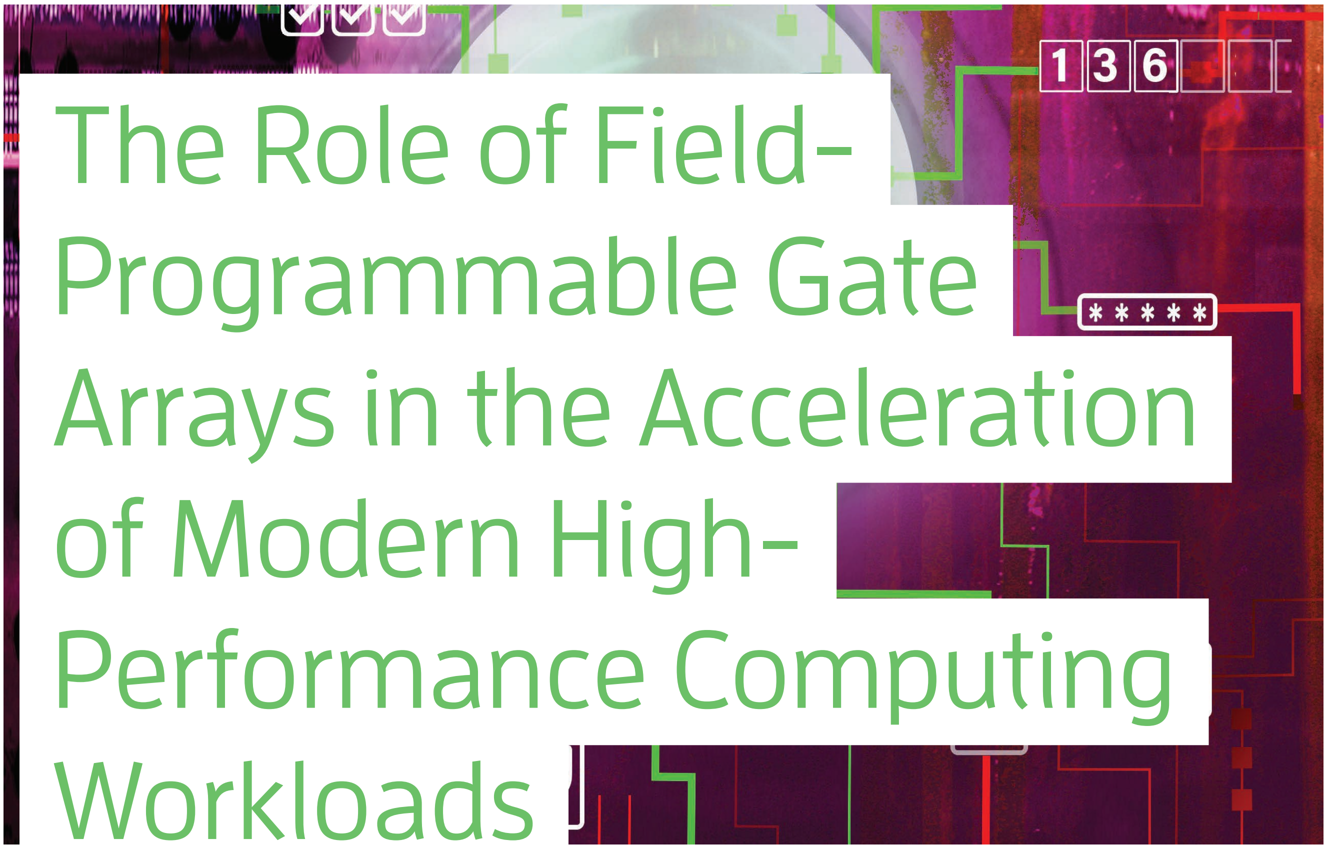

The Role of Field-Programmable Gate Arrays in the Acceleration of Modern High-Performance Computing Workloads

Manuel de Castro, University of Valladolid
David L. Vilariño, University of Santiago de Compostela
Yuri Torres and **Diego R. Llanos**, University of Valladolid

Reconfigurable hardware circuits, such as field-programmable gate arrays, have gained popularity in the high-performance computing (HPC) community in recent years. Nevertheless, their real contribution to accelerating HPC workloads is unclear in both potential and extent.

In the early 2000s, the increment in single-core CPU performance slowed down significantly with respect to previous decades. This caused new techniques and design paradigms, such as parallel (multicore) or vectorial processing, to emerge as alternatives to further increase CPU performance. Scientists also started investigating the potential use of GPUs as high-performance computational units for floating-point intensive computations. That encouraged the main GPU vendors to develop frameworks, languages, and runtime environments to ease the programming of GPUs for purposes beyond graphic processing. Consequently, general-purpose computing on GPUs was born. This entailed a paradigm shift for the high-performance computing (HPC) community, as heterogeneous systems including regular CPUs and specialized hardware accelerators became the standard for supercomputers, and data parallelism took the spotlight.

As a consequence of this shift toward heterogeneous systems, different kinds of hardware accelerators, from GPUs to field-programmable gate arrays (FPGAs) to application-specific integrated circuits (ASICs), have appeared

Digital Object Identifier 10.1109/MC.2024.3378380
Date of current version: 26 June 2024

during the last two decades. Among them, FPGAs have recently gained interest in the literature as a promising HPC platform. However, there exists a sharp contrast between this increasing research interest in FPGAs' theoretical capabilities and their low general adoption. This situation begs some questions: Are current data center FPGAs well suited for accelerating modern HPC workloads? When and how is it advisable to leverage FPGA devices to accelerate scientific computations? Let us discuss these topics in more detail by first putting heterogeneous accelerators in perspective and, later, analyzing the characteristics, advantages, and drawbacks of FPGAs, including their programmability and the portability of their code, to offer an answer to these questions.

GPUs: THE STANDARD HPC ACCELERATOR

As efforts to increase processing performance since the early 2000s have focused on parallel computing and its many forms, GPUs have revolutionized the field, due to their massively parallel architectures. GPUs include thousands of processing cores, simpler than the ones used for CPUs, which are designed so that all of them perform the same computations (that is, instructions) on different and independent datasets. Even though each individual GPU core is considerably less computationally powerful than a CPU core, the sheer number of them that a single device can contain makes GPUs superior to CPUs when it comes to data-parallel processing, both in raw performance and energy efficiency.

The high interest in GPUs manifested by the HPC community from the beginning has greatly influenced the industry. We highlight here two main consequences. First, GPU vendors started assembling what we may call "general-purpose versions" of their cards, adding error-correcting code memory and other features to better suit HPC needs. More recently, mainly due to the artificial intelligence (AI) market (and its convergence with HPC), GPU vendors also started to develop GPUs with scientific/AI computations in mind. Second, programming languages, frameworks, and models for heterogeneous computing mainly targeting GPUs have been created. Thus, their design philosophy has been GPU centric, or at least data parallelism centric. For example, OpenCL, SYCL, and Data Parallel C++ include programming constructs that map particularly well to GPU architectures, even though all of them are designed to work with a wide range of computing devices, not only GPUs. CPUs can easily translate these constructs to their own architectural resources and efficiently work with them, but this is not the case for all computing devices supported by these models (for example, FPGAs).

To maintain GPU dominance, vendors have recently started including more specific hardware in their devices, which further accelerate tasks of high current interest. For example, since 2018, Nvidia GPUs include dedicated tensor cores for the acceleration of deep learning workloads.

THE DECLINE OF FPGAs WAS NOT JUST AN ISSUE OF COMPUTING PERFORMANCE OR EFFICIENCY BUT ALSO A PROBLEM OF PRODUCTIVITY.

ASICs: SPECIFIC-PURPOSE ACCELERATORS

Some computational algorithms only moderately benefited from conventional GPU architectures, while others needed to be accelerated even further. In these cases, ASICs came to the rescue. ASICs are designed and built solely to solve the particular task of interest, with both increased performance and better energy efficiency as compared to those achievable by a CPU or GPU. This is another form of heterogeneous computing, although the adoption of ASICs is often limited to certain market niches, due to their specific nature. In research contexts, one of the most currently used ASICs is the tensor processing unit (TPU), developed by Google for neural network machine learning acceleration.[1] Another example is the use of ASICs in the context of bitcoin mining.[2]

FPGAs: RECONFIGURABLE HARDWARE ACCELERATORS

FPGAs are reconfigurable hardware devices. They can be used to synthesize different hardware designs or architectures over and over again. FPGAs were introduced in mid-1980 by Xilinx (now AMD) as a natural evolution of programmable logic devices. They were initially intended to serve as glue logic and a prototype of small digital circuits.

Since the beginning of the premulticore CPUs era, FPGAs appeared as an excellent proof-of-concept device to shorten the software development cycle for ASICs, as this development was allowed to start before any test chip had been manufactured. The increase in the available logic cells, together with large random-access memory blocks, digital signal processor (DSP) arithmetic units, and even embedded microprocessors, moved FPGA usage beyond proof-of-concept prototyping to final production on their own. Thus, in the 2000s, high-performance FPGA-based architectures were developed. At that time, FPGAs already exhibited high efficiency as accelerators of applications in a wide variety of areas, such as cryptography, signal processing, genomics, or pattern recognition, to name just a few. As a consequence, they were adopted as accelerator devices in some supercomputing clusters.[3]

In the mid-2000s, GPUs came into the game as a serious rival of FPGAs. Even if FPGAs were initially competitive against GPUs, the fast development of the latter, and, more importantly, the support of Nvidia delivering the CUDA platform in 2007, restricted FPGAs to embedded application domains where energy efficiency was critical, and GPUs took their place as accelerators in HPC clusters.[4]

In fact, the decline of FPGAs was not just an issue of computing performance or efficiency but also a problem of productivity. The programming of FPGAs required working at the register transfer level (RTL) with intricate hardware description languages (HDLs), such as VHDL or Verilog, which are rather less user-friendly than high-level programming languages and models. As in the case of GPUs, which were first programmed using clever tricks to take advantage of their capabilities, with their vendors later to develop friendlier programming environments, the main FPGA vendors have made efforts to provide high-level synthesis (HLS) tools, such as AMD's Vitis, which allow FPGA applications to be developed from a software perspective, viewing programmable logic as a computational resource instead of a hardware system.[5]

AS FPGAs ALLOW THE PROGRAMMER TO IMPLEMENT CUSTOM HARDWARE ARCHITECTURES, THEY, AT FIRST GLANCE, SEEMED TO BE WELL SUITED FOR HPC COMPUTATIONS.

However, this improvement comes at the cost of increased compilation time. The translation of HLS code to RTL, and from there to the desired FPGA configuration, involves multiple optimization steps to map the design onto the target FPGA architecture, and it usually takes a significant amount of time, on the order of hours.

Progress in the computing capabilities of FPGA technology has also been made so that these kinds of devices might be leveraged in research facilities, data centers, computing centers, and other similar environments. Several projects were also conducted in this regard, such as, for example, the Horizon 2020 Future and Emerging Technologies–HPC EuroEXA project (https://euroexa.eu/) and the Heterogeneous Accelerated Compute Clusters project (https://www.amd-haccs.io/), which remains ongoing. Regarding the integration of FPGAs in data centers, an overview of different developments is provided in Alonso and Bailis.[6]

Concerning vendors, there are currently two main FPGA vendors developing device models that target HPC contexts: Xilinx (property of AMD) and Intel (formerly Altera). Examples of HPC or data center accelerator FPGAs are Xilinx's Alveo and Versal FPGA families and Intel's Stratix 10 and Agilex FPGA family. However, even though data center FPGAs have been available for a few years now, interest among researchers has increased significantly, and although the prospect of near-future FPGA-powered supercomputers has existed since at least 2013,[7,8] there has not been significant adoption of FPGA devices as general-purpose accelerators in the industry. For example, many of the TOP500 list's newest entries are multi-CPU-and-GPU supercomputers. One of the few FPGA-powered supercomputers found in the latest TOP500 list is Noctua 2, inaugurated in 2022, in Paderborn, Germany.

This situation leads us to the following question: Are FPGAs really useful to accelerate HPC workloads, where absolute performance is the ultimate goal? To try to answer this question, we should first understand why FPGA architecture and programmability are so special.

SPECIAL CHARACTERISTICS OF FPGAs

Reconfigurability is the main property of FPGAs. They contain an array of programmable logic blocks as well as reconfigurable interconnections to link these blocks together, which allows them to implement complex logic functions. FPGAs can implement any logical function that an ASIC can perform. Most FPGAs also include memory elements, such as flip-flops, and modern FPGAs even include logic blocks for the fast execution of common low-level computations, such as DSPs for floating-point operations. Although FPGAs are designed to be able to implement (synthesize) arbitrary logic functions, they are limited by their quantity of resources and their clock speed. Thus, high-complexity functions might not be synthesizable into a given FPGA. Nevertheless, the resource amount present in FPGA models has greatly increased over time.

As FPGAs allow the programmer to implement custom hardware architectures, they, at first glance, seemed to be well suited for HPC computations. While common CPU execution must dedicate a significant amount of time and energy fetching and decoding every instruction to execute,[9] these steps and their cost can be avoided in custom hardware designs, where the computations to perform are known beforehand. Moreover, CPU instruction sets are composed mainly of simple operations that are combined to make more complex computations; however, FPGAs can potentially implement those complex computations directly, saving clock cycles in their execution. This includes the implementation of data- or task-parallel computations in hardware. By allowing specific computational tasks to be executed directly in hardware, FPGAs are highly power-efficient devices, reducing the need for general-purpose processor overheads. This direct execution path can significantly lower power consumption, especially for tasks that can be highly parallelized or require specialized processing. However, FPGAs offer lower clock speeds than CPUs. To overcome their limitations, engineers exploit the main strengths of FPGAs: fine- and coarse-grain parallelism as well as the previously mentioned low overhead in computations.

THE COMPILATION OF FPGA CODES IS A TIME-CONSUMING PROCESS, ESPECIALLY WHEN HIGH-LEVEL LANGUAGES ARE USED TO DESCRIBE SOPHISTICATED ALGORITHMS.

Overall, recent improvements in FPGA technology (both in device design and the software stack) have made the use of these devices seemingly viable as accelerators for HPC workloads. They are known for being able to successfully accelerate workloads composed of irregular data types and algorithms when compared to CPU executions as well as for achieving a considerably higher energy efficiency.[10,11] Additionally, FPGAs present a certain innate characteristic that cannot be replicated by any ASIC: reconfigurability. This is a crucial advantage in environments where multiple distinct applications need to be accelerated over different periods of time. Moreover, FPGAs can leverage dynamic partial reconfiguration to modify their behavior on the fly.[12] This possibility increases the accelerator's flexibility further and enables it to widen the number of tasks it can serve without requiring a complete reconfiguration (which incurs higher overheads).

Nevertheless, as devices for heterogeneous computing, it would be more appropriate to compare them with other accelerators used for heterogeneous computing. After all, modern general-purpose supercomputers rarely include only CPUs but, rather, a combination of CPUs and GPUs. This comparison should not only be carried out in terms of absolute performance and energy efficiency but also taking into account programmability and portability issues.

FPGA PROGRAMMABILITY AND PORTABILITY ISSUES

As we stated above, FPGAs are often programmed using HDLs, such as Verilog or VHDL, which provide deep low-level control over the electronic components or behavior of the devices. Although the use of these languages maximizes FPGA performance and minimizes resource utilization, from an HPC perspective, these languages are cumbersome and error prone and incur high development times. The reason is that they are too low level, and HPC engineers are not usually very familiar with the constructs on which they are based. Trying to program HPC kernels entirely with an HDL leads to very high development costs, even more so if the user has to program the entire logic to communicate the

FPGA (device) with the CPU (host) for data movement and task dispatching, which is architecture dependent. Thus, using HDL languages is deemed unfeasible in HPC contexts.

To alleviate these issues, HLS languages and frameworks have been developed, which leverage high-level software programming languages (mainly C based) for hardware design. HLS has succeeded in several areas, including deep learning, video transcoding, graph processing, and genome sequencing.[13] Examples of these languages are Vitis HLS (for Xilinx FPGAs only) and OpenCL (commonly used for Intel FPGAs). OpenCL is one of the most popular ones for Intel FPGAs and previously for Xilinx ones too. OpenCL was designed from the beginning to target heterogeneous systems and allow all their resources to be efficiently exploited, and it has been extensively used for programming CPU + GPU applications. Its design philosophy is to enable code portability across many different computing devices, that is, to be able to write a single device-agnostic code and execute it on any OpenCL-supported device (including CPUs, GPUs, and FPGAs). This, in theory, is perfect for heterogeneous computing, especially for FPGAs. Not only does the language make the complex low-level details of hardware design abstract by using a popular software programming language but it also allows any code written targeting any other accelerator (namely, GPUs) to execute on an FPGA.

Nevertheless, theory and reality are often known to differ. While it is true that OpenCL provides code portability across supported devices, it does not guarantee performance portability. Moreover, its high verbosity and the lack of support from important vendors (for example, Nvidia) have made it less commonly used lately. In the particular case of FPGA accelerators, although they are able to properly execute device-agnostic or GPU-optimized OpenCL code, the performance they achieve with such codes is, in general, considerably low.[14,15] Some optimization techniques are known to alleviate this situation (see "FPGA-Specific Optimization Techniques"). Although we centered our discussion on OpenCL capabilities, it is worth noting that these conclusions may be extended to any programming model or framework targeting different kinds of accelerators (namely, GPUs and FPGAs), such as SYCL and all its derived implementations, although their actual performance depends on the particular application considered and the internal compiler optimizations available. Other pragma-based languages, such as OpenACC and OpenMP, are also used for this purpose.

MOST BANDWIDTH LIMITS ON FPGAs COME FROM THE USE OF DOUBLE DATA RATE 4 TECHNOLOGY, WHILE GPUs HAVE BEEN USING FASTER MEMORY TECHNOLOGY FOR SOME YEARS NOW.

Other languages and frameworks used for high-level synthesis use C pragmas to target particular devices. For example, Vitis HLS uses pragmas to target AMD Xilinx FPGAs. The use of pragmas allows the code to be annotated with different pragmas to target several architectures at the same time. On the other hand, the use of OpenCL forces the rewriting of the code to take advantage of architectures whose vendors do not support OpenCL. Consequently, the use of C pragmas leads to an FPGA-centric design philosophy, which might result in fewer efforts and complexities to optimize naive or device-agnostic codes for FPGA execution. However, this optimization step is still unavoidable.

Overall, concerning the programmability of FPGAs for HPC applications, as of today, it seems unfeasible to rely only on compiler optimizations to efficiently execute device-agnostic code on FPGAs. Therefore, HPC researchers and engineers are expected to have some knowledge of the underlying architecture when trying to maximize performance for FPGA devices.

Moreover, the compilation of FPGA codes is a time-consuming process, especially when high-level languages are used to describe sophisticated algorithms that lead to complex hardware descriptions. HPC kernels for FPGAs are known to take several hours to compile, which further adds to the development costs associated with these devices. Overlay architectures for FPGAs show potential in reducing the long compilation and reconfiguration times traditionally associated with FPGA deployment. By providing a higher-level abstraction, overlays can simplify FPGA programming, making it more accessible and quicker to adapt to different applications.[16] This approach allows for rapid prototyping and iteration, which is crucial in research and development settings.

CURRENT HARDWARE LIMITATIONS OF FPGA DEVICES

In addition to the programmability and portability issues described previously, current FPGA technology presents some significant limitations that hinder achieving high performance, no matter how thoughtful of the underlying architecture the programming might be.

Lower clock frequency

FPGA devices present considerably lower working clock frequencies than other kinds of accelerators. For example, Cong et al.[17] studied the main performance differences between FPGAs and GPUs. The authors claim that the lower clock frequencies are partially alleviated by the fact that FPGAs are able to achieve a higher number of operations per cycle in each computing pipeline than GPUs. However, FPGAs still present a lower effective parallel factor, which makes GPUs the winner in terms of the absolute performance achievable.

Lower memory bandwidth and size

Cong et al. also state that the lower parallel factor presented by FPGAs, described previously, is largely caused by the FPGAs' far lower off-chip

FPGA-SPECIFIC OPTIMIZATION TECHNIQUES

To achieve high performance on FPGA devices, specific code optimizations are needed. These optimizations often differ considerably from CPU or GPU optimizations and require the programmer to be aware of the underlying architecture to a certain degree. The importance of optimizing the code for FPGAs is such that it can make the difference between underperforming and outperforming CPU executions of the same applications. There exist several particular optimization techniques that are known to considerably increase performance for FPGA executions of HPC workloads:

» *Pipeline single-threaded versus ND-range kernels:* Single-threaded loop-pipelined kernels usually achieve higher performance and allow for more FPGA-specific optimizations than multithreaded (also known as *ND-range*) kernels, which are commonly used for GPU and CPU execution. This is usually true even when the single-threaded kernels achieve a lower working frequency than the multithreaded ones since the further optimizations available to the single-threaded kernels enable a much higher number of computations per cycle to be achieved. FPGAs especially benefit from deep pipeline kernels, as executing every independent stage of the pipeline simultaneously on every single clock cycle achieves a computations-per-cycle rate proportional to the number of pipeline stages. Thus, algorithmic refactoring of a kernel might be needed to achieve the highest performance on FPGAs.

» *Memory hierarchy usage:* The memory hierarchy of FPGAs differs significantly from traditional general-purpose accelerators, and the user should take this into account when designing optimized kernels. Among others, the usage of the *restrict* C keyword, which is used in pointer declarations to indicate to the compiler that no other pointer will be used to access the object to which it points, usually provides a noticeable performance improvement. Some other well-known constructs, such as the use of shift registers and sliding window strategies, are able to efficiently exploit the FPGA resources and achieve high performance.

» *Other manual optimizations:* In general, automatic compiler optimizations do not achieve performance increments comparable to those of manual optimizations. In particular, manual loop unrolling and manual vectorization often result in increased performance.

More advanced optimizing transformations, including pipelining, data reuse, and resolving interface contention, are discussed in de Fine Licht et al.[S1]

REFERENCE

S1. J. de Fine 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, doi: 10.1109/TPDS.2020.3039409.

memory bandwidth. Low memory bandwidth is the other most important limitation of current FPGA devices, and it probably constitutes the main limiting factor for FPGAs to achieve high performance in numerous applications. Most bandwidth limits on FPGAs come from the use of Double Data Rate 4 technology, while GPUs have been using faster memory technology for some years now. This limitation is even more relevant when considering that available FPGA boards do not support the memory sizes available in GPUs, and getting data in and out of these cards is expensive and can easily destroy any potential benefit in the computation. FPGAs are designed for flexibility and programmability, with their architecture consisting of an array of programmable logic blocks and routing. This flexibility comes at the cost of not being optimized for high memory bandwidth in the same way GPUs are since GPUs are designed with parallelism and high-bandwidth memory interfaces from the outset.

Zohouri et al.[18] present a comprehensive analysis of the memory controller and memory bandwidth efficiency of Intel FPGAs, concluding that to achieve high memory performance, FPGA kernels must meet multiple and strict requirements related to access patterns, alignment, and memory hierarchy usage. These requirements are hard to meet in real-world applications, and thus, for many applications, it might not be possible to achieve more than 70% of the peak memory bandwidth. Overall, the low off-chip memory bandwidth compared to CPUs and GPUs, as well as the difficulties to efficiently exploit that bandwidth, put FPGA accelerators at a disadvantage against other accelerators for many applications.

The cited works conducted their research using older FPGA and GPU models, so their conclusion might not seem representative of the current state of the art. To provide some insight into how the state of the art might have changed since those works were published, Table 1 provides a comparison of clock frequencies and memory bandwidths among different Intel FPGA and Nvidia GPU models, including recent ones. Comparing FPGAs and GPUs just in terms of maximum clock frequency is an oversimplification based on theoretical hard limits and should be taken lightly. It is worth noting that FPGA working clock frequencies depend on the specific hardware design synthesized and rarely come close to the reported theoretical maxima (shown in the table), especially when

TABLE 1. A comparison of the clock frequency and peak memory bandwidth of several Xilinx and Intel FPGAs and Nvidia GPU models, sorted by release date.

	Release date	**Processing clock frequency**	**Peak memory bandwidth**
Virtex UltraScale+	First quarter 2016	Up to 819 MHz	76.8 GB/s
Nvidia Tesla V100 GPU	First quarter 2017	1,245 MHz (base), 1,380 MHz (boost)	900 GB/s
Intel PAC with Intel Arria 10 GX FPGA	Fourth quarter 2017 (FPGA model from 2013)	Up to 800 MHz	34.8 GB/s
Intel FPGA PAC D5005 (with Intel Stratix 10 GX)	Fourth quarter 2019 (FPGA model from 2013)	Up to 1,000 MHz	76.8 GB/s
Intel Stratix 10 MX* FPGA	FPGA model from 2017	Up to 1,000 MHz	512 GB/s
Nvidia A100 GPU	First quarter 2020	765 MHz (base), 1,410 MHz (boost)	1,555 GB/s
Xilinx Alveo U55C	Fourth quarter 2021	Up to 1,028 MHz	460 GB/s
Intel Agilex 7 FPGA* M-Series 039	FPGA model from first quarter 2022	Up to 800 MHz	1,000 GB/s
Nvidia H100 GPU	First quarter 2022	1,095 MHz (base), 1,755 MHz (boost)	2,039 GB/s

*The entry is an FPGA integrated circuit (chiplet) model to be integrated into a hardware package or module with other components, not a commercially available ready-to-use accelerator itself.

using a high-level synthesis language and compiler (such as OpenCL) instead of HDLs. For compute-intensive or HPC kernels, the cited works report working frequencies that usually range between 200 and 300 MHz on Arria 10 GX devices, which is 20% to 37.5%, respectively, of their reported peak frequency. In the case of Stratix 10 GX devices, it is usual to achieve working frequencies that range between 300 and 400 MHz (30% to 40% of their peak frequency).

Lower floating-point performance

One of the major limitations of FPGAs in the context of HPC seems to be the low single- and double-precision floating-point performance that could be achieved with such devices.[19] Some recent research works, such as two by Calore and Schifano[20,21] attempted to measure it in the context of the roofline model. These works offer a performance estimation of function kernels developed with high-level synthesis tools, revealing some of the FPGA limitations in the context of HPC.

ON THE POTENTIAL OF FPGAs IN MODERN HPC WORKLOADS

Regarding the question of whether current FPGAs are suitable to accelerate modern HPC workloads, we now contextualize the points discussed above for the case of real-world applications. To review the potential of FPGAs in HPC contexts, not only the absolute performance of FPGAs when executing common HPC tasks must be considered but also the performance relative to other accelerators since the potential of FPGAs is conditioned by the other available alternatives for HPC accelerators.

Back in 2014, Véstias et al.[19] reviewed the trends of CPU, GPU, and FPGA devices for HPC. Their conclusions were that FPGAs were not keeping pace with other platforms in terms of performance, which caused HPC applications to be migrated to other more powerful platforms, such as software-based manycore systems (CPUs and GPUs). The authors noted that FPGAs become competitive when working with applications with specific constructs or requirements for which general-purpose computing devices are not suited (for example, applications with operands with custom/user-defined data widths as well as combinational logic problems, finite-state machines, and parallel MapReduce problems). The authors also noted a trend for the HPC community toward adopting hybrid (heterogeneous) platforms with a mix of different kinds of devices working together. As that work is almost a decade old, it is worth reviewing more recent works so as to analyze whether these trends have continued. For example, in 2020, Nguyen et al.[11] explored the potential of FPGAs in HPC environments, testing both an Intel FPGA (Arria 10 GX) and a Xilinx FPGA (Alveo U280) and comparing them against other accelerators (namely, an Intel Xeon CPU and Nvidia V100 GPU). They found the single-precision FPGA performance and bandwidth still fall far below GPUs for compute and memory-intensive tasks; however, FPGAs can deliver nearly the same energy efficiency as GPUs for most applications, and even exceed it in some cases. The authors also noted that FPGAs are likely to continue being competitive in areas for which GPU and CPU computing models do not match the nature of the problem. Their work led them to interesting conclusions. First, they point to the low memory bandwidths of the FPGAs as the main limiting factor for achieving high performance on FPGAs. Second, they also note that exploiting such bandwidths is rather difficult, while only a very small fraction of the theoretical memory bandwidth is achieved by unoptimized codes. Third, they note that FPGAs are not power-proportional devices, in the sense that a significant increase in performance might require only a moderate increase in power consumption. This contrasts with CPUs and GPUs, where the power consumption increase is more pronounced. Their conclusions were that vendors should prioritize maximizing development productivity for FPGAs rather than increasing their amount and type of resources, as their FPGA implementations required greater orders of magnitude of software development time than the equivalent (and often superior) CPU and GPU implementations. Other works[22] note that FPGAs might present certain advantages compared to GPUs for applications that

FPGAs BECOME COMPETITIVE WHEN WORKING WITH APPLICATIONS WITH SPECIFIC CONSTRUCTS OR REQUIREMENTS FOR WHICH GENERAL-PURPOSE COMPUTING DEVICES ARE NOT SUITED.

can exploit temporal blocking or other forms of high pipelined parallelism.

FPGAs can be beneficial in scientific computing applications where latency and predictability of execution times are crucial, such as in urgent HPC scenarios, including interactive prototyping, urgent streaming data analysis, application steering, and in situ visualization. There are several reasons for this. First, FPGAs excel in providing low-latency processing. Unlike CPUs and GPUs, which have fixed hardware structures and instruction sets, FPGAs can be configured to perform specific computations directly in hardware, reducing their overhead. This is particularly relevant in the case of irregular applications, where the single-instruction, multiple data paradigm cannot be applied. Second, FPGAs offer more predictable performance compared to CPUs and GPUs. Since FPGAs can be configured with specific hardware paths for given tasks, they can execute these tasks consistently without the unpredictability introduced by shared resources (like caches or memory buses) in general-purpose processors. This predictability is critical in applications where timing and consistency of computation are vital. Third, FPGAs can be tailored for specific algorithms or data processing tasks. This customization allows for highly efficient execution of particular tasks in scientific computing, such as data analysis or simulation, which can be critical in urgent computing scenarios where quick accurate results are required. Due to these reasons, FPGA benefits in scenarios like the ones described previously can be substantial, especially when immediate data processing and decision making are crucial.

For some years, FPGAs have been considered well suited for deep learning computations, due to the pipelined nature of deep learning models and the potential to optimize them by means of custom irregular data types as well as irregular algorithms. Nurvitadhi et al.[23] conclude that recent trends in deep neural network algorithms might favor FPGAs over GPUs and that FPGAs have potential to become the platform of choice for accelerating deep neural networks, offering superior performance. Nevertheless, the deep learning market has been of significant importance in recent times, and many vendors of electronic components (including CPUs, GPUs, and ASICs) have tried to get into and expand inside that market. Since the publication of that work, two major breakthroughs have been made concerning hardware acceleration of deep learning applications. First, many GPUs have started to include dedicated hardware for AI acceleration, such as Nvidia's tensor cores. Second, Google launched the TPU, an ASIC for AI acceleration. These new kinds of hardware attempt to accelerate AI tasks that GPUs are not well suited for, including algorithms dealing with custom irregular data types. Thus, both of them pose new challenges to FPGAs to become the platform of choice to accelerate deep neural networks. Over recent years, we have seen a significant increment in TPU and tensor core GPU utilization for accelerating real-world AI tasks; however, FPGAs do not seem to have made significant progress in this field, and nowadays, they are not the platform of choice for accelerating large-scale deep neural networks.

WHAT ABOUT THE USE OF FPGAs AS COOPERATIVE DEVICES IN HETEROGENEOUS SYSTEMS?

Besides the potential use of FPGAs as stand-alone devices for the acceleration of HPC workloads, whose strengths and limitation were described previously, there is also certain interest in studying the potential use of FPGAs cooperatively with other devices, both FPGAs and other accelerators, so as to exploit all the available resources of a given heterogeneous potentially distributed system.

Many works explore the possibilities of using FPGAs cooperatively in heterogeneous environments. Some explore the possibility of using FPGA-powered network interface cards to carry out CPU-less processing of incoming and outcoming network data, thus reducing latency. This can be applied to inter-FPGA communications to efficiently connect multiple distributed FPGAs together. Other works discuss direct memory access (DMA) mechanisms to connect GPUs and FPGAs together in order to efficiently communicate different kinds of accelerators from different perspectives: either the GPU is the peripheral component interconnect express (PCIe) master or this task is assigned to the FPGA. Both approaches show that performance penalties are incurred for DMA transfers in which the PCIe master is the destination device. These two techniques can be combined to enable efficient cooperative work between GPUs and FPGAs over different nodes.[24]

Although these are promising techniques for heterogeneous environments, there do not seem to be many real case applications that clearly benefit from cooperative FPGA approaches. While GPU technology is making significant progress in the distributed multi-GPU field for real-world applications, the multi-FPGA and hybrid GPU–FPGA fields for real-world applications seem to be considerably less explored. One major cause for this seems to be the lower scaling capabilities of FPGA devices, which hinder the development of multi-FPGA solutions, as well as the higher hardware costs, which hinder the integration of FPGAs into heterogeneous clusters already including GPUs. Besides this, the existence of applications with computational patterns that would benefit from simultaneous GPU and FPGA acceleration is still unclear.

Overall, modern FPGA technology focused on HPC environments still presents important limitations that put FPGA devices at a disadvantage compared to GPUs, namely, low memory bandwidth and size, lower raw computational power, the need for sophisticated manual tuning due to poor automatic compiler optimizations, development complexity, and very long compilation times. FPGAs can still prove useful in the acceleration of irregular tasks for which general-purpose architectures (CPU and GPU) are poorly optimized, such as tasks with irregular data types or algorithms, as long as it is not profitable to build and deploy ASICs for those applications. FPGAs also show potential for accelerating tasks in environments where flexibility and/or energy efficiency are crucial. Nevertheless, FPGA technology still has to make some progress, both in hardware capabilities and ease of development, to become competitive at accelerating most modern HPC workloads. C

ACKNOWLEDGMENT

The work of Manuel de Castro, Yuri Torres, and Diego R. Llanos has been supported in part by Grant PID2022-142292NB-I00 (NATASHA Project), funded by MCIN/AEI/10.13059/501100011033, and by the European Regional Development Fund's A Way of Making Europe project. Yuri Torres and Diego R. Llanos have been supported in part by Junta de Castilla y León FEDER Grant VA226P20 (PROPHET-2 Project). Diego R. Llanos has been supported in part by Grant TED2021-130367B-I00, funded by MCIN/AEI/10.13039/501100011033, and by Next Generation EU Plan de Recuperación, Transformación, y Resiliencia. The work of David L. Vilariño has been supported by Grants PID2022-141623NB-I00 and PID2019-104834GB-I00 (funded by MCIN/AEI/10.13039/501100011033/FEDER, UE) and by the Conselleria de Cultura, Educacion, e Ordenacion Universitaria, Xunta de Galicia (Accreditation ED431C 2022/16). Thanks to the anonymous reviewers for many useful suggestions.

REFERENCES

1. N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, Jun. 2017, doi: 10.1145/3140659.3080246.
2. M. Bedford Taylor, "The evolution of bitcoin hardware," *Computer*, vol. 50, no. 9, pp. 58–66, Sep. 2017, doi: 10.1109/MC.2017.3571056.
3. C. Stephen et al., "Examining the viability of FPGA supercomputing," *EURASIP J. Embedded Syst.*, vol. 2007, Jan. 2007, Art. no. 93652, doi: 10.1155/2007/93652.
4. D. H. Jones et al., "GPU versus FPGA for high productivity computing," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2010, pp. 119–124, doi: 10.1109/FPL.2010.32.
5. N. Brown, "Weighing up the new kid on the block: Impressions of using Vitis for HPC software development," in *Proc. 30th Int. Conf. Field-Programmable Logic Appl. (FPL)*, 2020, pp. 335–340, doi: 10.1109/FPL50879.2020.00062.
6. G. Alonso and P. Bailis, "Research for practice: FPGAs in datacenters," *Commun. ACM*, vol. 61, no. 9, pp. 48–49, 2018, doi: 10.1145/3209275.
7. M. Baity-Jesi et al., "An FPGA-based supercomputer for statistical physics: The weird case of Janus," in *High-Performance Computing Using FPGAs*, W. Vanderbauwhede and K. Benkrid, Eds., New York, NY, USA: Springer-Verlag, 2013, pp. 481–506.
8. "Top500 NEWS: Good times for FPGA enthusiasts," Top500, Sinsheim, Germany, 2016. Accessed: Sep. 2023. [Online]. Available: https://www.top500.org/news/good-times-for-fpga-enthusiasts
9. S. R. Hines, "Improving processor efficiency through enhanced instruction fetch," Ph.D. thesis, Florida State Univ., Tallahassee, FL, USA, 2008.
10. S. Kestur, J. D. Davis, and O. Williams, "BLAS comparison on FPGA, CPU and GPU," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Piscataway, NJ, USA: IEEE, 2010, pp. 288–293, doi: 10.1109/ISVLSI.2010.84.
11. 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)*, Piscataway, NJ, USA: IEEE, 2020, pp. 8–19, doi: 10.1109/PMBS51919.2020.00007.
12. K. Vipin and S. A. Fahmy, "FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018, Art. no. 72, doi: 10.1145/3193827.
13. J. Cong et al., "FPGA HLS today: Successes, challenges, and opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, pp. 1–42, Aug. 2022, doi: 10.1145/3530775.
14. K. Krommydas et al., "Bridging the performance-programmability gap for FPGAs via OpenCL: A case study

ABOUT THE AUTHORS

MANUEL DE CASTRO is a Ph.D. candidate in the Department of Computer Science, University of Valladolid, 47011 Valladolid, Spain. His research interests include parallel and distributed computing and GPU and field-programmable gate array programming. de Castro received a M.S. in computer science from the Universidade de Coruña. Contact him at manuel@infor.uva.es.

DAVID L. VILARIÑO is an associate professor in the Department of Electronic and Computation, University of Santiago de Compostela, 15782 Santiago de Compostela, Spain. His research interests include the design of algorithms and special-purpose hardware modules for reconfigurable architectures (field-programmable gate array and coarse-grain reconfigurable architecture), with a focus on fast and efficient computation. Vilariño received a Ph.D. in computer science from the Universidade de Santiago de Compostela. Contact him at david.vilarino@usc.es.

YURI TORRES is an associate professor in the Department of Computer Science, University of Valladolid, 47011 Valladolid, Spain. His research interests include parallel and distributed computing, parallel programming models, and embedded computing. Torres received a Ph.D. in computer science from the University of Valladolid. Contact him at yuri.torres@infor.uva.es.

DIEGO R. LLANOS is a full professor in the Department of Computer Science, University of Valladolid, 47011 Valladolid, Spain. His research interests include parallel and distributed computing, the Internet of Things, and embedded systems. Llanos received a Ph.D. in computer science from the University of Valladolid. He is a Senior Member of IEEE and the Association for Computing Machinery. Contact him at diego.llanos@uva.es.

with OpenDwarfs," in *Proc. IEEE 24th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, 2016, pp. 198–198, doi: 10.1109/FCCM.2016.56.

15. H. R. Zohouri et al., "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2016, pp. 409–420, doi: 10.1109/SC.2016.34.
16. H. K.-H. So and C. Liu, "FPGA overlays," in *FPGAs for Software Programmers*, D. Koch, F. Hannig, and D. Ziener, Eds., Cham, Switzerland: Springer-Verlag, 2016, pp. 285–305.
17. J. Cong et al., "Understanding performance differences of FPGAs and GPUs," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, New York, NY, USA: Association for Computing Machinery, 2018, p. 288, doi: 10.1145/3174243.3174970.
18. H. R. Zohouri et al., "The memory controller wall: Benchmarking the Intel FPGA SDK for OpenCL memory interface," in *Proc. IEEE/ACM Int. Workshop Heterogeneous High-Perform. Reconfigurable Comput. (H2RC)*, Nov. 2019, pp. 11–18, doi: 10.1109/H2RC49586.2019.00007.
19. M. Véstias et al., "Trends of CPU, GPU and FPGA for high-performance computing," in *Proc. 24th Int. Conf. Field Programmable Logic Appl. (FPL)*, 2014, pp. 1–6, doi: 10.1109/FPL.2014.6927483.
20. E. Calore and S. F. Schifano, "Performance assessment of FPGAs as HPC accelerators using the FPGA empirical roofline," in *Proc. 31st Int. Conf. Field-Programmable Logic Appl. (FPL)*, Piscataway, NJ, USA: IEEE, 2021, pp. 83–90, doi: 10.1109/FPL53798.2021.00022.
21. E. Calore and S. F. Schifano, "FER: A benchmark for the roofline analysis of FPGA based HPC accelerators," *IEEE Access*, vol. 10, pp. 94,220–94,234, 2022, doi: 10.1109/ACCESS.2022.3203566.
22. H. R. Zohouri et al., "Combined spatial and temporal blocking for high-performance stencil computation on FPGAs using OpenCL," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, New York, NY, USA: Association for Computing Machinery, 2018, pp. 153–162, doi: 10.1145/3174243.3174248.
23. E. Nurvitadhi et al., "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, New York, NY, USA: Association for Computing Machinery, 2017, pp. 5–14, doi: 10.1145/3020078.3021740.
24. R. Kobayashi et al., "OpenCL-enabled high performance direct memory access for GPU-FPGA cooperative computation," in *Proc. HPC Asia Workshops (HPCAsia Workshops)*, New York, NY, USA: Association for Computing Machinery, 2019, pp. 6–9, doi: 10.1145/3317576.3317581.