different resources used in the design. The FER benchmark can be used to measure these parameters for a given architecture; to validate them if already available in the datasheets; or to directly measure the peak performance, and peak memory bandwidths for off-chip and on-chip memories.

In this paper we have described the theoretical model on which FER relies, we have presented its implementation details and we have highlighted the fact that third party applications can reach a performance in line with FER predictions. Using FER we have also assessed the performance of three different Xilinx Alveo cards and compared it with the one measured by the ERT benchmark on two commodity CPU processors, based on different architectures.

The collected results show that FP workloads with a high arithmetic intensity, can reach on FPGAs a performance in the same order of magnitude as the one on CPU processors. Thus in this context FPGAs can not stand yet the competition with GPUs. On the other hand, for workloads with a lower arithmetic intensity, FPGAs equipped with HBM memories can outperform CPUs, since their off-chip memory bandwidth is comparable with the ones of GPUs. It is also worth to mention that FPGA on-chip memories have a bandwidth in the same order as cache memories, but the former may allow for parallel accesses in any order, whereas the latter are organized in cache lines. This makes FPGAs particularly well suited for memory-bound kernels requiring complex memory accesses, being difficult to optimize on cache based architectures (e.g., performing complex stencil operations [54]).

Moreover, using half-precision and fixed-point operations, FPGAs could outperform CPU processors also from the compute performance point of view. In fact, in this case the U250 is able to deliver respectively up to 1.4 TF/s and 2.7 TOp/s, whereas the tested CPU architectures would not show any

performance benefit in using such low-precision math, over their SP performance.

We expect that in the next future FPGAs will embed more hardened FP cores, but we doubt that they will be able to compete in this field with other accelerators, such as GPUs. At the same time, we believe that the interest towards reconfigurable hardware is increasing, favoring the use of FPGAs where precision can be traded for performance [55]. This requires to exploit new performance optimization strategies for HPC applications adopting math with lower and custom precisions. As an example, in the context of machine learning, the weights associated to graph nodes can be quantized and reduced in precision without negative impacts on results accuracy [56]. And the same approach has started to be investigated also for more traditional HPC workloads [57], [58], initially motivated by the availability of half-precision hardware cores on GPUs [59], but possibly further motivated in the next future by a wider availability of FPGA-based accelerators.

As future works, we plan to port and run FER on other FPGA-based accelerators. In fact, thanks to the C/HLS kernel at its core, FER is easy to port across different frameworks sharing similar directives, allowing to target also other devices, such as Intel FPGAs [17]. Moreover, running FER with more complex *op* we also aim to study the impact on routing resources and investigate the conditions in which they became a performance limiting factor, aiming to refine the presented theoretical model accordingly.

ACKNOWLEDGMENT

The authors are extremely grateful to Prof. Raffaele Tripiccione or simply Lele, as he preferred to be called, for all the thoughts he shared with us over the past years. He has been all one could look for in a good mentor and friend. They thank Dr. Aggelos D. Ioannou for his help with the HPCG benchmark for FPGAs. They also acknowledge Xilinx and ETH Zurich for providing access to the Alveo U280 cards installed at the Xilinx Adaptive Computing Cluster (XACC).

REFERENCES

[1] *Top500 Ranking*. Accessed: Dec. 2021. [Online]. Available: https://www.top500.org/

[2] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, and M. Haselman, "A reconfigurable fabric for accelerating large-scale datacenter services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, May/Jun. 2015.

[3] G. Alonso and P. Bailis, "Research for practice: FPGAs in datacenters," *Commun. ACM*, vol. 61, no. 9, pp. 48–49, Aug. 2018.

[4] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[5] J. Fowers, K. Ovtcharov, M. K. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "Inside project Brainwave's cloud-scale, real-time AI processor," *IEEE Micro*, vol. 39, no. 3, pp. 20–28, May 2019.

[6] W. Vanderbauwhede and K. Benkrid, *High-Performance Computing Using FPGAs*, vol. 3. Berlin, Germany: Springer, 2013.

[7] M. Véstias and H. Neto, "Trends of CPU, GPU and FPGA for high-performance computing," in *Proc. 24th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2014, pp. 1–6.

[8] F. A. Escobar, X. Chang, and C. Valderrama, "Suitability analysis of FPGAs for heterogeneous platforms in HPC," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 600–612, Feb. 2016.

[9] D. F. Bacon, R. Rabbah, and S. Shukla, "FPGA programming for the masses," *Commun. ACM*, vol. 56, no. 4, pp. 56–63, Apr. 2013.

[10] B. van Werkhoven, W. J. Palenstijn, and A. Sclocco, "Lessons learned in a decade of research software engineering GPU applications," in *Proc. ICCS*. Cham: Springer, 2020, pp. 399–412.

[11] F. Brosser, H. Y. Cheah, and S. A. Fahmy, "Iterative floating point computation using FPGA DSP blocks," in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–6.

[12] B. Ronak and S. A. Fahmy, "Mapping for maximum performance on FPGA DSP blocks," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 4, pp. 573–585, Apr. 2016.

[13] T. Vanevenhoven, "High-level implementation of bit- and cycle-accurate floating-point DSP algorithms with Xilinx FPGAs," Xilinx, San Jose, CA, USA, White Paper 409, Oct. 2011.

[14] BDT. (Feb. 2013). *Floating-Point DSP Energy Efficiency on Altera 28 nm FPGAs*. Berkeley Design Technology. Independent Evaluation. [Online]. Available: http://www.altera.com/literature/wp/wp-01192-bdti-altera-fp-dsp-energy-efficiency.pdf

[15] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2016, pp. 409–420.

[16] R. Nane, V. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[17] J. D. F. Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1014–1029, May 2021.

[18] J. Bosch, X. Tan, A. Filgueras, M. Vidal, M. Mateu, D. Jimenez-Gonzalez, C. Alvarez, X. Martorell, E. Ayguade, and J. Labarta, "Application acceleration on FPGAs with OmpSsFPGA," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2018, pp. 70–77.

[19] R. Yasudo, J. Coutinho, A. Varbanescu, W. Luk, H. Amano, and T. Becker, "Performance estimation for exascale reconfigurable dataflow platforms," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2018, pp. 314–317.

[20] J. Lant, J. Navaridas, M. Lujan, and J. Goodacre, "Toward FPGA-based HPC: Advancing interconnect technologies," *IEEE Micro*, vol. 40, no. 1, pp. 25–34, Jan. 2020.

[21] E. Calore. (2020). *FPGA Empirical Roofline*. [Online]. Available: https://baltig.infn.it/EuroEXA/FER

[22] Y. J. Lo, S. Williams, B. Van Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright, M. W. Hall, and L. Oliker, "Roofline model toolkit: A practical tool for architectural and program analysis," in *High Performance Computing Systems, Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham, Switzerland: Springer, 2015, pp. 129–148.

[23] E. Calore and S. F. Schifano, "Performance assessment of FPGAs as HPC accelerators using the FPGA empirical roofline," in *Proc. 31st Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2021, pp. 83–90.

[24] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High bandwidth memory on FPGAs: A data analytics perspective," in *Proc. 30th Int. Conf. Field-Program. Log. Appl. (FPL)*, Aug. 2020, pp. 1–8.

[25] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[26] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on FPGAS," in *Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2020, pp. 111–119.

[27] S. W. Nabi and W. Vanderbauwhede, "MP-STREAM: A memory performance benchmark for design space exploration on heterogeneous HPC devices," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 194–197.

[28] H. R. Zohouri and S. Matsuoka, "The memory controller wall: Benchmarking the Intel FPGA SDK for OpenCL memory interface," in *Proc. IEEE/ACM Int. Workshop Heterogeneous High-Perform. Reconfigurable Comput.*, Nov. 2019, pp. 11–18.

[29] Z. Jin, H. Finkel, K. Yoshii, and F. Cappello, "Evaluation of a floating-point intensive kernel on FPGA," in *Euro-Par Parallel Processing Workshops*. Cham, Switzerland: Springer, 2018, pp. 664–675.

[30] M. Meyer, T. Kenter, and C. Plessl, "Evaluating FPGA accelerator performance with a parameterized OpenCL adaptation of selected benchmarks of the HPCChallenge benchmark suite," in *Proc. IEEE/ACM Int. Workshop Heterogeneous High-Perform. Reconfigurable Comput.*, Nov. 2020, pp. 10–18.
[31] A. Zeni, K. O'Brien, M. Blott, and M. D. Santambrogio, "Optimized implementation of the HPCG benchmark on reconfigurable hardware," in *Euro-Par Parallel Processing*, L. Sousa, N. Roma, and P. Tomás, Eds. Cham, Switzerland: Springer, 2021, pp. 616–630.
[32] K. Nagasu, K. Sano, F. Kono, and N. Nakasato, "FPGA-based tsunami simulation: Performance comparison with GPUs, and roofline model for scalability analysis," *J. Parallel Distrib. Comput.*, vol. 106, pp. 153–169, Aug. 2017.
[33] B. D. Silva, A. Braeken, E. H. D'Hollander, and A. Touhafi, "Performance modeling for FPGAs: Extending the roofline model with high-level synthesis tools," *Int. J. Reconfigurable Comput.*, vol. 2013, pp. 1–10, Jan. 2013.
[34] M. Siracusa, L. D. Tucci, M. Rabozzi, S. Williams, E. D. Sozzo, and M. D. Santambrogio, "A CAD-based methodology to optimize HLS code via the roofline model," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.
[35] M. Siracusa, E. Del Sozzo, M. Rabozzi, L. Di Tucci, S. Williams, D. Sciuto, and M. D. Santambrogio, "A comprehensive methodology to optimize FPGA designs via the roofline model," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1903–1915, Aug. 2022.
[36] S. Muralidharan, K. O'Brien, and C. Lalanne, "A semi-automated tool flow for roofline anaylsis of OpenCL kernels on accelerators," in *Proc. 1st Int. Workshop Heterogeneous High-Perform. Reconfigurable Comput.*, 2015, pp. 1–8.
[37] E. Calore and S. Schifano, "Energy-efficiency evaluation of FPGAs for floating-point intensive workloads," in *Parallel Computing is Everywhere* (Advances in Parallel Computing), vol. 36. Amsterdam, The Netherlands: IOS Press, 2020, pp. 555–564.
[38] T. Nguyen, S. Williams, M. Siracusa, C. MacLean, D. Doerfler, and N. J. Wright, "The performance and energy efficiency potential of FPGAs in scientific computing," in *Proc. IEEE/ACM Perform. Model., Benchmarking Simul. High Perform. Comput. Syst. (PMBS)*, Nov. 2020, pp. 8–19.
[39] T. Nguyen, C. MacLean, M. Siracusa, D. Doerfler, N. J. Wright, and S. Williams, "FPGA-based HPC accelerators: An evaluation on performance and energy efficiency," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 20, p. e6570, Sep. 2022.
[40] J. M. de Haro, J. Bosch, A. Filgueras, M. Vidal, D. Jiménez-González, C. Álvarez, X. Martorell, E. Ayguadé, and J. Labarta, "OmpSsFPGA framework for high performance FPGA computing," *IEEE Trans. Comput.*, vol. 70, no. 12, pp. 2029–2042, Dec. 2021.
[41] J. D. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Comput. Soc. Tech. Committee Comput. Archit. (TCCA) Newslett.*, vol. 2, nos. 19–25, pp. 1–8, Dec. 1995.
[42] M. Parker, "Understanding peak floating-point performance claims," Intel, Santa Clara, CA, USA, White Paper 01222, 2017.
[43] C. Yang, T. Kurth, and S. Williams, "Hierarchical roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 perlmutter system," *Concurrency Comput., Pract. Exper.*, vol. 32, no. 20, p. e5547, Oct. 2020.
[44] *Alveo Data Center Accelerator Card Platforms*, UG1120, V1.2, Xilinx, San Jose, CA, USA, Jun. 2020.
[45] *Vitis Unified Software Platform Documentation*, UG1393, V2020.1, Xilinx, San Jose, CA, USA, Aug. 2020.
[46] *VivadoHLS*. Accessed: Aug. 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html
[47] *UltraScale Architecture Memory Resources*, UG573, V1.11, Xilinx, San Jose, CA, USA, Aug. 2020.
[48] E. Calore, A. Gabbana, S. Schifano, and R. Tripiccione, "ThunderX2 performance and energy-efficiency for HPC workloads," *Computation*, vol. 8, no. 1, p. 20, 2020.
[49] *Performance and Resource Utilization for Floating-Point V7.1*, Vivado Design Suite Release 2020.1, Xilinx, San Jose, CA, USA, 2020.
[50] *Alveo U50 Data Center AcceleratorCard Data Sheet*, V1.7.1, Xilinx, San Jose, CA, USA, Aug. 2020.
[51] J. D. F. Licht, G. Kwasniewski, and T. Hoefler, "Flexible communication avoiding matrix multiplication on FPGA with high-level synthesis," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*. New York, NY, USA, Feb. 2020, pp. 244–254.
[52] S. Corda, B. Veenboer, A. J. Awan, J. W. Romein, R. Jordans, A. Kumar, A.-J. Boonstra, and H. Corporaal, "Reduced-precision acceleration of radio-astronomical imaging on reconfigurable hardware," *IEEE Access*, vol. 10, pp. 22819–22843, 2022.
[53] *Floating-Point Operator V7.1*, Vivado Design Suite, Xilinx, San Jose, CA, USA, 2020.
[54] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Optimization of lattice Boltzmann simulations on heterogeneous computers," *Int. J. High Perform. Comput. Appl.*, vol. 33, no. 1, pp. 124–139, 2019.
[55] U. I. Minhas, S. Bayliss, and G. A. Constantinides, "GPU vs FPGA: A comparative analysis for non-standard precision," in *Reconfigurable Computing: Architectures, Tools, and Applications* (Lecture Notes in Computer Science), vol. 8405. Cham, Switzerland: Springer, 2014, pp. 298–305.
[56] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of low numeric precision deep learning inference using Intel FPGAs," in *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 73–80.
[57] S. Cherubin, G. Agosta, I. Lasri, E. Rohou, and O. Sentieys, "Implications of reduced-precision computations in HPC: Performance, energy and error," in *Parallel Computing is Everywhere* (Advances in Parallel Computing), vol. 32. Amsterdam, Netherlands: IOS, 2018, pp. 297–306.
[58] S. Cherubin and G. Agosta, "Tools for reduced precision computation: A survey," *ACM Comput. Surveys*, vol. 53, no. 2, pp. 1–35, Apr. 2020.
[59] A. Haidar, P. Wu, S. Tomov, and J. Dongarra, "Investigating half precision arithmetic to accelerate dense linear system solvers," in *Proc. 8th Workshop Latest Adv. Scalable Algorithms Large-Scale Syst.*, 2017, pp. 1–8.

ENRICO CALORE received the graduate degree in computer engineering from the University of Padua, Italy, in 2010, and the Ph.D. degree in computer science from the University of Milan, Italy, in 2014. He has been a Postdoctoral Researcher with the Italian National Institute of Nuclear Physics (INFN), as well as with the University of Ferrara, where he has also been an Adjunct Professor with the Mathematics and Computer Science Department. He is currently a Research Fellow with the University of Ferrara and INFN Associate. He is the author of more than 50 articles and conference papers. His research interests include HPC, parallel, and distributed computing; focusing in particular on scientific computing, parallel programming, performance engineering, and energy-efficiency.

SEBASTIANO FABIO SCHIFANO received the degree in computer science from the University of Pisa, Italy. He spent the early scientific career at IEI-CNR and INFN. He is currently a Professor with the University of Ferrara, Italy. He had a major role in several projects for the development of parallel systems optimized for Lattice Gauge Theory, fluid dynamics, and spin-glasses. In 2007, he was a coauthor of the QPACE Project to develop a massively parallel system based on IBM Cell-BE Processors, interconnected by a custom 3D-mesh network. From 2005 to 2010, he contributed to the design and development of the Janus systems and an array of FPGAs for spin-glass simulations. He is the author of more than 100 articles and conference papers. More recently, his research interests include the design and optimization of applications for multi- and many-core processor architectures. He was awarded as the Best Green500 System, from November 2009 to June 2010.