The Performance and Energy Efficiency Potential of FPGAs in Scientific Computing

Tan Nguyen, Samuel Williams
Computational Research Division
Lawrence Berkeley National Laboratory
{*TanNguyen, SWWilliams*}*@lbl.gov*

Marco Siracusa
DEIB
Politecnico di Milano
marco.siracusa@mail.polimi.it

Colin MacLean, Douglas Doerfler, Nicholas J. Wright
National Energy Research Scientific Computing
Lawrence Berkeley National Laboratory
{*ColinMacLean, DWDoerf, NJWright*}*@lbl.gov*

***Abstract*—Hardware specialization is a promising direction for the future of digital computing. Reconfigurable technologies enable hardware specialization with modest non-recurring engineering cost. In this paper, we use FPGAs to evaluate the benefits of building specialized hardware for numerical kernels found in scientific applications. In order to properly evaluate performance, we not only compare Intel Arria 10 and Xilinx U280 performance against Intel Xeon, Intel Xeon Phi, and NVIDIA V100 GPUs, but we also extend the Empirical Roofline Toolkit (ERT) to FPGAs in order to assess our results in terms of the Roofline Model. Although FPGA performance is known to be far less than that of a GPU, we also benchmark the energy efficiency of each platform for the scientific kernels comparing to microbenchmark and technological limits. Results show that while FPGAs struggle to compete in absolute terms with GPUs on memory- and compute-intensive kernels, they require far less power and can deliver nearly the same energy efficiency.**

***Keywords*-Hardware Reconfigurability, FPGAs for HPC, Empirical Roofline Toolkit**

I. INTRODUCTION

The last decade has seen a paradigm shift in HPC centers as year-over-year CPU performance slowed and power emerged as a major constraint. In order to meet the ever increasing demands for higher workload performance and capability in a power- and cost-constrained environment, HPC centers have the incentive to explore alternative technologies. Fifteen years ago researchers began experimenting with GPUs, hypothesizing that the high throughput performance and low cost demands of gaming would synergize with the needs of HPC. Although GPUs certainly had their deficiencies, five years of evolution, innovation, and adaptation made them viable and another ten years of performance scaling made them generally superior to multicore and manycore CPU offerings in many cases. As a result, today many of the top HPC centers in the world have embraced GPUs or some other form of accelerated computing.

Although GPUs can satisfy the throughput computing requirements of many applications, they have their limitations. In order to tap into the full potential of a GPU, programmers must rewrite their applications in a hybrid programming model using different models for distributed memory communication, on-node shared memory computation (multicore), and accelerated (GPU) computation. GPUs, being highly parallel architectures, require massive, coarse-grained parallel operations to attain superior performance. Although they have embraced double-precision arithmetic (essential for numerical simulations) and 16-bit precision (essential for machine learning), they have limited support for narrow integer data types (int4, int8) that might be needed in the fields of bioinformatics or graph analytics. Perhaps more limiting in the distributed memory environment, the small-message communication performance of a GPU can often suffer compared to a latency-optimized CPU. This can result in the raw compute potential being underutilized.

As GPUs transition from principally a graphics processor into a hyperscalar data center processor optimized for deep learning, their power constraints have been unbridled (enabling higher energy efficiency on AI codes) while demand has led to a substantially high price. These trends imperil the suitability for GPUs in the HPC environment where energy efficiency is not predicated on AI performance, and there is price sensitivity.

At the National Energy Research Scientific Computing Center (NERSC) [1], one can find many large-scale applications in various science domains, including chemistry, nuclear physics, astrophysics, climate, and life science. Many of these codes have been heavily-optimized for multicore CPUs and GPUs. Nevertheless, there is a sizable fraction of the NERSC workload for which CPUs are not currently used [2]. Thus, even if a GPU were to provide infinite speedup for NERSC's GPU-accelerated applications, the net benefit to overall center performance (throughput) is limited to a factor of 2-3×.

Unlike the traditional Von Neumann instruction processor architectures (including both CPUs and GPUs) where programs, stored in memory, are sequences of instructions, Field-Programmable Gate Arrays (FPGA) represent a distinct class of reconfigurable spatial architectures in which the entirety of the program is realized as a sequential logic circuit in hardware. Thus, instead of instructions being fetched, decoded, and executed on time-multiplexed functional units, operations can be executed in a pipelined manner on a FPGA by sending them through the circuit. There is no instruction decode, register files, or caches. Rather, FPGAs are built from an array of reconfigurable logic blocks called LUTs (Look Up Tables) that the compiler configures and interconnects to form a sequential logic circuit. Newer FPGA architectures have instantiated hardened functional units for arithmetic, integrated registers and block RAM (BRAM) for

local storage, included integrated ARM cores and NICs, and use the latest HBM memory technology. Ultimately, FPGAs allow users to create a custom architecture optimized for each computational kernel in their program (for example using FIFOs instead of caches or 3-element SIMD units).

In this paper, we explore the potential for FPGAs in the HPC environment along the axes of performance, energy efficiency, and programmability. To that end, we evaluate both Intel's Arria 10 GX1150 and Xilinx's Alveo U280. We commence with a study of the peak performance and efficiency as a function of data reuse using the well-established Roofline Model [3] to frame the conversation. We then proceed by examining three HPC kernels (SGEMM, SpMV, and Smith-Waterman) spanning a range of compute intensity, parallelism, synchronization requirements, and underlying data type. In order to provide context, we compare against both the energy efficiency of underlying memory technologies (an ideal architecture imposes as little energy overhead as possible) as well as existing CPU and GPU architectures — Intel Xeon (Haswell), Intel Xeon Phi (Knights Landing), and NVIDIA V100 (Volta) GPU. This highlights the value of DDR and MCDRAM/HBM memory technologies as well as multicore, manycore, and GPU architectures.

II. Related Work

Over the last decade, several publications investigated the benefits of using FPGA devices in specific domains such as Machine Learning [4–6], Linear Algebra [7, 8] and Image Processing [9, 10]. However, the resulting considerations are not general enough to provide a thorough FPGA characterization. In this direction, other works [11–15] analyzed FPGA performance by accelerating common benchmark suites and comparing the obtained results with other architectures. In particular, Cong et al. [15] proposed an FPGA-GPU comparison on the Rodina [16] benchmarks and provided a performance breakdown based on an analytical model. By means of this method, the authors identified the low FPGA memory bandwidth as the main limiting factor in several benchmarks. However, the authors used the analytical model to estimate the performance benefit of running the kernels on higher-bandwidth FPGA boards such as the Xilinx Alveo U280 now available on the market. In fact, this board provides an aggregate bandwidth an order of magnitude higher than previous DDR-based FPGA boards, tightening the gap between FPGA and GPU bandwidth availability. The projected results justify the effort several authors recently spent in benchmarking [17] and microbenchmarking [18] this device. However, despite the appealing results achieved by the authors, these analyses have not been directly compared against other accelerators.

The proposed work, instead, considers DDR-based and newer HBM-based FPGA boards for a performance and power efficiency comparison with other accelerators such as multi-core CPUs and GPUs. This comparison is done through a selection of kernels that stress several FPGA components under different loads. In this way, we enable an easier and parametric performance breakdown better highlighting architectural limitations. As in Cong et al. [15], we discuss our considerations through a performance model (i.e. Roofline). Although the literature already proposes some FPGA Roofline model formulations [19, 20], these works mainly focus on FPGA optimization. As such, the authors do not discuss any architectural characterization nor cross-architectural comparison by means of this model. Moreover, these works do not take into account energy-efficiency [21, 22], a fundamental aspect for FPGA devices.

III. Experimental Setup

A. Evaluated Architectures

In this paper, we evaluate the performance and energy potential of two FPGAs — the Intel Arria 10 GX 1150 [23] and the Xilinx Alveo U280 [24]. The relevant features of the two architectures are summarized in Table I. Both FPGAs integrate over one million LUTs and thousands of hardened multipliers, however their theoretical peak performance is lower than a GPU owing to their greatly reduced nominal frequency. Many HPC applications have a low arithmetic intensity (FLOP:Byte ratio) which places high demands on the memory subsystem. Both FPGAs include 50-66MiB of BRAM (far more than the typical CPU or GPU cache capacity) allowing for software-defined architectures to exploit spatial and temporal locality. This is complemented by ample register space. However, the Arria 10 only includes two channels of DDR memory limiting its memory bandwidth to 34GB/s. This is far less than a typical CPU and roughly 25× lower than the typical GPU. Conversely, the U280 includes both DDR and HBM memory, the latter providing a little better than half the bandwidth of a modern GPU. In this paper, we will benchmark these architectures to determine their attainable memory bandwidth and compute potential.

In order to provide context and comparisons to contemporary architectures, we also run experiments on CPU and GPU systems at NERSC [25]. Whereas the CPUs are DDR-based and the GPUs-HBM based, we also run on NERSC's Knights Landing (KNL) system in order to explore CPU cores with MCDRAM (HBM-like) memory technology.

The V100 (Volta) is NVIDIA's flagship GPU. It includes 5,120 single-precision FMAs running at 1.5GHz and more than 800GB/s of HBM memory bandwidth. Although this provides exceptional peak performance, it is predicated on users expressing massive data parallelism and comes with more than a 200W power requirement. Memory latency is hidden via massive multithreading. As such, performance and energy efficiency is highly application-dependent.

NERSC's Cori system includes two partitions. The first uses conventional CPUs in the form of dual-socket Intel Xeon E5-2698 v3 (Haswell) nodes. Each node includes 32 cores running at a nominal 2.3GHz supporting AVX2 (total

Resource	Intel Arria 10 GX1150	Xilinx Alveo U280	Intel Haswell (2P×16c)	Intel Knights Landing (68c)	NVIDIA V100
Peak Frequency	0.45GHz	0.45GHz	2.3GHz	1.4GHz	1.53GHz
Logic Blocks	1150K Logic Elements	1,304K LUTs	-	-	-
32b FPUs	1,518 DSPs	9,024 slices	512	2,176	5,120
Theoretical Peak	1.37 TF/s	-	2.35 TF/s	5.2 TF/s	15.7 TF/s
SRAM	66MiB BRAM	53MiB BRAM	80MiB L3$	1MiB L2$/2 cores	6 MiB L2
			256KiB L2$/core	32KiB L1$/core	96KiB L1$/SM
Registers	213KB	326KB	5KiB/core	4KiB/core	256KiB/SM
DDR Pin Bandwidth	34GB/s	38GB/s	136GB/s	-	-
HBM Pin Bandwidth	-	460GB/s	-	460GB/s	900GB/s

Table I: Hardware specifications of studied processor architectures

of 512 FMAs). Collectively, the node's 8 DDR-3 channels provide a theoretical pin bandwidth of 136GB/s.

For throughput-intensive applications, the Intel Xeon Phi 7250 (Knights Landing) provides the bulk of Cori's performance. Each manycore processor includes 68 cores each with two AVX-512 vector units (2176 32b FMAs) running at 1.4GHz. Each of the single socket nodes instantiates 16GiB of MCDRAM memory providing more than 460GB/s of memory bandwidth. Like the V100 GPU, and to a lesser extent the Haswell CPU, attaining peak performance on the KNL processor is predicated on massive data parallelism. However, unlike the GPUs, both KNL and Haswell exploit hardware stream prefetchers to hide memory latency.

B. Programming Models

As this paper is focused on the potential performance and energy efficiency gains FPGAs might provide over CPUs or GPUs, we are willing to sacrifice some degree of productivity and portability in order to maximize performance. Although writing codes in RTL might maximize FPGA performance, such low-level programming is anathema to the large, long-lived HPC codes found at NERSC. Rather, programming in a high-level language and enduring programming model is prized. To that end, most FPGA code is written in OpenCL (Arria 10 spatial locality benchmark used DPC++) as OpenMP on FPGAs was judged to be too immature. Although OpenCL can be used for CPUs and GPUs, OpenCL software stacks have fallen behind compared to contemporary OpenMP and CUDA compilers. Thus, we use OpenMP for Haswell and KNL and CUDA for the V100. There is an exception with the matrix multiply kernel where we use MPI to scale to all the 32 cores of two Haswell processors. This choice comes from the underlying SUMMA algorithm [26] which can hide communication cost well, but was presented with only an MPI implementation.

C. Performance and Power Instrumentation

Arria 10 and Alveo U280 power measurements were conducted using the FPGA's self-reported statistics accessed through the Intel Open Programmable Acceleration Engine (OPAE) and Xilinx Board Utility, respectively. To measure performance of our benchmarks, we averaged multiple trials. The number of trials was dependent upon the amount of data read each trial, with longer reads using fewer trials. In a few cases where we report the maximum performance among trials, we explicitly mention this methodology in the performance discussion.

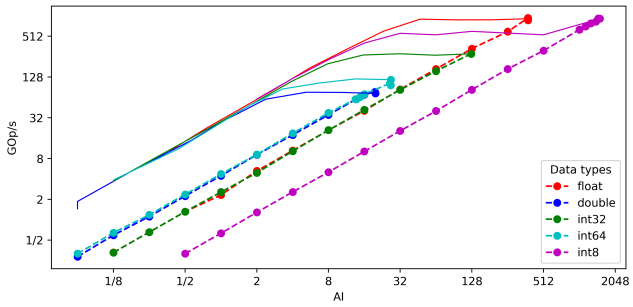
IV. FPGA Microbenchmarks

Although CPUs and GPUs can often sustain a high fraction of compute or bandwidth, there have been instances where sustained performance falls well below peak performance. As theoretical (marketing) numbers cannot always be trusted, it falls on the user to benchmark their systems to provide realistic guidance as to architecture performance. In this paper, we characterize FPGAs along two axes: performance as a function of temporal locality (arithmetic intensity) and bandwidth as a function of spatial locality. The former allows us to assess how well the FPGA software stack can identify and exploit reuse in a pipeline as well as an instruction processor can exploit reuse through a hardware cache. The latter informs us of how well the FPGA software stack can utilize the memory subsystem under different memory access patterns.

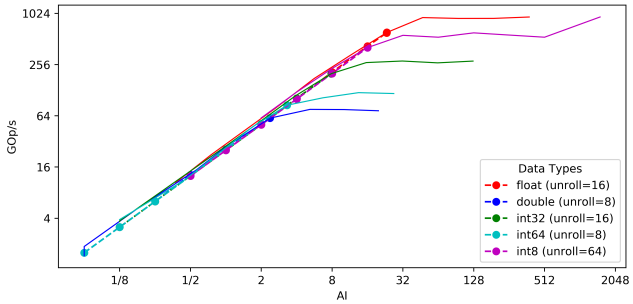
A. Temporal Locality (Roofline)

The Roofline model visualizes bottlenecks by plotting architectural performance bounds and applications characteristics in a performance-data locality 2D plane [3]. The performance (upper) bound is defined as a curve in the plane wherein the arithmetic intensity (FLOPs per Byte) is the data locality x-axis. Such visualizations allow us to understand how well an architecture responds to increases in data locality. Ideally, there is a linear relationship up to the maximum performance of the machine. For brevity, we will only present results obtained on the Arria 10.

In this work, we port the Empirical Roofline Tool (ERT) [27, 28] to OpenCL and enable user-selection of data type (`float`, `int32`, `int8`, etc...). ERT is premised around evaluating an arbitrary degree polynomial for each element of a vector. As one increases the degree of the polynomial, one increases arithmetic intensity (more FLOPs per byte). As one varies the vector size, one varies the cache working set and quantifies bandwidth tapering within the cache hierarchy. As one varies the data type, one varies the natural quanta for memory access.



(a) Single element DRAM transfers



(b) Optimal DRAM transfers

Figure 1: Arria 10 ERT Roofline plot for `float`, `double`, `int8`, `int32`, and `int64` (dotted lines). Prior to tuning (a), Roofline bandwidth depends on data type. Conversely, after tuning (b), Roofline bandwidth on the Arria 10 mimics the traditional Roofline. ERT polynomials create straight lines on an FPGA architecture as pipeline synthesis adds proportional compute and data movement as polynomial degree increases. FPGAs are instead limited by available resources to synthesize such data flow pipelines, with resource-limited Roofline ceilings shown in solid colors. Optimal unrolling fills a 512-bit wide DRAM transfer.

As FPGAs do not have a hardware cache hierarchy, we do not observe any benefit for reduced vector sizes unless data is explicitly placed in BRAM. Moreover, whereas CPU cache hierarchies regiment DRAM and SRAM data movement in quanta of cache lines, FPGAs provide natural width access to DRAM up to the hardware controller limit while BRAM controllers are synthesized for purpose with many configuration options. Although a wide transaction to DRAM may be initiated, without a cache, only the data needed within a given clock cycle is returned (neighboring intra-line data is discarded). Ultimately, exploitation of spatial locality is a fine balance between synthesized frequency, data parallelism, compile-time knowledge of memory access pattern, and the compiler's ability to synthesize a burst-coalesced load store unit. Thus, it is imperative one benchmark the system in order to quantify attainable bandwidth.

Figure 1(a) highlights that unlike CPUs and GPUs, FPGA bandwidth without tuning can vary from 0.6 (`int8`) to 5GB/s (`double`/`int64`) — far less than the theoretical memory bandwidth of 34GB/s. This is shown to be a pipeline bottleneck by dividing measured bandwidth by the kernel clock frequency of roughly 312MHz. The approximately 2 bytes for `int8` and approximately 16 bytes for `double` and `int64` is exactly the amount of data read and written each clock cycle. Unlike instruction processors where hardware is dedicated for either bandwidth or compute, resources on FPGAs are fungible. As a result, we define a set of compiler-derrived theoretical Roofline ceilings based on available FPGA resources and arithmetic intensity.

Two major approaches to optimization can improve bandwidth. First, unlike CPUs and GPUs which run at a relatively constant frequency, FPGA frequency is highly dependent on the compiler. We attained a 34% increase in bandwidth through the addition of the `__fpga_reg()` intrinsic in order to exploit reuse within the polynomial. Second, one can increase data parallelism (replicate pipelines) via the `#pragma unroll` directive. Figure 1(b) shows that after tuning, bandwidth is a consistent 30GB/s regardless of data type. However, unlike CPUs and GPUs where one sees performance saturate at the peak performance of the architecture, we see no saturation in FPGA performance. Rather, the tool set exhausts resources (abrupt end of trendlines) prior to saturating performance. This is because increasing the number of ERT operations increases the data pathway proportionally to the compute resources used. A kernel which passes data multiple times through the same pipeline before leaving the kernel may face pipeline width bottlenecks or data ingest/write bottlenecks depending upon the data flow. An ERT polynomial calculated by multiple passes through a smaller polynomial would see the Roofline plateau at the point of the smaller polynomial. Future work will investigate transformations that exploit graph similarity to maximize performance while reducing hardware utilization.

Figure 2 shows the benefit of manually unrolling to increase data parallelism and performance. Overall, unrolling by 32 improves bandwidth for intensities less than 16 by about 16× to about 30GB/s. However, for high intensity, unrolling can result in unsynthesizable code that fails to run. In this regime, one must reduce unrolling to successfully compile and saturate performance at about 930GFLOP/s.

We will use Roofline data throughout the paper to assess the results of our kernel benchmarking. For a given architecture, we may introspect its Roofline based on a kernel's arithmetic intensity. This informs us of how well the implementation of a kernel can make use of a target machine while the Roofline relative to pin bandwidth and the number of FMAs tells us how well an architecture can exploit the underlying technology.

Although the Arria 10's sustained bandwidth and peak performance is less than that of a CPU or GPU, it often requires far less power. Figure 3 plots the relationship

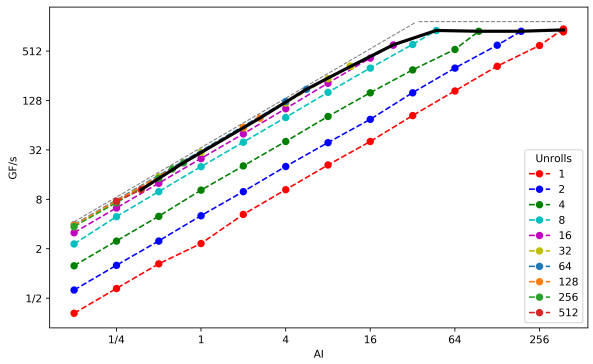


Figure 2: Arria 10 float32 ERT Rooflines as a function of unrolling (data parallelism). Synthesizable ERT polynomial degree is limited by available DSPs. Resource-constraint Roofline is shown in black; theoretical limits in grey.

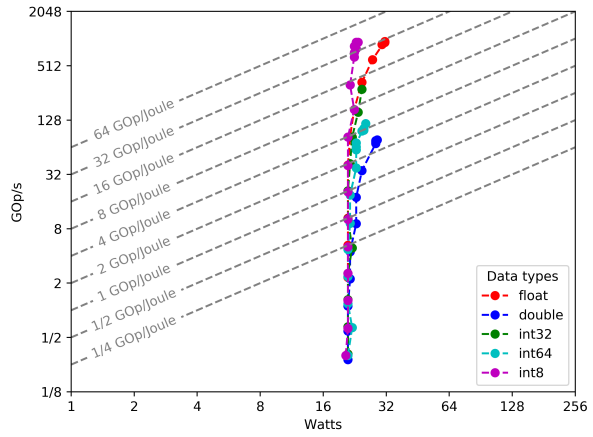


Figure 3: Arria 10 ERT energy efficiency trendlines for increasing arithmetic intensity (polynomial degree) for `float`, `double`, `int8`, `int32`, and `int64` data types. FPGAs exhibit a high idle power consumption relative to peak, requiring well-performing kernels for high efficiency.

between ERT performance and power as a function of data type (colored trendlines) and arithmetic intensity (points within a trendline). As one can see, regardless of data type, the Arria 10 consumes about 24W at low performance (but maximum bandwidth) and increases to a maximum of around 32W at maximum performance. As arithmetic intensity increases, performance increases rapidly, but power increases slowly. The result is that energy efficiency trends toward the operational energy efficiency asymptotes (diagonals). However, unlike a CPU or GPU, the FPGA clearly incentivizes energy efficient computation for `float` and `int8` data types while providing 8-16× lower energy efficiency for the `double` and `int64` data types. Moreover, the near vertical trendline implies the Arria 10 FPGA is incapable of power-proportional performance.

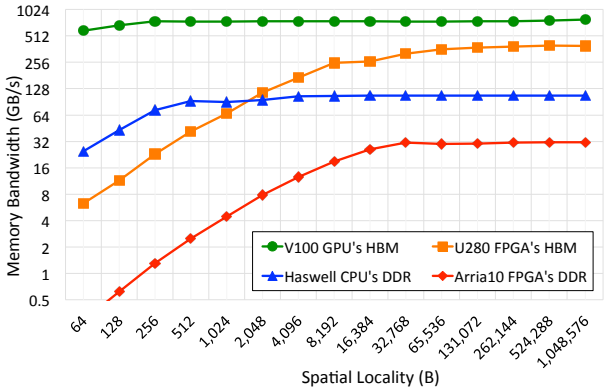


Figure 4: Realized Memory bandwidth vs. spatial locality on different architectures. FPGAs exhibit poor performance at low spatial locality relative to other architectures.

B. *Spatial Locality*

Whereas ERT attains maximum DRAM memory bandwidth with vectors approaching gigabytes in size, many applications do not exhibit such high spatial locality. Rather, many applications may access a few consecutive elements before jumping to another location in memory. Such memory access patterns can wreak havoc on stream prefetchers and demand superior approaches to latency hiding. To that end, we created a series of Stanza Triad-like [29] benchmarks for GPUs and FPGAs in order to understand how memory bandwidth scales with spatial locality.

Figure 4 shows sustained memory bandwidth when accessing data under varying degrees of spatial locality (the left proxies random[1] 64B access while the right proxies the STREAM benchmark). As expected, all architectures approach their ERT or STREAM bandwidths with high spatial locality. However, we observe very different properties for CPUs (stream prefetchers), GPUs (multithreading), and FPGAs (pipelining). CPU bandwidth degrades precipitously under 512B while GPU bandwidth only sees moderate degradation under 256B. Conversely, the DDR-based Arria 10 sees markedly reduced bandwidth under 32KiB while the HBM-based Xilinx U280 requires at least 64KiB of spatial locality to saturate bandwidth. In fact, for less than 2KiB of spatial locality, the DDR-based Haswell provides superior bandwidth. Clearly, depending on the lack of spatial locality in an application, the differences among architectures can be greatly exaggerated.

The parallel diagonal lines for Haswell, U280, and Arria 10 imply the bandwidth for all three architectures is well proxied by a simple α-β model of sequential access bandwidth time (time per byte) and "startup penalty" time (time for first byte). Asymptotically this approaches a band-

[1]Random access doesn't require a true random generator. Instead, it can be implemented with a dynamic order so that the memory controller and cache system cannot assume the address of future segments

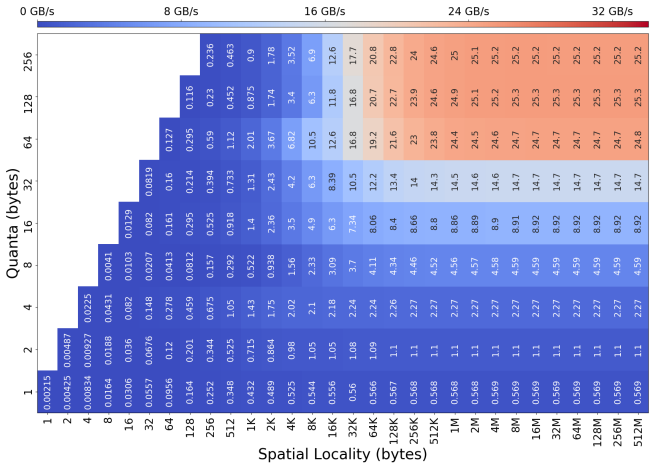


Figure 5: Arria 10 Memory Access Efficiency: Memory transfers with load-store unit (LSU) widths of **quanta** bytes. **Spatial locality** bytes of consecutive memory access. The two DRAM controllers are used in parallel to read and write from two separate memory buffers. Efficient DRAM access requires high spatial locality and coalesced data transfers.

width of $locality/\alpha$ for low spatial locality and $1/\beta$ for high spatial locality. Sharp transitions imply overlap (time being a maximum of these two terms) while a smooth transition implies serialization (a sum of these two terms). As this benchmark was run in parallel, bandwidth is combined and the startup penalty times (roughly 64B divided by bandwidth at 64B) should be scaled up by the number of workers.

Whereas CPUs and GPUs quantize DRAM access into 32B, 64B, or 128B transactions, FPGAs leave it to the programmer to orthogonalize memory access quanta and data type. Figure 5 shows how small memory access quanta can severely limit bandwidth. Although programmers accessing data structures containing 4B data might be motivated to use a 4B memory quanta, it is clear that doing so will degrade bandwidth by an order of magnitude. Rather, FPGAs should access memory using a 64B quanta (a reflection of the underlying DRAM technology), extract the relevant words, and cache/buffer the 64B quanta for subsequent reuse.

V. HPC Kernels

Now that we have distinguished the true performance and energy efficiency potential of the FPGAs from the theoretical performance and power limits, we may assess HPC kernel performance and efficiency in context. To that end, we examine three HPC kernels: dense matrix-matrix multiplication, sparse matrix-vector multiplication (SpMV), and Banded Smith-Waterman (BSW) [30, 31]. These kernels exhibit highly varied arithmetic intensity, degrees of parallelism, and requirements for fine-grained synchronization. We restrict ourselves to single precision for matrix multiplication (SGEMM) and SpMV to ensure the lack of hardened double precision functional units does not unduly skew our assessments of FPGA potential. Similar to many other Genomics codes, BSW employs 16-bit integer variables for computing alignment scores. In all cases, we presume data is resident on the FPGA accelerator thus obviating any PCIe data movement.

A. Dense Matrix-Matrix Multiplication

In the popular mindset, dense matrix-matrix multiplication (GEMM) represents the quintessential HPC numerical method. Although its performance is key to many applications in astrophysics, quantum chemistry, and machine learning, it is but one of a myriad of numerical kernels used in scientific computing. Nevertheless, GEMM performance and efficiency is a barometer for an architecture's potential. Here, we implement a single-precision SGEMM on the Arria 10 and U280 and compare performance and efficiency to that obtained on GPUs and CPUs.

All architectures make use of floating-point functional units (FPUs), either hardened or synthesized, to perform multiply and add operations in SGEMM. However, the data movement is fundamentally different between load/store architectures (i.e. CPUs and GPUs) and FPGAs. On FPGAs, FPUs are arranged in a 2D mesh of FPUs called a systolic array [32, 33]. Data move only between neighboring FPUs with a short distance, enabling the design to scale to many FPUs with modest frequency degradation. On CPUs and GPUs where the data paths and clock frequency are almost fixed, on-chip data reuse has a high impact on performance. We perform explicit register and cache (shared memory) blocking optimizations for GPU. On CPU, we rely on the Intel MKL library for single-core matrix multiplication. To perform SGEMM on 32 cores of a Cori-Haswell node, we use MPI to implement the SUMMA algorithm [26] which can hide communication overhead among processes.

Figure 6 plots SGEMM performance on CPUs, GPUs, and FPGAs as a function of hardware resources (number of FPUs). We ran with large matrix sizes (up to 32K×32K on CPU and GPU and 13K×13K on FPGAs). We vary hardware utilization on CPUs by controlling the number of MPI processes (thus the number of cores), on GPUs by controlling the number of thread blocks (thus number of SMs), and on FPGAs by synthesizing multiple designs that gradually increase the number of DSP blocks. The trend lines terminate when hardware is exhausted (CPUs/GPUs) or when synthesis fails (FPGAs). Straight lines in this figure denote perfect scaling. Most architectures scale very well with some loss in performance on HSW beyond 16 cores and on the U280 when using the full chip. Using this formalism allows us to define isocurves of "effective frequency" that denote the average frequency FPUs are clocked at ($\frac{GFLOP/s}{2*\#FPUs}$). Generally speaking, the architectures trend towards a frequency isocurve slightly lower than their nominal frequencies (nominal V100 frequency is 1.5GHz but

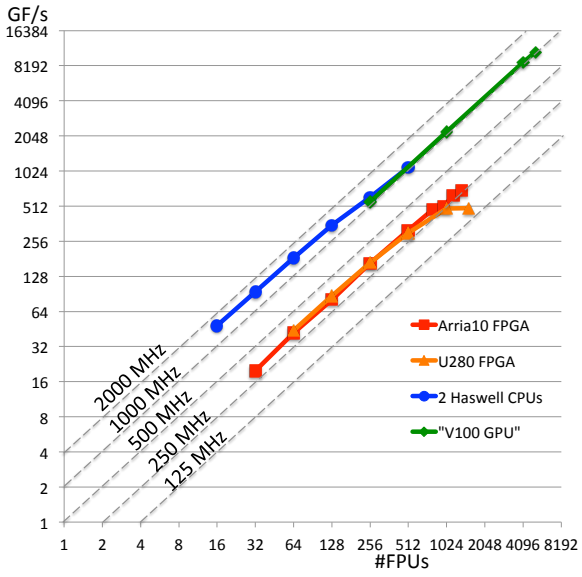


Figure 6: SGEMM performance scales linearly with hardware resources (FPUs). Effective frequency remains close to nominal frequency indicating a high percent of peak.

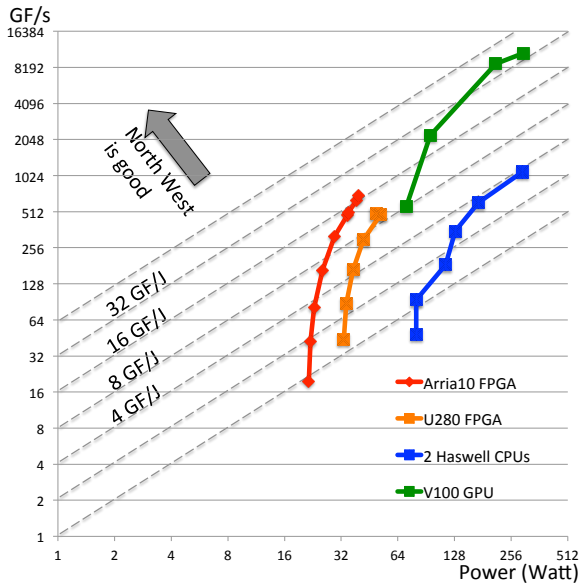


Figure 7: Although FPGA power is low and SGEMM energy efficiency approaching the 32 GFLOP/J ERT limit, GPUs still provide 2× higher energy efficiency.

effective frequency is 1.0GHz) indicating a high fraction of peak. However, the U280 clearly shows a substantial loss in frequency when using the full chip. This FPGA consists of 3 super logic regions and designs that span boundaries of these regions need to operate at a low clock frequency.

Volta's 15× speedup over FPGAs can be attributed to 5× more FPUs and 3× higher frequency. However, V100 requires far more power to do so. Figure 7 plots SGEMM performance as a function of power for increasing hardware resource utilization (concurrency). Although a power-proportional architecture would be bounded along an isocurve, we see all architectures show only modest increases in power for immense increases in concurrency and performance. Whereas the GPUs and CPUs require comparable power, the FPGAs see only moderate increases in power requiring less than one-seventh the power of a GPU. As before, we may draw isocurves of constant energy efficiency. The GPU's energy efficiency ultimately exceeds 32 GFLOP/J (approximately 31pJ/FLOP) while the Arria 10 and U280 FPGAs attain 16 and 8 GFLOP/J respectively.

We use ERT to evaluate the Arria 10 SGEMM designs shown in Figs. 6 and 7. Figure 8 plots SGEMM performance as a function of hardware resource represented by AI. The results shown in Figure 8 indicate that all SGEMM designs are very close to ERT performance limits given the same amount of hardware usage.

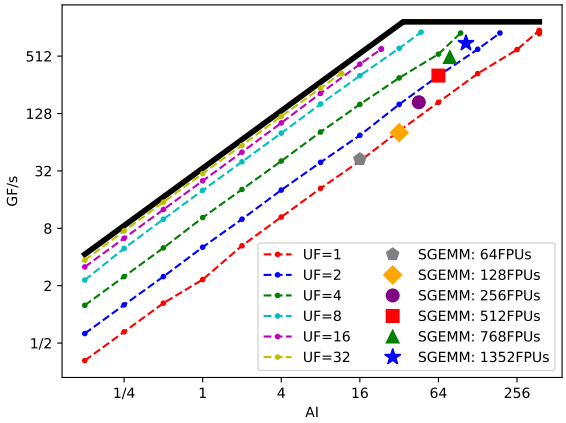


Figure 8: ERT trend lines establish performance upper bound as data traffic varies. Data points on these trend lines that have the same performance indicate the same hardware usage. SGEMM performance on Arria 10 is represented by large symbols, which are very close to the Roofline limit.

B. *Sparse Matrix-Vector Multiplication (SpMV)*

SpMV is an increasingly ubiquitous numerical kernel due to its broad applicability in many scientific, engineering, and graph analytical domains. Unlike SGEMM, SpMV has low arithmetic intensity (making it memory-bandwidth bound), can suffer from irregular loop lengths (making it hard to parallelize), and requires indirect memory access (necessitating exploitation of word-level temporal locality).

Rather than trying to understand all three aspects in conjunction, we build our FPGA SpMV up from two simpler kernels. First, we build a dot product kernel (*dotP*) that computes the sum of pairwise multiplications of two large

dense vectors (a core component of SpMV). Next, we split this computation into many smaller randomly sized dot products (*multiDot*) thus mimicking the intra-row dot products within SpMV. We exploit both coarse and fine grained parallelisms in all three kernels. For coarse-grained parallelism (partial sums of a dot product and independent rows in *multiDot* and SpMV), we use workers which are threads on CPUs, thread blocks on GPUs, and circuits on FPGAs. The fine-grained reduction sums in all three kernels are computed in a SIMD manner.

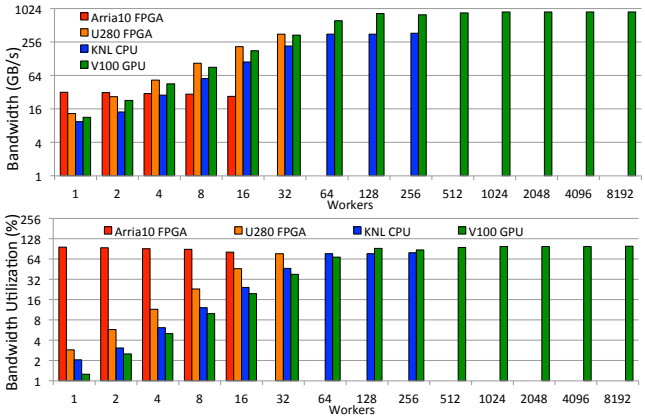


Figure 9: Bandwidth (top) and utilization (bottom) for a dot product as a function of the number of workers (concurrency). Observe superior GPU performance, but high utilization with low concurrency on the Arria 10.

Dot Product: The Arria 10 attains high bandwidth utilization needing only a single worker. Using multiple workers degrades performance due to uncoalesced memory access. On U280, one must synthesize FPUs which ultimately pull data from memory at a slower rate compared to that on Arria 10. To rectify this, we map two inputs to the same memory bank and 32 workers to saturate the 32 HBM banks. Although this implementation sustains 350GB/s (75% of ERT shown in Fig. 4), it is comparable to the bandwidth of the MCDRAM-based KNL. On a GPU, with 128 workers our implementation yields almost 900GB/s. Figure 9 compares dot product bandwidth and utilization as a function of the number of workers on FPGAs, KNL, and V100. The Arria 10 stands in stark contrast to the other architectures attaining high bandwidth utilization with few workers. Nevertheless, the KNL, V100, and U280 deliver far superior bandwidth.

multiDot: Executing multiple dot products concurrently challenges an architecture's ability to cope with a lack of spatial locality and high loop startup costs. Relative to Figure 9, Figure 10 shows that all architectures suffer in both bandwidth and utilization relative to the high spatial locality dot product with the FPGAs suffering more heavily on short vector lengths. Here we report the maximum

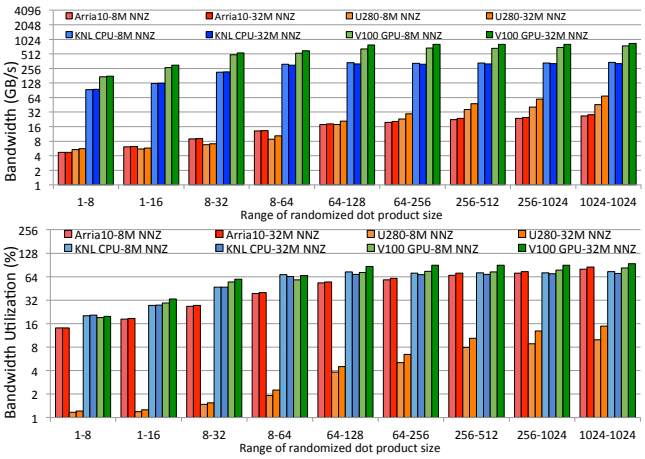


Figure 10: Memory bandwidth and utilization for multiple concurrent dot products of varying sizes. Observe the imense drop in bandwidth on the U280 relative to a dot product.

performance among all experiments, thus the result can be even lower, especially in the extremely low spatial locality cases. Increasing concurrency from 8M nonzeros to 32M nonzeros does not address this deficiency. Interestingly, the U280 is particularly sensitive to a lack of spatial locality with even 1024 element dot products too small to saturate an HBM bank. Such observations are well explained by the ERT spatial locality data in Section IV-B.

SpMV: As FPGAs lack a cache, we need to architect a solution to mitigate the random access associated with the source vector whilst preserving temporal in spatial locality. To that end, in our SpMV implementation, we create a replica of the vector in a worker-private BRAM. As BRAM capacity is fixed, this imposes a limit on the number of workers and only four HBM banks on the U280.

Figure 11 shows SpMV realized bandwidth measured by $(8N + 8\text{NNZ})/\text{runtime}$, where N is number of rows and

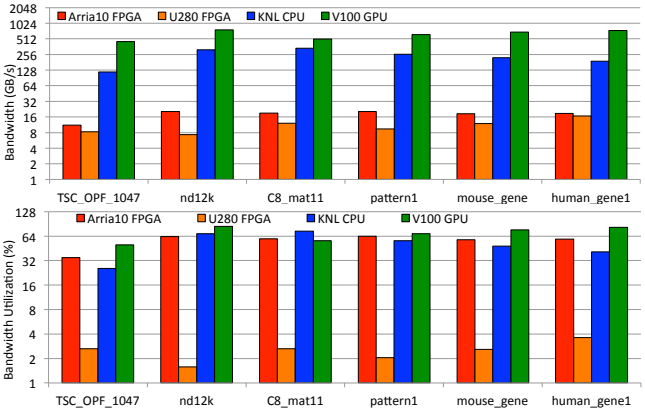


Figure 11: SpMV bandwidth and utilization for different matrices. NNZ/row increases from 250(left) to 1,107(right).

columns and NNZ is number of nonzeros, and utilization (% of ERT) for each architecture for a variety of matrices. Observe that SpMV bandwidth is well-correlated with *multi-Dot* bandwidth with KNL, V100, and Arria 10 sustaining 32-60% of ERT bandwidth and the U280 delivering about 2-4%. Clearly, the lack of even a last-level cache has necessitated hefty design requirements while the inability of the compiler to synthesize a design resilient against low spatial locality can dramatically limit SpMV performance.

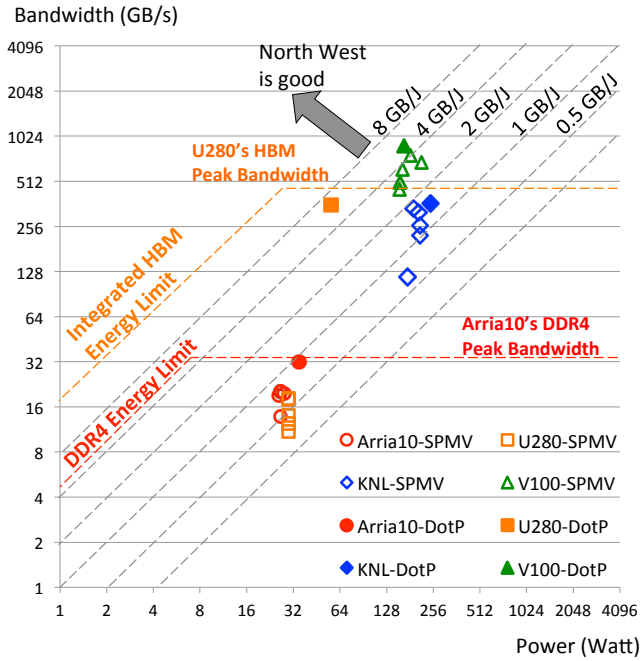


Figure 12: Dot product (closed symbol) and SpMV (open symbol per matrix) bandwidth and power. Observe isocurves of constant energy efficiency as well as HBM (orange) and DDR (red) technology's limits on energy efficiency.

Figure 12 plots dot product (closed symbol) and SpMV (open symbols for different matrices) bandwidth as a function of power. We also show isocurves of constant energy efficiency to highlight the energy efficiency of each architecture. The ultimate limits on HBM (orange) and DDR (red) energy efficiency [34, 35] and bandwidth are also marked.

For the streaming dot product, the U280 FPGA is 1.2× and 4× more energy efficient than the V100 GPU and the KNL CPU whilst attaining near peak bandwidth. Moreover, as HBM provides a lower limit of 7pJ/bit or 17.8GB/s/J, we can see the U280 delivers energy efficiency within about 2× of the technological limit. Conversely, the Arria 10 delivers far less bandwidth for comparable power and is thus nearly an order of magnitude less energy efficient.

When it comes to SpMV (open symbols), we generally see bandwidth and energy efficiency similar to that of the dot product. However, due to its poor SpMV performance, the U280 delivers the lowest energy efficiency allowing the GPU to claim the top spot. Broadly speaking, the GPU is about twice as energy efficient as KNL (due mostly to bandwidth) and KNL twice as efficient as the Arria 10 (more bandwidth but more power). As GPU SpMV energy efficiency is about one third of the HBM technology limit, we surmise that GPU efficiency for bandwidth-limited codes can grow by no more than 3× without first improving HBM efficiency.

C. Banded Smith-Waterman (BSW)

Metagenomic analysis is an important area of bioinformatics within DOE requiring HPC resources. merAligner [36] is a parallel sequence aligner based on the seed-and-extend algorithm. Although it builds a number of distributed data structures, it uses the Batched Smith-Waterman (BSW) algorithm to execute multiple local alignments [30, 31].

In this paper, we modify Muaaz Awan's *Batched Smith-Waterman* code [30] to map the application parallelisms on the Arria 10 and U280 FPGAs. At the high level, the local alignment simultaneously performs many alignment operations, one for each pair of the input query-target sequences. Implementations on hardware architectures differ in the way we implement the alignment operation and how we fit many concurrent alignment operations on the hardware. Specifically, the CPU implementation SIMDizes the computations on the same anti-diagonal of the Smith-Waterman score table, where there is no data dependency. Independent alignment operations are mapped to different processor cores using OpenMP. On GPUs, operations on an antidiagonal are executed by threads in a block while multiple thread blocks perform different alignments. On FPGAs, each alignment is represented as a deep pipelined dataflow graph that is replicated to perform multiple alignments concurrently.

Figure 13 shows BSW strong-scaling performance as we increase hardware resources (CPU cores, GPU SMs, FPGA LUTs). Thanks to a communication avoiding optimization that reduces the demands on memory bandwidth, all implementations scale to the entire chip. As a result, CPU and GPU performance scale linearly with increased hardware with only slight degradation when using the second Haswell CPU. Unlike an instruction processor, FPGA's synthesis tools reserve 10-25% of the LUTs. Another 25-30% is needed for routing. This means that our FPGA designs are fairly area efficient, since with only half of the available LUTs we can synthesize many alignment circuits (e.g. 72 circuit replications on Arria 10 compared to 80 SMs on V100). Owing to their high frequency and concurrency, the GPUs and CPUs still provide the highest throughput.

Figure 14 plots BSW time-to-solution as a function of power as one increases concurrency. Unlike prior HPC kernels, most architectures see only slight increases in power within a socket with the Arria 10 consuming about 30W. However, whereas GPU throughput increases by roughly 80×, its power increases by about 4×. As a result, the

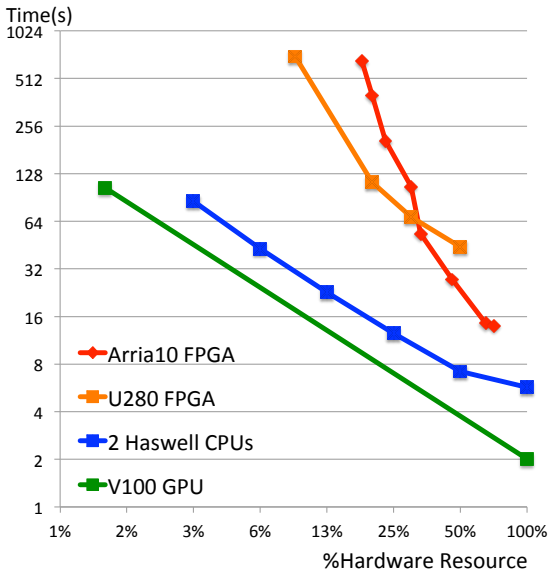


Figure 13: BSW Strong scaling performance as a function of hardware resources (FPGA LUTs, CPU Cores, GPU SMs)

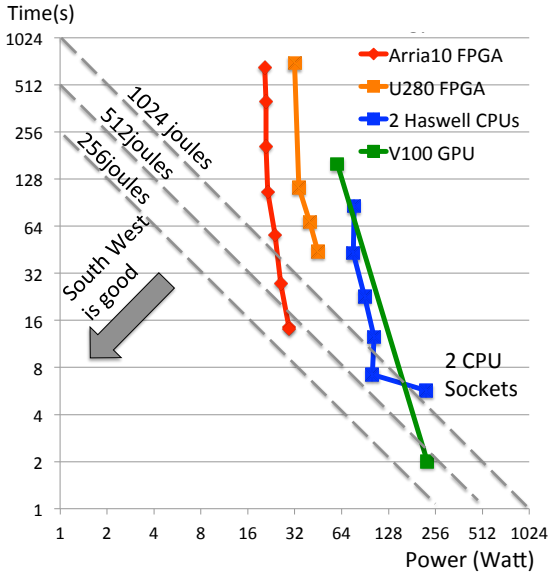


Figure 14: BSW performance, power, and energy efficiency as a function of hardware resources (each dot)

Arria 10 ultimately requires 10% and 40% less energy-to-solution than a GPU or single socket CPU.

VI. Conclusions and Future Work

Recent years have seen FPGAs integrate hardened FPUs and HBM in order to maximize performance and energy efficiency. In this paper, we first extended the empirical Roofline Toolkit to benchmark FPGA performance and bandwidth. We show that FPGAs can exploit this raw bandwidth and compute potential, but performance can be extremely brittle. Subsequently, we evaluated FPGA performance and efficiency on HPC kernels. We show that although single-precision FPGA performance and bandwidth still falls far below GPUs for compute and memory-intensive tasks, the energy efficiency of FPGAs with hardened DSPs is now within a factor of two for SGEMM and SpMV, and in the case of genomics, can exceed that of GPUs by 10%.

Against conventional wisdom, we do not believe FPGA architects should prioritize the integration of hardened 64b functional units. The Arria 10 already includes hardened 32b FPUs, yet GPU performance and efficiency on arithmetically-intensive computations was superior. One expects this to hold in double-precision as well. Although 64b *functionality* is essential, FPGA vendors should strive for a machine balance of at least one 64b FLOP per byte.

Ultimately, vendors should prioritize productivity making it easier for programmers to maximize memory bandwidth, minimize data movement, and automatically balance pipelining, unrolling, and data parallelism whilst dramatically reducing compilation time. Our FPGA implementations required orders of magnitude more software development time than the equivalent (and often superior) CPU and GPU implementations. Programmers had to manually tune implementations to balance hardware utilization against memory bandwidth and create crude scratch pads to make up for the lack of caching paradigms that have been included in CPUs and GPUs for decades. FPGA software should either automatically synthesize structures to exploit spatial and temporal locality or FPGA architects should include memory interfaces and last-level memory-side caches that obviate the need for programmers to micromanage channel parallelism, and detect and exploit spatial and temporal locality.

FPGAs will likely to continue to shine in the areas GPUs and CPUs are poorly optimized for — memory-intensive streaming computations with patterns of spatial and temporal locality known at compile time, pipelined computations devoid of massive fine-grained data parallelism, operations on short integer and user-defined data types, and possibly latency-sensitive, network-intensive computations.

Acknowledgements

This research was supported by the Advanced Scientific Computing Research Program in the U.S. Department of Energy, Office of Science, under Award Number DE-AC02-05CH11231, and used resources of the National Energy Research Scientific Computing Center (NERSC) which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. We thank Muaaz Awan for access to his Smith-Waterman code and Farzad Fatollahi-Fard for administration of our testbed.

REFERENCES

[1] NERSC, "National Energy Research Scientific Computing Center," Accessed September 8th, 2020. [Online]. Available: https://www.nersc.gov

[2] ——, "NERSC-10 Workload Analysis," Accessed September 8th, 2020. [Online]. Available: https://portal.nersc.gov/project/m888/nersc10/workload/N10_Workload_Analysis.latest.pdf

[3] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures." [Online]. Available: https://doi.org/10.1145/1498765.1498785

[4] B. Van Essen, C. Macaraeg, M. Gokhale, and R. Prenger, "Accelerating a random forest classifier: Multi-core, gp-gpu, or fpga?" in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, 2012, pp. 232–239.

[5] E. Nurvitadhi, D. Sheffield, Jaewoong Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 77–84.

[6] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 5–14. [Online]. Available: https://doi.org/10.1145/3020078.3021740

[7] Y. Zhang, Y. H. Shalabi, R. Jain, K. K. Nagar, and J. D. Bakos, "Fpga vs. gpu for sparse matrix vector multiply," in *2009 International Conference on Field-Programmable Technology*, 2009, pp. 255–262.

[8] S. Kestur, J. D. Davis, and O. Williams, "Blas comparison on fpga, cpu and gpu," in *2010 IEEE Computer Society Annual Symposium on VLSI*, 2010, pp. 288–293.

[9] S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance comparison of fpga, gpu and cpu in image processing," in *2009 International Conference on Field Programmable Logic and Applications*, 2009, pp. 126–131.

[10] M. Birk, M. Zapf, M. Balzer, N. Ruiter, and J. Becker, "A comprehensive comparison of gpu- and fpga-based acceleration of reflection image reconstruction for 3d ultrasound computer tomography," *J. Real-Time Image Process.*, vol. 9, no. 1, p. 159–170, Mar. 2014. [Online]. Available: https://doi.org/10.1007/s11554-012-0267-4

[11] S. Che, J. Li, J. W. Sheaffer, K. Skadron, and J. Lach, "Accelerating compute-intensive applications with gpus and fpgas," in *2008 Symposium on Application Specific Processors*, 2008, pp. 101–107.

[12] M. Véstias and H. Neto, "Trends of cpu, gpu and fpga for high-performance computing," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–6.

[13] K. Krommydas, W. Feng, M. Owaida, C. D. Antonopoulos, and N. Bellas, "On the characterization of opencl dwarfs on fixed and reconfigurable platforms," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, 2014, pp. 153–160.

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

[15] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, "Understanding performance differences of fpgas and gpus," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, pp. 93–96.

[16] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, ser. IISWC '09. USA: IEEE Computer Society, 2009, p. 44–54. [Online]. Available: https://doi.org/10.1109/IISWC.2009.5306797

[17] M. Meyer, T. Kenter, and C. Plessl, "Evaluating FPGA accelerator performance with a parameterized opencl adaptation of the HPCChallenge benchmark suite," *arXiv preprint arXiv:2004.11059*, 2020.

[18] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking high bandwidth memory on fpgas," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 111–119.

[19] M. Siracusa, M. Rabozzi, E. Del Sozzo, L. Di Tucci, S. Williams, and M. D. Santambrogio, "A cad-based methodology to optimize hls code via the roofline model," in *2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

[20] J. Cardoso, B. da Silva, A. Braeken, E. H. D'Hollander, and A. Touhafi, "Performance modeling for fpgas: Extending the roofline model with high-level synthesis tools," *International Journal of Reconfigurable Computing*, vol. 2013, p. 428078, 2013. [Online]. Available: https://doi.org/10.1155/2013/428078

[21] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *2013 IEEE 27th International Symposium on Parallel and Distributed*

Processing, 2013, pp. 661–672.

[22] A. Ilic, F. Pratas, and L. Sousa, "Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 52–58, 2017.

[23] "Intel Arria 10 Device Overview," accessed: 2020-09-12. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf

[24] "Alveo U280 Data Center Accelerator Card," accessed: 2020-09-12. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/alveo/u280.html

[25] NERSC, "Cori Supercomputer," Accessed September 8th, 2020. [Online]. Available: https://www.nersc.gov/systems/cori/

[26] R. A. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," USA, Tech. Rep., 1995.

[27] "Empirical Roofline Toolkit (ERT)," accessed: 2020-08-01. [Online]. Available: https://bitbucket.org/berkeleylab/cs-roofline-toolkit/src/master/

[28] C. Yang, R. Gayatri, T. Kurth, P. Basu, Z. Ronaghi, A. Adetokunbo, B. Friesen, B. Cook, D. Doerfler, L. Oliker, J. Deslippe, and S. Williams, "An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability," in *P3HPC Workshop at SC*, 2018.

[29] S. Kamil, P. Husbands, L. Oliker, J. Shalf, and K. Yelick, "Impact of modern memory subsystems on cache optimizations for stencil computations," in *Workshop on Memory System Performance (MSP)*, 2005.

[30] K. A. Yelick, L. Oliker, and M. G. Awan, "Gpu accelerated smith-waterman for performing batch alignments (gpu-bsw) v1.0," [Computer Software] https://doi.org/10.11578/dc.20191223.1, nov 2019. [Online]. Available: https://doi.org/10.11578/dc.20191223.1

[31] M. Farrar, "Striped Smith–Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 11 2006. [Online]. Available: https://doi.org/10.1093/bioinformatics/btl582

[32] S. Y. Kung, *VLSI Array Processors*. USA: Prentice-Hall, Inc., 1987.

[33] J. de Fine Licht, G. Kwasniewski, and T. Hoefler, "Flexible communication avoiding matrix multiplication on fpga with high-level synthesis." *CoRR*, vol. abs/1912.06526, 2019. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1912.html#abs-1912-06526

[34] "Virtex UltraScale+ HBM FPGA: A Revolutionary Increase in Memory Performance." [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp485-hbm.pdf

[35] K. Bergman, G. Hendry, P. Hargrove, J. Shalf, B. Jacob, K. Hemmert, A. Rodrigues, and D. Resnick, "Let there be light!: The future of memory systems is photonics and 3d stacking," 01 2011.

[36] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar, and K. A. Yelick, "meraligner: A fully parallel sequence aligner," in *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*. IEEE Computer Society, 2015, pp. 561–570. [Online]. Available: https://doi.org/10.1109/IPDPS.2015.96