FPGA-based HPC Accelerators: an Evaluation on Performance and Energy Efficiency

Tan Nguyen[1] | Colin MacLean[2] | Marco Siracusa[3] | Douglas Doerfler[2] | Nicholas J. Wright[2] | Samuel Williams[1]

[1]Computational Research Division, Lawrence Berkeley National Laboratory, California, USA
[2]National Energy Research Scientific Computing, Lawrence Berkeley National Laboratory, California, USA
[3]DEIB, Politecnico di Milano, Milan, Italy

Correspondence
Tan Nguyen, Email: TanNguyen@lbl.gov

Abstract

Hardware specialization is a promising direction for the future of digital com-puting. Reconfigurable technologies enable hardware specialization with modest non-recurring engineering cost, but their performance and energy efficiency com-pared to state-of-the-art processor architectures remain an open question. 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 per-formance, 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 Empir-ical Roofline Toolkit (ERT) to FPGAs in order to assess our results in terms of the Roofline Model. We show design optimization and tuning techniques for peak FPGA performance at reasonable hardware usage and power consumption. As FPGA peak 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 against microbench-mark 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.

1 | 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. 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 relative to CPU-only solutions. In order to tap into the full potential of a GPU, programmers must modify their applications to exploit a hybrid programming model using different models for distributed memory parallelism and on-node 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 for some application classes and scaling regimes.

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 price sensitivity cannot be ignored.

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 GPUs 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 an FPGA by sending them through a circuit of operations. 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 article, an expansion of our previous work[3], 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 FPGAs. We commence with a study of the peak performance and efficiency as a function of data reuse using the well-established Roofline Model[4] to frame the conversation. In this study, we also learn how the on-chip network and DRAM respond to memory access patterns with varying degrees of spatial data locality. We then proceed by examining three HPC kernels (GEMM, SpMV, and Smith-Waterman) spanning a range of compute intensity, parallelism, synchronization requirements, and underlying data types. 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. Relative to our previous work, we have expanded our Empirical Roofline Toolkit (ERT) analysis to include both direct and indirect addressing of BRAM, expanded the HPC kernel suite to include 64b DGEMM, created performance models for each kernel to facilitate analysis, and conducted a deep dive on design space exploration (DSE) for SGEMM and DGEMM.

2 | RELATED WORK

Over the last decade, several publications have investigated the benefits of using FPGA devices in specific domains such as Machine Learning[5,6,7], Linear Algebra[8,9] and Image Processing[10,11]. However, the resulting considerations are not general enough to provide a thorough FPGA characterization. In this direction, other works[12,13,14,15,16] analyzed FPGA performance by accelerating common benchmark suites and comparing the obtained results with other architectures. In particular, Cong et al.[16] proposed an FPGA-GPU comparison on the Rodina[17] 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[18] and microbenchmarking[19] this device. However, despite the appealing results achieved by the authors, these analyses have not been directly compared against other accelerators.

In our earlier work[3], we considered 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. That comparison was 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. This paper extends the kernel set to understand the on-chip network in both regular and adverse memory access patterns. The paper also extends the comparison on double precision, which is extremely important for HPC codes. As in Cong et al.[16], we discuss our considerations through a performance model (i.e. Roofline). Although the literature already proposes some FPGA Roofline model formulations[20,21], 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[22,23], a fundamental aspect for FPGA devices. FER[24] is an ERT-inspired tool, using microbenchmarks to evaluate the performance and energy efficiency on FPGAs. We use more complex HPC kernels. In this paper, we also present an effective approach to tuning hardware by combining application knowledge with design space exploration. The result is an optimal design at reasonable hardware usage and power/energy consumption.

One of the limitations of simple energy-efficiency studies is that the reader is left trying to balance performance against energy. Ultimately these both translate into cost which, with sufficient parameters, the Energy Delay Sum and Energy Delay Distance[25] provide actionable guidance to programmers and procurement teams. As the subtleties of DOE procurement pricing, bulk energy pricing, power infrastructure upgrades, and costs associated with supporting multiple architectures are beyond the scope of this paper, we will focus on performance and energy and leave it to the reader to weight and combine these terms to suit their needs.

3 | EXPERIMENTAL SETUP

3.1 | Evaluated Architectures

In this paper, we evaluate the performance and energy potential of two FPGAs — the Intel Arria 10 GX 1150[26] and the Xilinx Alveo U280[27]. The relevant features of the two architectures are summarized in Table 1. 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) that 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[28]. 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 lightweight CPU cores with MCDRAM (HBM-like) memory technology.

The NVIDIA V100 (Volta) GPU 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 of 512 FMAs). Collectively, the node's eight 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

TABLE 1 Hardware specifications of the architectures we examined

	Intel Arria 10 GX1150	Xilinx Alveo U280	Intel Haswell (2P×16c)	Intel KNL (68c)	NVIDIA V100
Peak Frequency	0.45GHz	0.45GHz	2.3GHz	1.4GHz	1.53GHz
Logic Blocks	1150K LEs	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$ 256KiB L2$/core 32KiB L1$/core	1MiB L2$/2 cores 32KiB L1$/core	6 MiB L2$ 96KiB L1$/SM
Registers	213KiB	326KiB	7KiB/core	8KiB/core	256KiB/SM
DDR Pin Bandwidth	34GB/s	38GB/s	136GB/s	-	-
HBM Pin Bandwidth	-	460GB/s	-	460GB/s	900GB/s

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.

3.2 | Programming Models

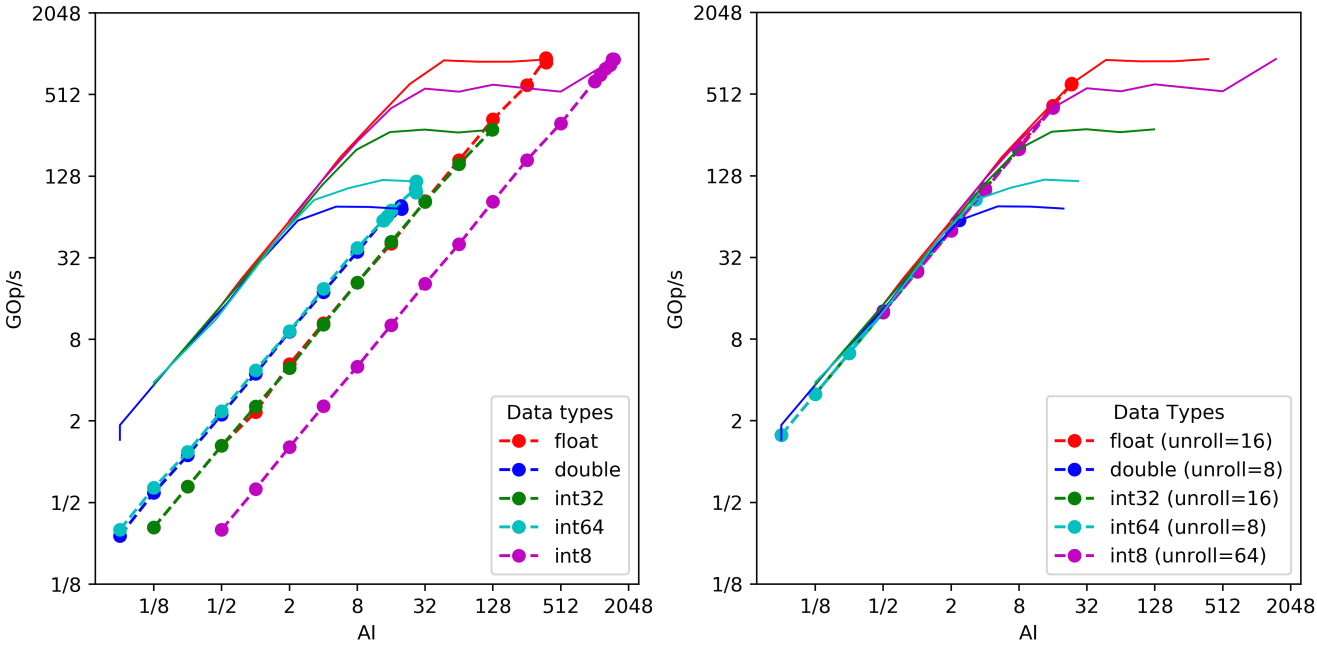
As this paper is focused on the potential performance and energy efficiency gains FPGAs might provide over CPUs or GPUs, we are willing to sacrifice some degree of productivity and portability in order to maximize performance. Although writing codes in RTL might maximize FPGA performance, such low-level 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 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 the dual-Haswell node. This choice comes from the underlying SUMMA algorithm[29] which can hide communication cost well, but was presented with only an MPI implementation.

3.3 | 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 the Xilinx Board Utility, respectively. Similarly, on V100 GPU we used the power usage reported by the Nvidia Management Library (NVML). On Haswell and KNL, we chose LIKWID to collect CPU power consumption at the socket level. 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.

The recently, we have observed two trends in accelerated computing. First, the power of a GPU has steadily increased with V100 sustaining around 300W. Conversely, CPU power has remained relatively low ranging from roughly 40W(idle) to 150W(active). Second, to maximize energy efficiency, the ratio of accelerators to CPUs has steadily increased from 1:1 (Titan[30]) to 3:1 (Summit[31]) to 4:1 (Perlmutter[32]) to 8:1 (NVIDIA A100 HGX[33]). As a result, aggregate active GPU power can now constitute up to 95% of node power[1]. Although FPGA active power is often less than half an active CPU's power, the inclusion of up to eight FPGAs on a node ensures FPGA power dominates node power. Thus, examining accelerator energy efficiency is a good proxy for accelerated node energy efficiency.

[1]The growing GPU:CPU ratio ensures the bulk of the performance is in the GPUs and obviates any need to partition a kernel between CPU and GPU to maximize performance



(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. We observe sub-optimal bandwidth with single element request on small data types. Conversely, after tuning (b), Roofline bandwidth on the Arria 10 mimics the traditional Roofline. The tuning process adjusts data parallelism (unrolling factor) so that all memory pins are occupied and the frequency gap is filled (350MHz chip design vs. 2130MHz DDR). Resource-limited theoretical performance is shown in solid lines.

4 | 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 or an instruction processor can exploit reuse through a hardware cache. The latter informs us of how well the FPGA software stack can hide latency and maximize utilization of the memory subsystem under different memory access patterns.

4.1 | Temporal Locality (Roofline)

The Roofline model visualizes bottlenecks by plotting architectural performance bounds and applications characteristics in a performance-data locality 2D plane[4]. The performance (upper) bound is defined as a curve in the plane where 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)[34,35] 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 quanta for memory access.

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

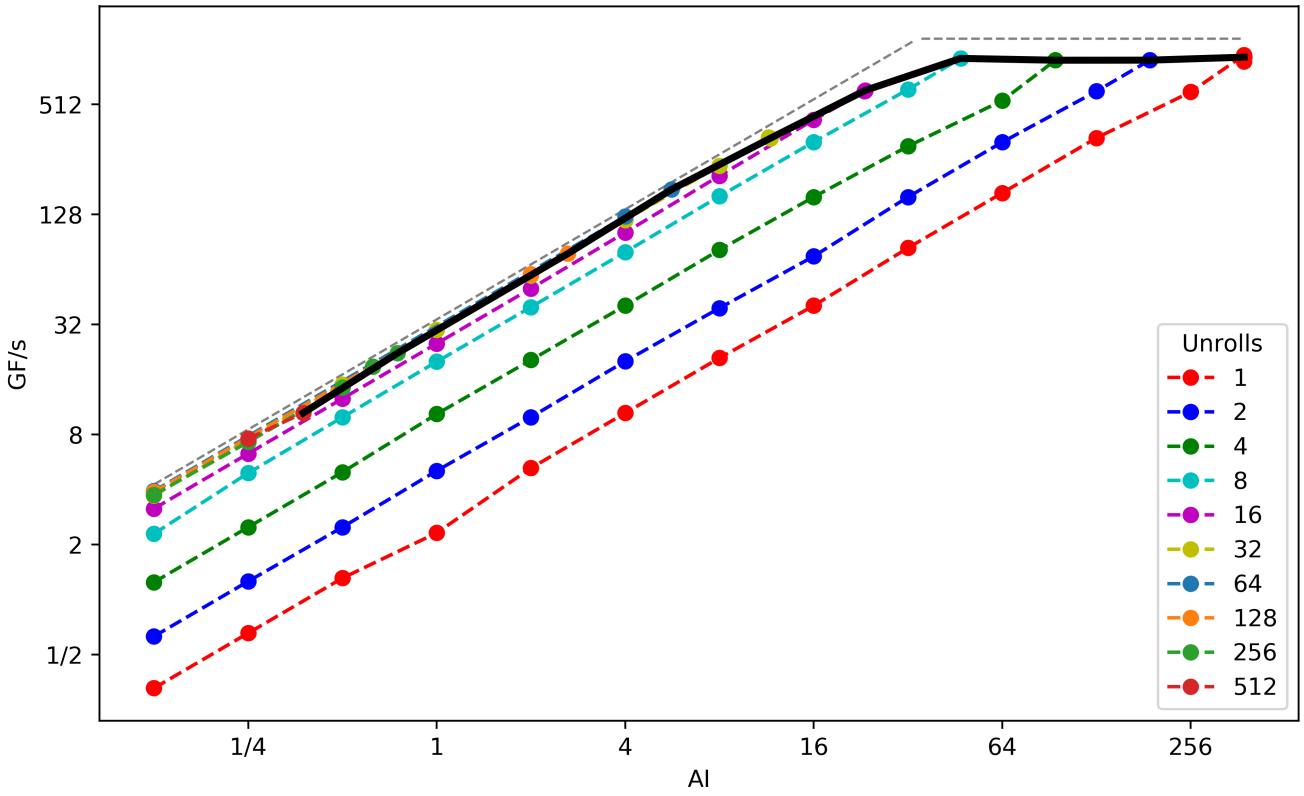


FIGURE 2 Arria 10 float32 ERT Rooflines as a function of unrolling (data parallelism). Synthesizable ERT polynomial degree is limited by available DSPs. Resource-constraint Roofline is shown in black; theoretical limits in grey.

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 an efficient load store unit.

Figure 1(a) highlights that unlike CPUs and GPUs, FPGA bandwidth without tuning (dotted lines) is data type dependent and can vary from 0.6GB/s (`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-derived theoretical Roofline ceilings (solid lines) based on available FPGA resources and arithmetic intensity. Clearly, empirical performance is well below these theoretical ceilings.

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 improve the attainable frequency. This is an advanced built-in keyword to suggest the compiler insert a register to replicate a signal. This register serves as an intermediate buffer to reduce the distance that a signal has to travel, thereby increasing the frequency. 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, but performance does not saturate at the resource-limited theoretical bound (solid lines) before becoming unsynthesizable.

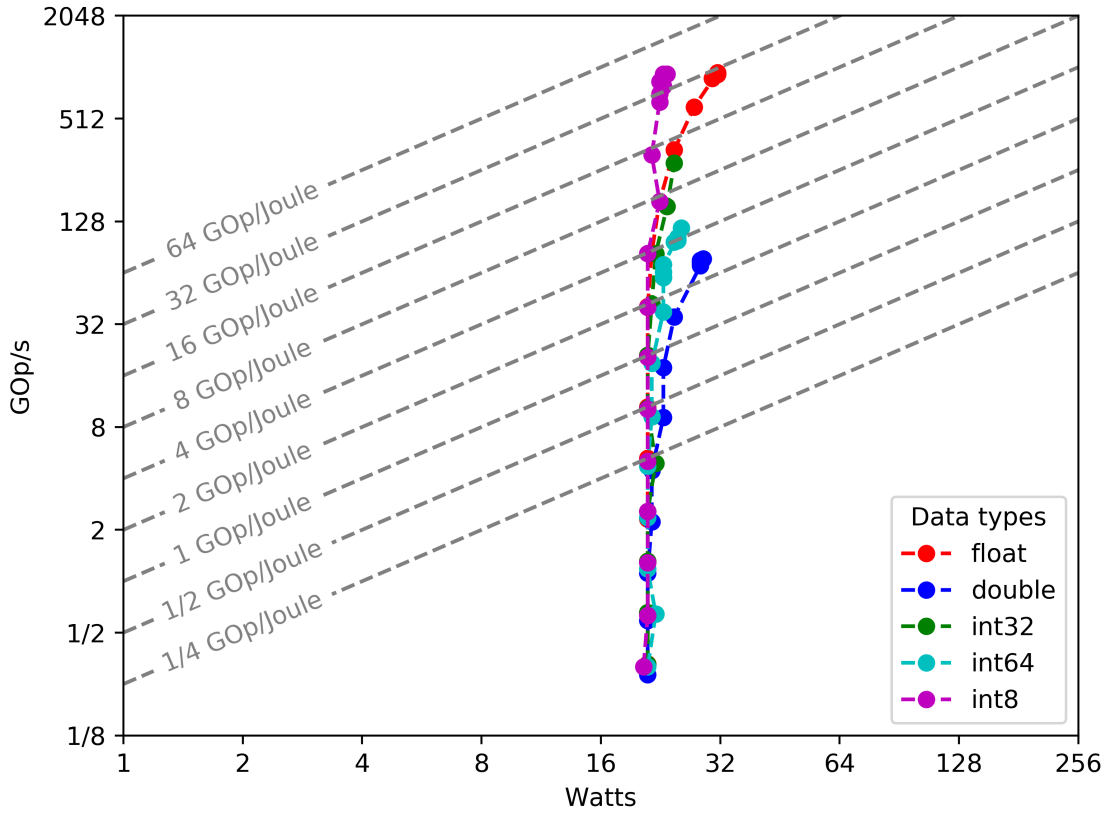


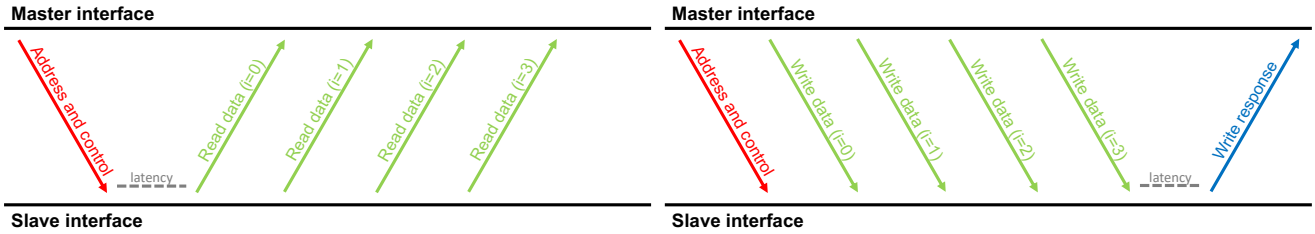
FIGURE 3 Arria 10 ERT energy efficiency trendlines for increasing polynomial degree for various data types. FPGAs exhibit a high idle power consumption relative to peak, requiring well-performing kernels for high efficiency.

Figure 2 highlights the impact of manually unrolling (data parallelism) on 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 attain a peak performance of about 930GFLOP/s.

One can conceptualize utilization of FPGA resources as the product of two terms. First, pipeline depth (the number of ERT operations) can be thought of as one dimension, while data parallelism (unrolling factor) can be thought of as the other. As one increases the ERT polynomial, one increases pipeline depth and as a result, parallelism is reduced. Unfortunately, there is a maximum polynomial (all resources are dedicated to the pipleine and none to parallelism) beyond which code becomes unsynthesizable. 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.

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 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 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.

We will use Roofline and energy efficiency data throughout the rest of the paper to assess the results of our kernel benchmarking. We can compare the Empirical Roofline bound to an architecture's pin bandwidth, number of FMAs, and peak frequency in order to understand how well an architecture can exploit the underlying process technology. Similarly, we can compare an



(a) Read burst mode **(b)** Write burst mode

FIGURE 4 AXI burst transfer for read and write operations.

architecture's Roofline bound given kernel's arithmetic intensity and performance in order to quantify how well the hardware and software can exploit the capabilities of the underlying architecture.

4.2 | Spatial Locality and DRAM Bandwidth

Whereas ERT attains maximum DRAM memory bandwidth with vectors approaching gigabytes in size, many applications do not exhibit such high spatial locality. Here, we define the spatial locality as the number of bytes consecutively accessed by each outer iteration of the benchmark before jumping to a new location in memory. Modern CPUs employ hardware prefetchers to preemptively load elements from long predictable access patterns into multiple cache levels thus hiding the per-word or per-cache line memory latency. Conversely, FPGAs incorporate neither caches nor prefetchers, and all memory access is programmer-directed toward either on-chip SRAM/registers distributed across the fabric or off-chip DRAM. in order to quantify how memory bandwidth scales with increasing spatial locality, we created a series of Stanza Triad-like[36] benchmarks for GPUs and FPGAs. Generally speaking, due to the longer latency and lack of configurability, off-chip DRAM can be highly sensitive to a lack of spatial locality. As such, we focus our FPGA analysis on the interplay between spatial locality and DRAM bandwidth.

We measured that the round-trip load-to-use latency to access an HBM bank on the Alveo U280 is about 200ns — corresponding to 100 clock cycles at 450 MHz. Whereas naive implementations might expose this latency on every transfer, for linear predictable access patterns, these transfers can be performed exploiting the AXI protocol's burst mode[37]. Figure 4 graphically highlights how the AXI transfer protocol allows the FPGA to transfer long streams of data requiring a single handshake at the beginning of the stream. However, as spatial locality decreases, the more the cost of this handshake impinges on performance.

Figure 5 shows sustained memory bandwidth when accessing data under varying degrees of spatial locality. With low spatial locality, there are many requests for short chunks of data starting at various addresses. With high spatial locality, there is more sequential access that can be exploited by the memory subsystem. As a result, low spatial locality proxies random[2] 64B access while high spatial locality proxies the STREAM benchmark. 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 bandwidth 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.

As expected, all architectures approach their ERT or STREAM bandwidths ($1/\beta$) with high spatial locality. However, we observe very different properties for CPUs (stream prefetchers), GPUs (multithreading), and FPGAs (pipelining) as spatial locality decreases. 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 to the HBM-based Xilinx. Clearly, some architectural paradigms are far more sensitive to the lack of spatial locality in an application than the underlying memory technology would suggest.

[2]Random access does not 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.

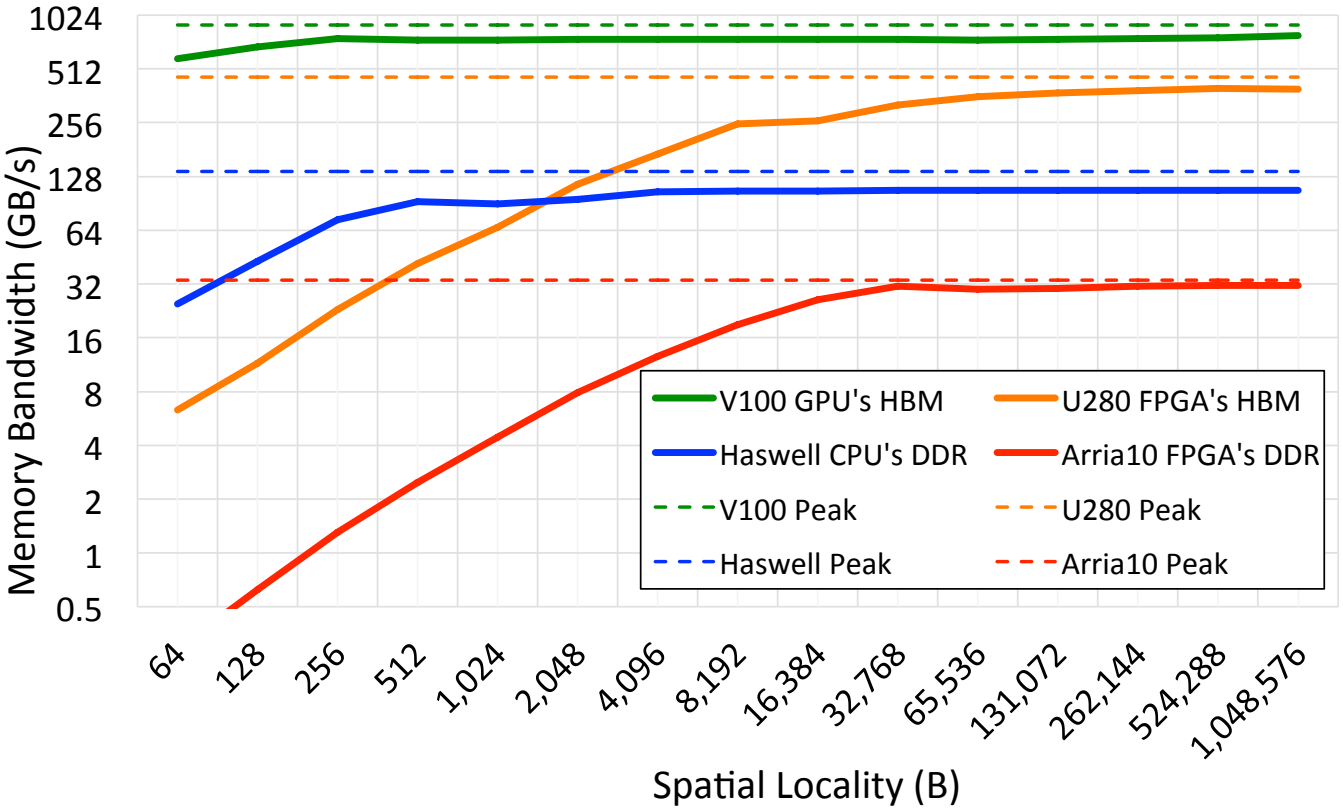


FIGURE 5 Realized Memory bandwidth vs. spatial locality on different architectures. FPGAs exhibit poor performance at low spatial locality relative to other architectures.

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 6 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 manually cache/buffer the 64B quanta for subsequent reuse.

4.3 | On-chip Memory Bandwidth

HPC applications exhibit a variety of memory access patterns. On-chip memory resources (BRAM and registers) can be configured in order to regularize off-chip memory access and exploit the spatial and temporal locality characteristics unique to each benchmark. To quantify the potential benefit from specialization, we study BRAM performance as a function of data type, concurrency, and memory access pattern (unit-stride and indirect).

Figure 7 presents the aggregate BRAM bandwidth for a simple unit-stride AXPY pattern ($z[:] = \alpha \times x[:] + y[:]$) on a single super logic region of a U280 FPGA [3]. To help the compiler achieve high frequency, each processing element is assigned with a constant number of data elements and we set it to 1 in this study. For smaller designs, it is relatively easy to attain 450MHz. However, as one increases concurrency, the complexity of mapping BRAM memory locations to FPUs increases. As a result, at high concurrency, effective frequency drops to 245MHz in double precision. Thus, unlike CPUs and GPUs where performance scales with concurrency until one saturates cache bandwidth, as one increases concurrency on FPGAs, BRAM bandwidth scales (sublinearly) without saturation.

[3]There are three of these regions on the card but using all three effectively requires excessive tuning which is beyond the scope of this performance study.

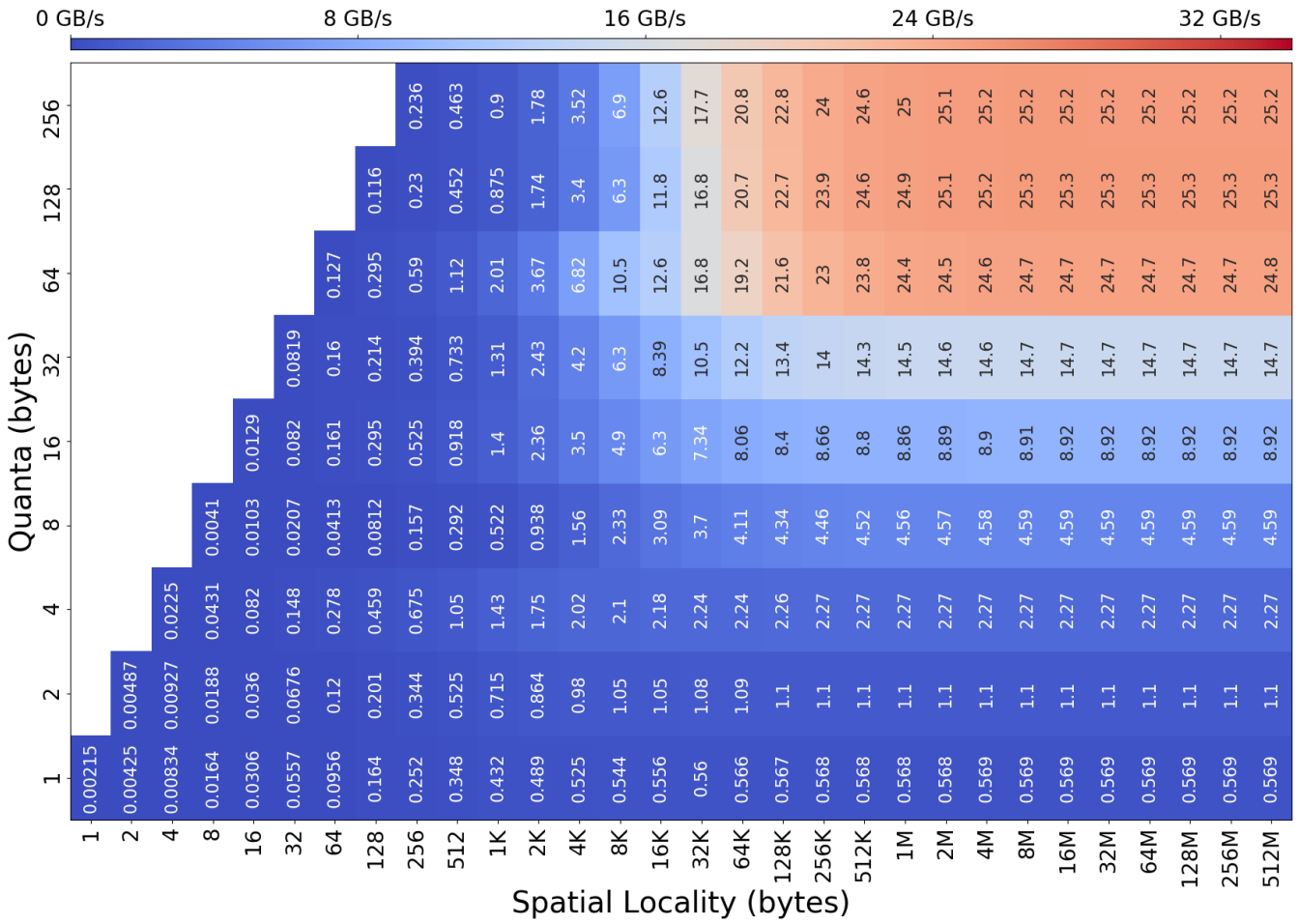


FIGURE 6 Heatmap of Arria 10 memory bandwidth as a function memory access **quanta** and **Spatial locality.** Efficient DRAM access requires high spatial locality and coalesced data transfers.

Indirect memory access, unlike the AXPY example, prevents the compiler from exploiting any compile-time knowledge to co-locate BRAM with processing elements. As a result, the compiler must be able to route from any memory location to any processing element. Figure 8 shows aggregate BRAM bandwidth for an indirect memory access pattern ($y[:] = \alpha \times x[idx[:]]$). Each processing element is assigned with an index and the index value is not known until run time. This index is then used to reference to x, a dense vector of values accounting for 2MiB space on BRAM. The bandwidth is calculated based on data movement of both values and indices (32b).

Clearly, indirect memory access bandwidth is different from unit-stride. Single- and double-precision bandwidth is cut in half (lower frequency) while 8-bit increases by 50% at low concurrency. As one increases the number of processing elements, routing becomes more complex, frequency drops, and synthesizability is challenged beyond 64 processing elements. As such, we partition x, y, and idx into subsectors in order to maintain frequency and synthesizability beyond 64 elements. With 512 processing elements (8 subsectors), the aggregate space of x is $8 \times 2 = 16$MiB. Unlike DRAM and its dependence on spatial locality for bandwidth, BRAM bandwidth is fairly stable as one varies the indices from unit-stride to stanza to fully random.

5 | 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 (GEMM), sparse matrix-vector multiplication (SpMV), and Batched Smith-Waterman (BSW) [38,39]. These kernels exhibit highly varied arithmetic intensity, degrees of parallelism, and requirements for

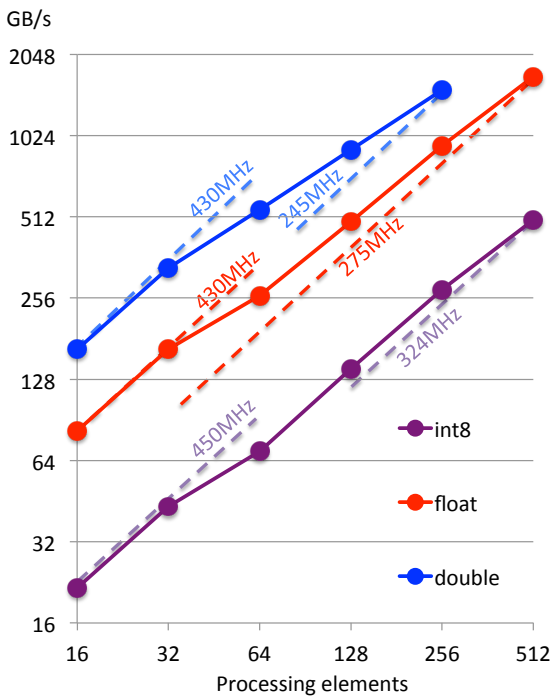


FIGURE 7 Unit-stride AXPY bandwidth from the BRAM of one U280 super logic region. Aggregated bandwidth depends on number of processing elements and the frequency. Small designs attain high frequency, but frequency drops as parallelism increases.

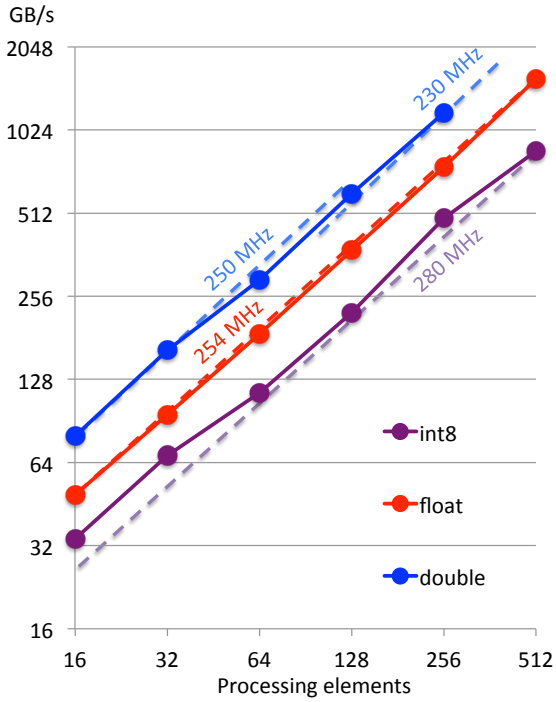


FIGURE 8 Indirect memory access bandwidth from BRAM. Even small designs suffer from the complex access pattern. Designs beyond 64 processing elements partition x, y, and idx into subsectors in order to maintain frequency and synthesizability.

fine-grained synchronization. For GEMM and SpMV, we use floating-point data types as they often appear in scientific applications. 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.

5.1 | 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 both single-precision SGEMM and double-precision DGEMM kernels on the Arria 10 and U280 and compare performance and energy efficiency to those 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. 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[40,41]. Data moves only between neighboring FPUs within a short distance, enabling the design to scale to many FPUs with modest frequency degradation. On CPUs and GPUs where the datapaths 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 GPUs. On the CPU, we rely on the Intel MKL library for single-core matrix multiplication. To perform GEMM on 32 cores of a Cori-Haswell node, we use MPI to implement the SUMMA algorithm[29] which can hide the communication overhead among processes.

Figures 9 and 11 plot SGEMM and DGEMM performance on the CPU, GPU, and FPGAs as a function of hardware resources (number of FPUs). We use large matrix sizes (up to 32K×32K on the CPU and GPU and 13K×13K on FPGAs). We vary hardware utilization on the CPU by controlling the number of MPI processes (thus the number of cores), on the GPU 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

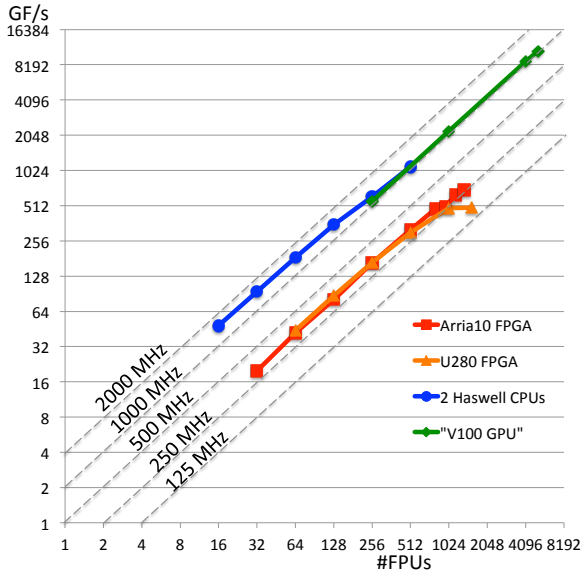


FIGURE 9 SGEMM performance scales linearly with hardware resources (FPUs). Effective frequency remains close to nominal frequency indicating a high percent of peak.

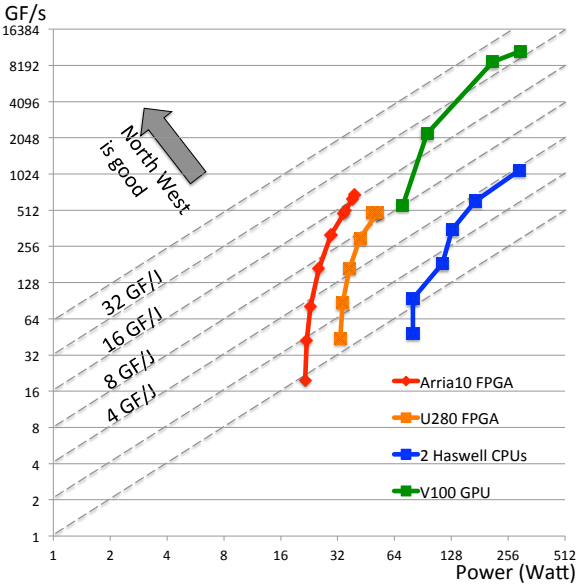


FIGURE 10 Although FPGA power is low and SGEMM energy efficiency approaching the 32 GFLOP/J ERT limit, GPUs still provide 2× higher energy efficiency.

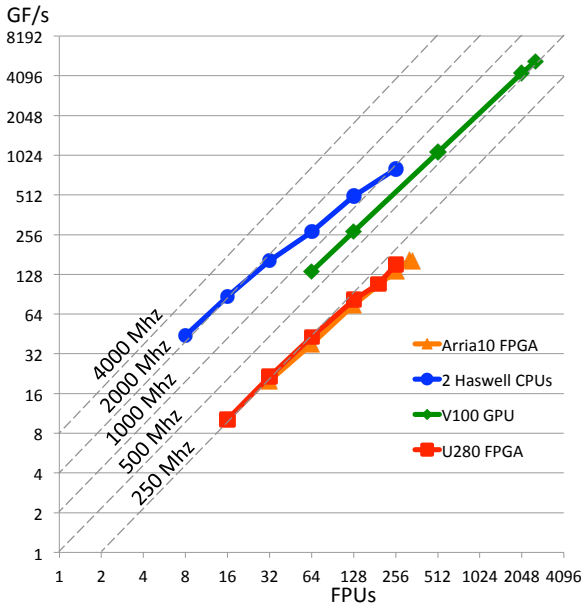


FIGURE 11 DGEMM scales poorly on FPGAs due to high hardware usage (it costs 4× more DSP blocks to synthesize a 64b FPU than a 32b FPU). On Haswell CPU and V100 GPU, the ratio between 32b and 64b FPUs is set to 2 .

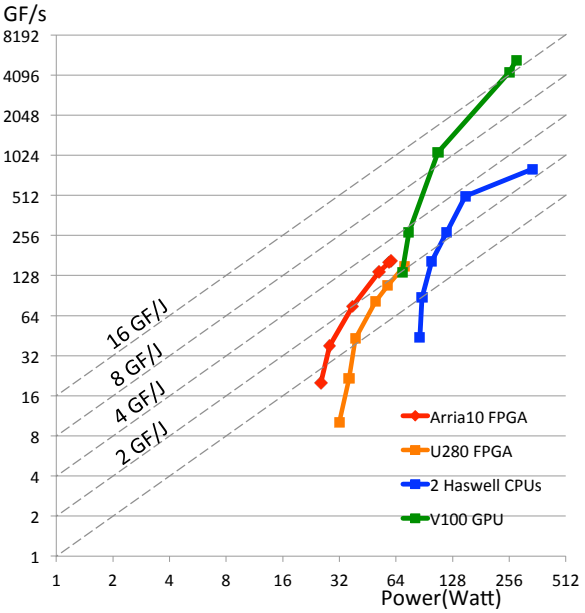


FIGURE 12 On all architectures, DGEMM performance is lower than SGEMM while power slightly increases. The impact on energy efficiency is most significant on FPGAs due to the major lost in performance and scalability.

fails (FPGAs). While straight lines in this figure denote perfect scaling, we may also use this formalism to define isocurves of "effective frequency" that denote the average frequency FPUs are clocked at ($\frac{GFLOP/s}{2\times\#FPUs}$).

Clearly, Most architectures scale very well with some loss in performance on HSW beyond 16 cores and on the U280 when using the full chip. Moreover, architectures trend towards a frequency isocurve slightly lower than their nominal frequencies (nominal V100 frequency is 1.5GHz but 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. As this FPGA consists of three super logic regions, designs

that span boundaries of these regions need to operate at a low clock frequency. Volta's 15× speedup for SGEMM over FPGAs can be attributed to 5× more FPUs and 3× higher frequency.

Although V100 delivers superior performance, it requires far more power to do so. Figure 10 and 12 plot SGEMM and DGEMM performance as a function of power for increasing hardware resource utilization (concurrency). Although a power-proportional architecture would be bounded along an isocurve of constant energy efficiency, we see all architectures show only modest increases in power for immense increases in concurrency and performance. Whereas the GPU and CPU require comparable power, the FPGAs see only moderate increases in power requiring less than one-seventh the power of a GPU. The GPU's SGEMM energy efficiency ultimately exceeds 32 GFLOP/J (approximately 31pJ/FLOP) while the Arria 10 and U280 FPGAs attain 16 and 8 GFLOP/J respectively. This is highly correlated with the ERT-based peak energy efficiency estimates from Figure 3.

DGEMM performance trends are similar to SGEMM's on all architectures. Nevertheless, the performance and scalability of DGEMM is reduced by a factor of two on the CPU and GPU and a factor of four on FPGAs due to limited hardware resources. CPUs and GPUs support double precision at the hardware level, and their designs guarantee that double-precision operations can be processed at half the rate of single precision. On FPGAs, one must synthesize double-precision units using single-precision or fixed-point DSP blocks. Inherently, this demands a double-precision floating-point unit consume at least 4× as many DSP blocks as a single-precision unit. As SGEMM and DGEMM attain the same effective frequency on FPGAs, the performance difference is attributable to the reduction in the number of floating-point units. Although CPUs and GPUs require roughly the same power for DGEMM as they do for SGEMM, DGEMM on FPGAs nearly doubles SGEMM power. The multiplicative effect of reduced performance and increased power results in FPGA energy efficiency below that of a CPU and far below that of a GPU.

5.2 | 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" that 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. Note, all SpMV experiments were conducted in single-precision.

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 Figure 5), it is comparable to the bandwidth of the MCDRAM-based KNL. On a GPU, with 128 workers our implementation yields almost 900GB/s. Figure 13 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 13, Figure 14 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 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 4.2.

SpMV: As FPGAs lack a cache, programmers must architect a solution to mitigate the random access associated with the source vector whilst preserving temporal and spatial locality. To that end, in our SpMV implementation, we create a replica of

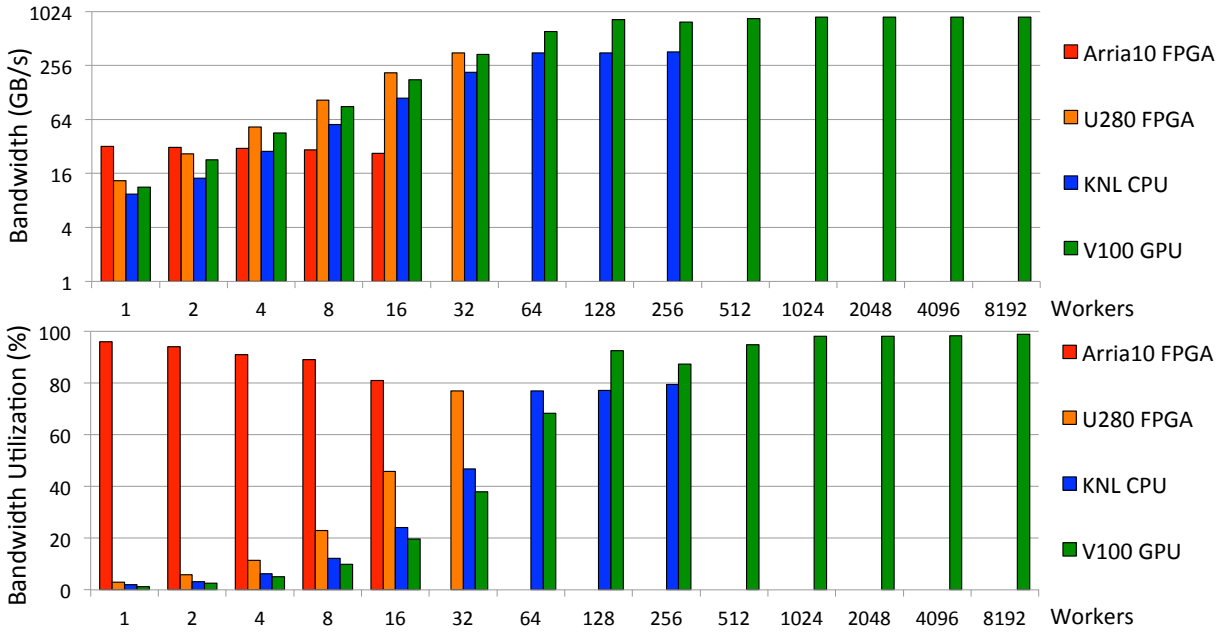


FIGURE 13 Bandwidth (top) and utilization (bottom) for a dot product as a function of the number of workers (concurrency). Observe superior GPU performance at high concurrency but high Arria 10 utilization at low concurrency.

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 15 shows SpMV realized bandwidth measured by $(8N + 8NNZ)/runtime$, where N is number of rows and 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 *multiDot* 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 16 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[42,43] 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/W, 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.

5.3 | Batched Smith-Waterman (BSW)

Metagenomic analysis is an important area of bioinformatics within DOE requiring HPC resources. merAligner[44] 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[38,39].

In this paper, we modify Muaaz Awan's *Batched Smith-Waterman* code[38] to map the application parallelisms on the Arria 10 and U280 FPGAs. At a high level, the local alignment simultaneously performs many alignment operations, one for each pair

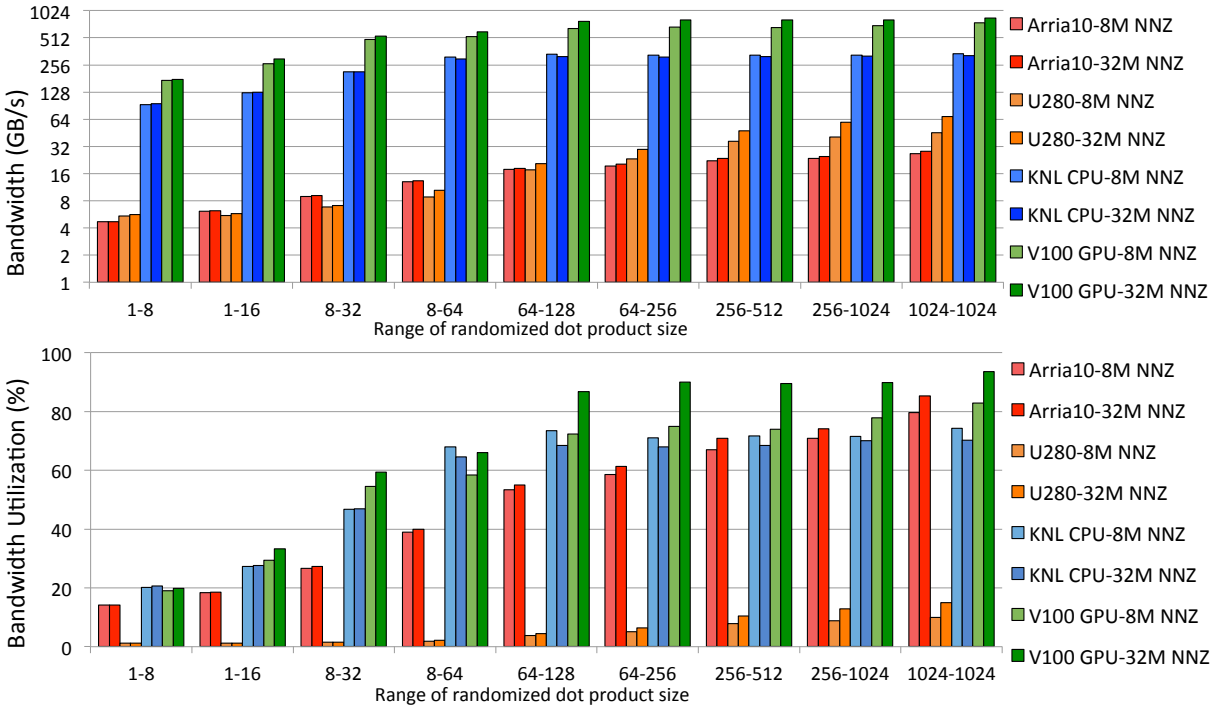


FIGURE 14 Memory bandwidth and utilization for multiple concurrent dot products of varying sizes. Observe the immense drop in bandwidth on the U280 relative to a dot product with a large number of workers.

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 architecture. 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 17 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 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 18 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 Arria 10 ultimately requires 10% and 40% less energy-to-solution than a GPU or single socket CPU respectively.

6 | FURTHER DISCUSSIONS

In this section, we detail the hardware designs presented in the previous section and develop models to highlight performance nuances and provide programmers with intuition. Not only do these activities improve performance and energy, but they also show code migration path to future FPGA architectures.

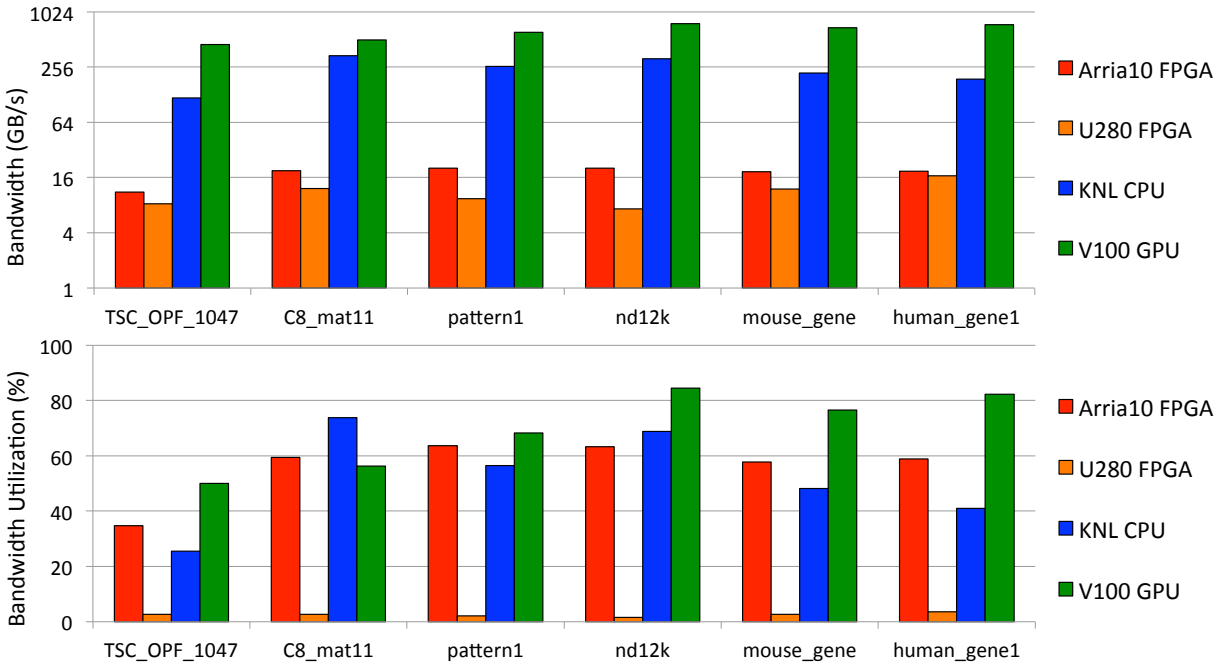


FIGURE 15 SpMV bandwidth and utilization for different matrices. NNZ/row increases from 250(left) to 1,107(right).

6.1 | Performance Modeling

Hardware vendors provide general guidelines on how to develop an optimal OpenCL kernel for FPGAs[45,46]. We also build application specific performance models for further understanding of the performance behavior of our kernels. Prior to writing any OpenCL FPGA code, we first envision the structure of a kernel's computation in terms of a dataflow graph. Such graphs include key compute components (e.g. FMAs, logical gates, memories/buffers, etc...) and the datapaths among them. As each component and datapath encodes a performance and bandwidth, these graphs, in conjunction with our ERT and stanza microbenchmarks, form the basis of our performance modeling efforts.

Figure 19 shows the systolic array we used as the basis for both the SGEMM and DGEMM kernels — a 2D mesh of processing elements (PEs), each an array of FMAs performing dot products. Blocks of A and B matrices are streamed from DRAM and fed into FIFO buffers along the left and top edges of the array. On each iteration of the algorithm, a block is processed by a PE and passed to neighboring PEs to maximize on-chip reuse. We may bound the performance of this design as twice the product of the number of FMAs with frequency. The performance model can be represented by the following equation.

$$Predicted\_Flop\_Rate = 2 \times \left(PE_{row} \times PE_{col} \times \#FMAs_{per\ PE}\right) \times frequency \tag{1}$$

Memory latency and bandwidth can be ignored with sufficiently large matrices and on-chip SRAM buffer capacity. Large input matrices allow for a long-running kernel, thereby amortizing the kernel launch overhead. With spacious SRAM buffer, we can use big tile sizes (e.g. 256×256), increasing the arithmetic intensity so that memory bandwidth is no longer a performance bottleneck. This simple bound is highly correlated with the performance data shown in Figures 9 and 24. For example, a 12x15x8 SGEMM systolic array attains 802GFLOP/s while Equation 1 suggests 835GFLOP/s. The small prediction error is due to the unaccounted overheads for filling up pipelines in the systolic array. ERT max performance is a bit higher, 930GFLOP/s. The performance difference can be well attributed to the attained frequencies and the number of FMAs. ERT's simple 1D benchmark can utilize all 1518 FMAs at 310MHz while the 12x15x8 SGEMM systolic array occupies 1440 FMAs operating at 290MHz.

SpMV, on the contrary, has inherently low arithmetic intensity. Thus, data movement and memory bandwidth cannot be ignored. Moreover, as only the vector x sees reuse, it should be cached in a local buffer while the values and column indices may be streamed from DRAM (see Fig. 20). We can bound the minimum run time as:

$$Predicted\_Run\_Time = \#nonzeros \times \left(sizeof(valueType) + sizeof(indexType)\right) / Memory\ Bandwidth. \tag{2}$$

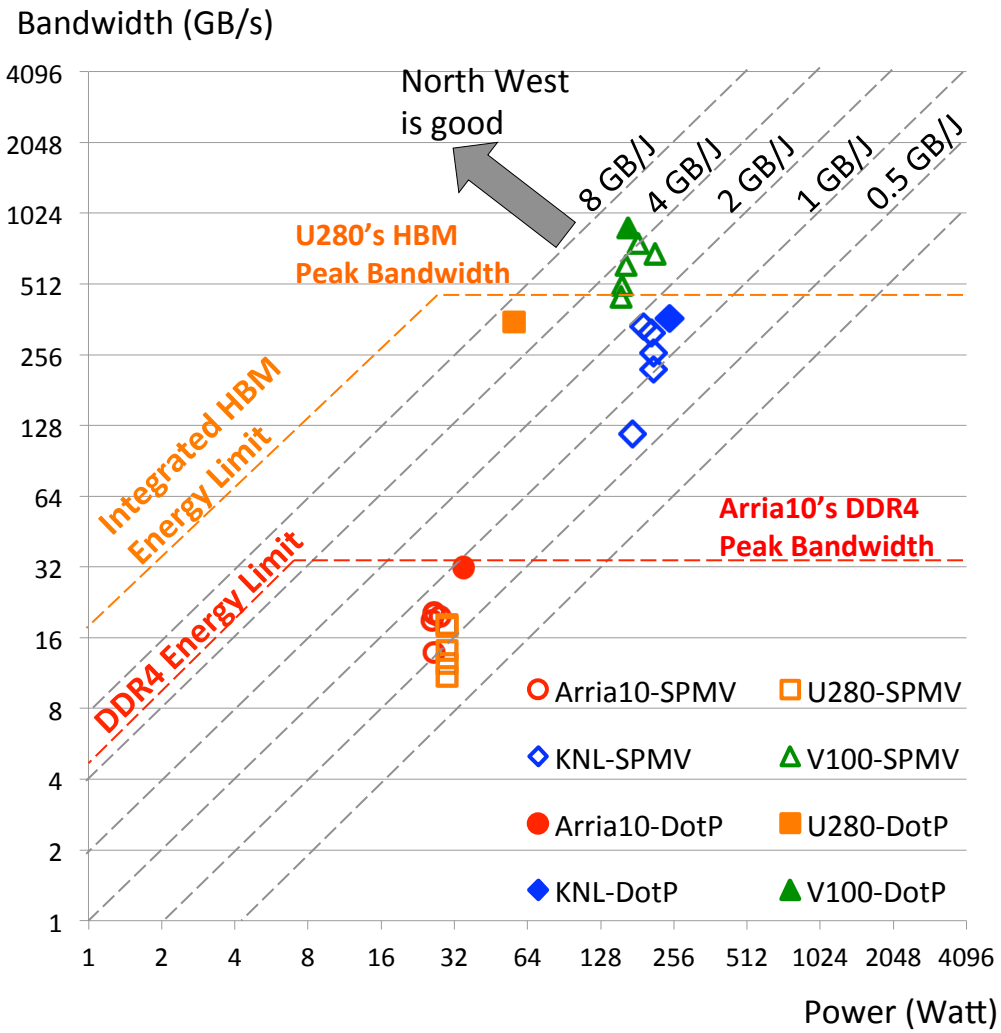


FIGURE 16 Dot product (closed symbol) and SpMV (open symbol per matrix) bandwidth and power. Observe isocurves of constant energy efficiency as well as HBM (orange) and DDR (red) technology's limits on energy efficiency.

Memory bandwidth would nominally only depend on the number of memory banks that we can synthesize. Unfortunately, FPGAs incur pipeline startup/empty stalls on each row which propagate into memory stalls. Matrices with few nonzeros per row exacerbate such penalties but can be quantified with the stanza benchmark from Section 4.2.

As for batched Smith Waterman (SW), the performance depends on how fast each pipelined SW processing element can perform an alignment, and how many of these independent processing elements can fit on an FPGA (Fig. 21). We found that both Intel and Xilinx high level synthesis tools can implement our design with an initiation interval of one, meaning that we can start evaluating a new cell of the SW table only one cycle after the previous cell started. The latency per cell is also a small number, i.e. up to 13 cycles, and can be amortized by the length of the reference genome sequence. The result is that the average cycles per cell is close to one. The performance in mega-cell-updates-per-second (MCUP) can be bounded as follows.

$$MCUP = \#SW\ processors \times frequency \tag{3}$$

The number of synthesizable SW processors depends on the number and the type of LUTs. We observed that the Arria10 FPGA has a bit fewer but more complex LUTs (8-input ALMs) than the Alveo U280 FPGA. The result is that we were able to synthesize 72 and 32 SW processors on Arria10 and Alveo U280 chips, respectively. Both run at roughly 156MHz. On Arria10, the kernel scales well and performance is close to our expectation. Using Equation 3, the predicted performance in MCUP is 11232 on Arria10. This implementation takes 14 seconds to perform 1 million sequence matching operations each containing

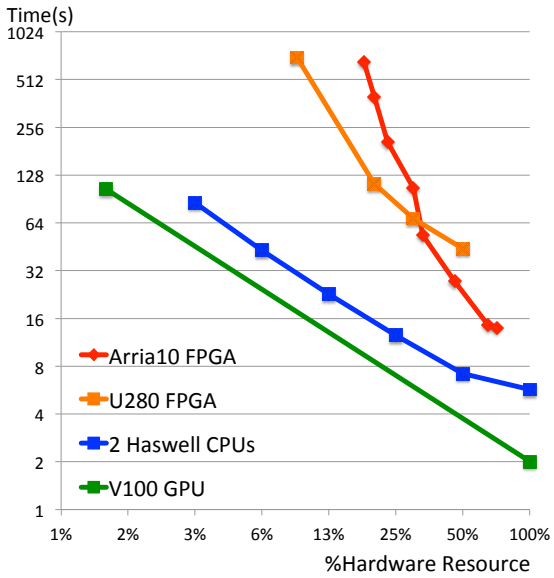


FIGURE 17 BSW Strong scaling performance as a function of hardware resources (FPGA LUTs, CPU Cores, GPU SMs)

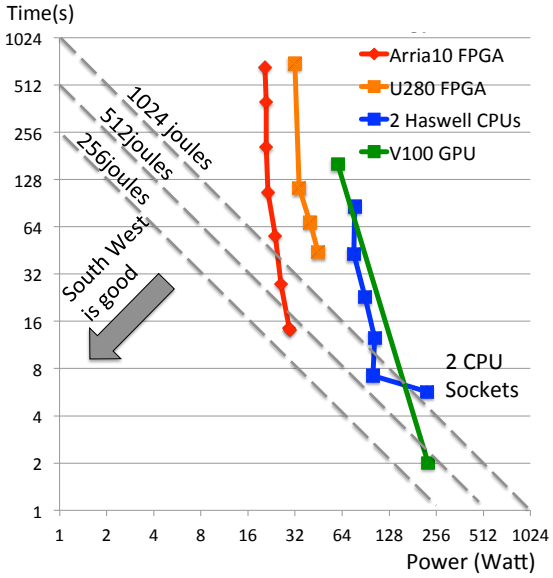


FIGURE 18 BSW performance, power, and energy efficiency as a function of hardware resources (each dot)

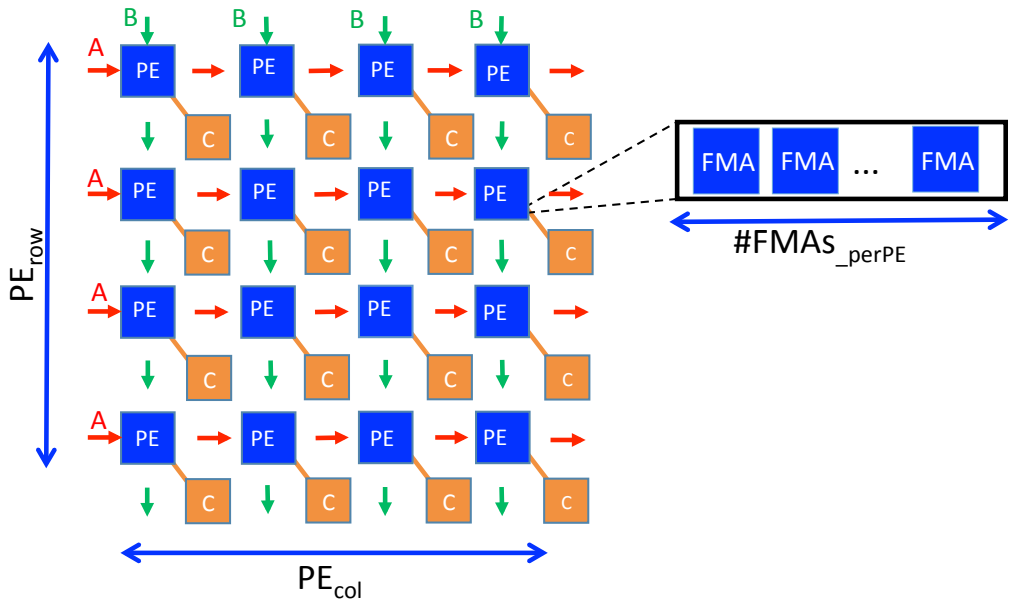


FIGURE 19 A systolic array architecture for GEMM. Processing elements (PEs) are wired with nearest neighbor connections to maximize the frequency. A and B are fed at the boundary and reused by other PEs. C is to keep partial dot products and will be drained to the DRAM when there is no more update from PEs.

128×1024 cell updates, attaining $128 \times 1024/14 = 9362$ MCUPs. The performance difference is due to some unaccounted data movement between DRAM and BRAM and startup cost of kernel pipelines.

6.2 | Design Space Exploration

Although accurate performance models can facilitate the selection of hardware configurations, actual performance may be less than the predicted performance bound depending on the hardware synthesis tool and kernel implementation details. In this

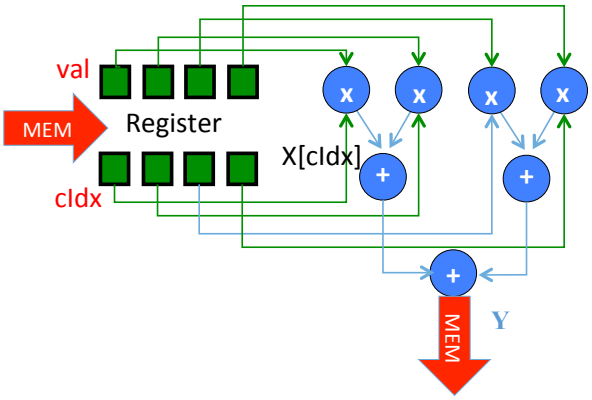


FIGURE 20 SpMV performance is dominated by the cost of moving value and index data from DRAM. This cost grows when there are more nonzeros in the matrix (more work) and fewer nonzeros per row (lower effective bandwidth).

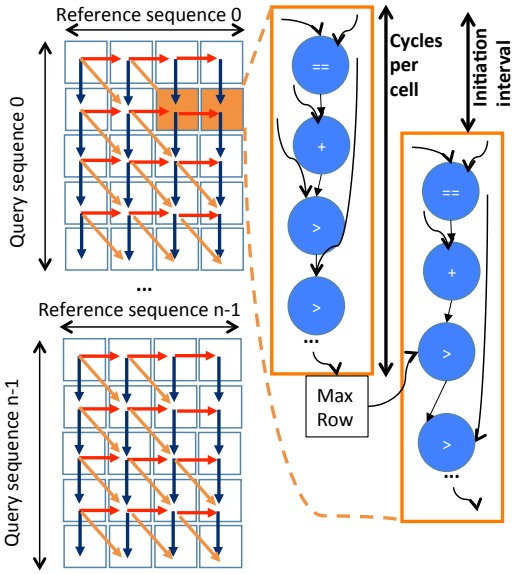


FIGURE 21 Batched Smith-Waterman throughput increases proportionally with the number of synthesized SW processors, which mainly depend on available LUTs and LUT type.

scenario, finding the optimal design requires additional tuning. Here we use SGEMM and DGEMM kernels on the Arria 10 to demonstrate an effective design space exploration combining domain knowledge and parameter space generation. The results also enable energy-aware designs by co-tuning performance and power.

Since matrix multiply is heavily compute-bound, the synthesized frequency has direct impact on performance. Tuning the architecture of the systolic array: number of processing element rows (R) and columns (C), $\#FMAs_{perPE}$ aka Vector Length (VL), and size of the matrix block held in SRAM ($BR \times BC \times VL$), one can find the configuration that maximizes frequency and thus performance. As one increases the number of FPUs ($R \times C \times VL$), one exploits more and more parallelism and thus raises the attainable performance of the FPGA. Conversely, as one increases the SRAM size, one increases arithmetic intensity as arithmetic intensity scales as $O(SRAM^{\frac{1}{2}})$.

We explore designs ranging from 32 to 1456 FMAs for single-precision and 32 to 336 for double-precision (n.b., a double-precision FMA requires about 4× DSP blocks as single-precision). We concurrently vary the block dimensions over a range that is big enough to exploit sufficient reuse to be compute-bound yet small enough to synthesize at high frequency. As each configuration requires 10 to 24 hours to synthesize, one must carefully bound/navigate the design space exploration.

Figures 22 and 23 show the performance and power for our range of SGEMM and DGEMM designs, respectively. To ease the comparison between designs, we normalize performance to that of the best iso-FMA configuration. Power results are presented in the same fashion. Clearly, most designs perform at a rate that is within 25% of the best configuration. This performance variation can be attributed to the achieved frequency. For single-precision, the frequency varies from 300MHz to 340MHz for small designs and from 220MHz to 290MHz for large designs. For double-precision, 240MHz-300MHz and 220MHz-260MHz are the frequency ranges for small and large designs, respectively. We also observe that as we increase the matrix block size, performance reaches its peak and then declines or oscillates at a lower performance level instead of forming a plateau. Whereas conventional wisdom on CPUs and GPUs which suggests using more cache capacity will only help performance, FPGAs defy this conventional wisdom as increased SRAM usage degrades synthesized frequency and performance.

Interestingly, power and performance are not directly correlated. Generally speaking, as we increase the aggregate size of the SRAM buffer, power consumption often declines. Increases in SRAM buffer size often reduce both frequency and DRAM bandwidth requirements. The latter is likely to have the greater impact on overall power, as we observe that off-chip data movement contributes to 50% to 75% of the dynamic power.

We may use the Roofline to understand Arria 10 SGEMM performance for a variety of designs and gain insights as to why some designs are more energy efficient. Figure 24 plots SGEMM performance as a function of the number of FPUs in the systolic array (symbol shape) and the size of the SRAM used by the array (consecutive dots for a given shape) against

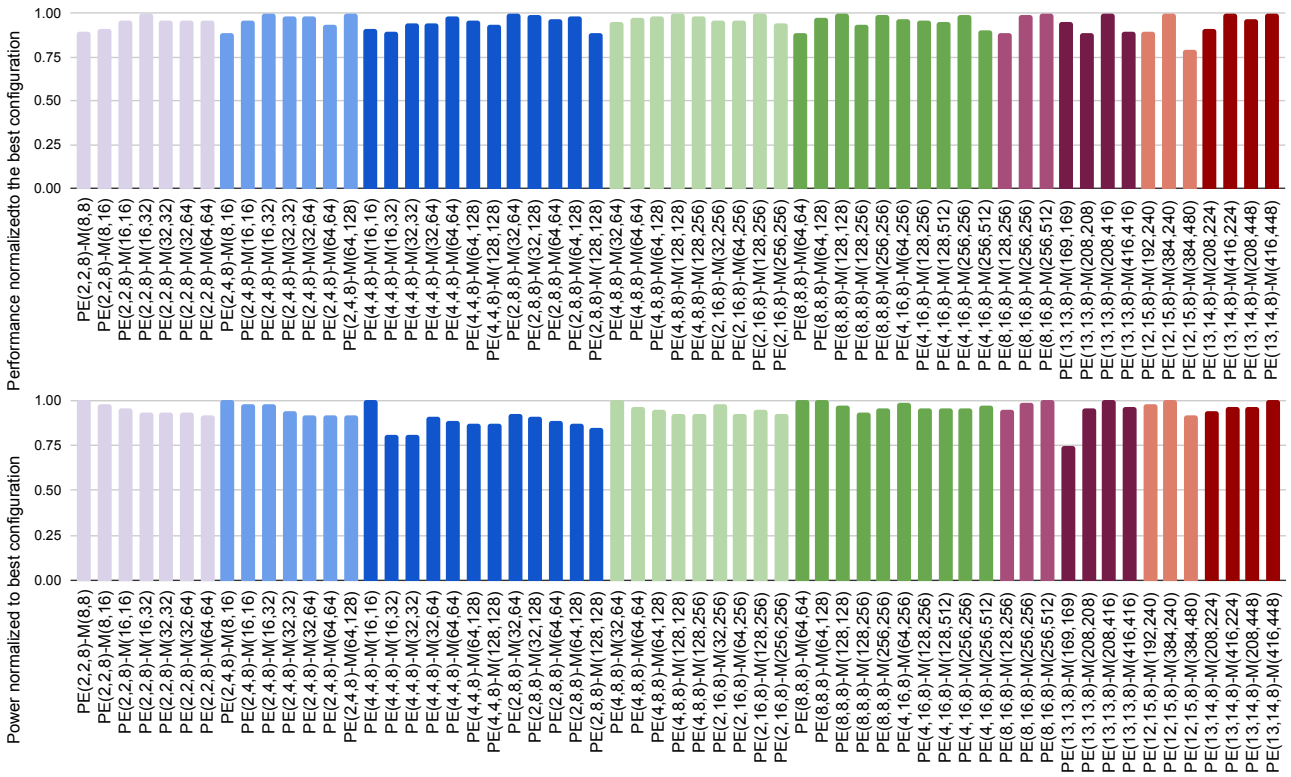


FIGURE 22 SGEMM design space exploration on the Arria 10: *PE(R,C,VL)* denotes a 2D systolic array of size R×C processing elements, each performing SIMD dot products of length VL. *M(BR,BC)* indicates SRAM buffer of size $BR \times BC \times VL \times sizeof(float)$. Performance (top) and power (bottom) are normalized to the best configuration that use the same number of DSP blocks.

ERT performance as a function of unrolling factor. Broadly speaking, performance is well-bounded by the number of FPUs in the array. Similarly, as one increases SRAM buffer size, arithmetic intensity increases (DRAM data movement decreases), but performance remains capped at the number of FPUs. Clearly, the requisite unrolling factor required to move the SRAM buffer to/from DRAM decreases with increasing SRAM capacity. Moreover, the increased arithmetic intensity afforded by increased SRAM capacity has the benefit of reducing average DRAM energy per element. Ultimately, one can exploit design space exploration to select either the performance-optimal configuration, the energy-optimal configuration, or the power-optimal configuration.

7 | 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 FPGA performance and bandwidth still fall 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%. For DGEMM, FPGAs perform poorly due to high area cost required to synthesize double-precision FPUs on regular DSP blocks.

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

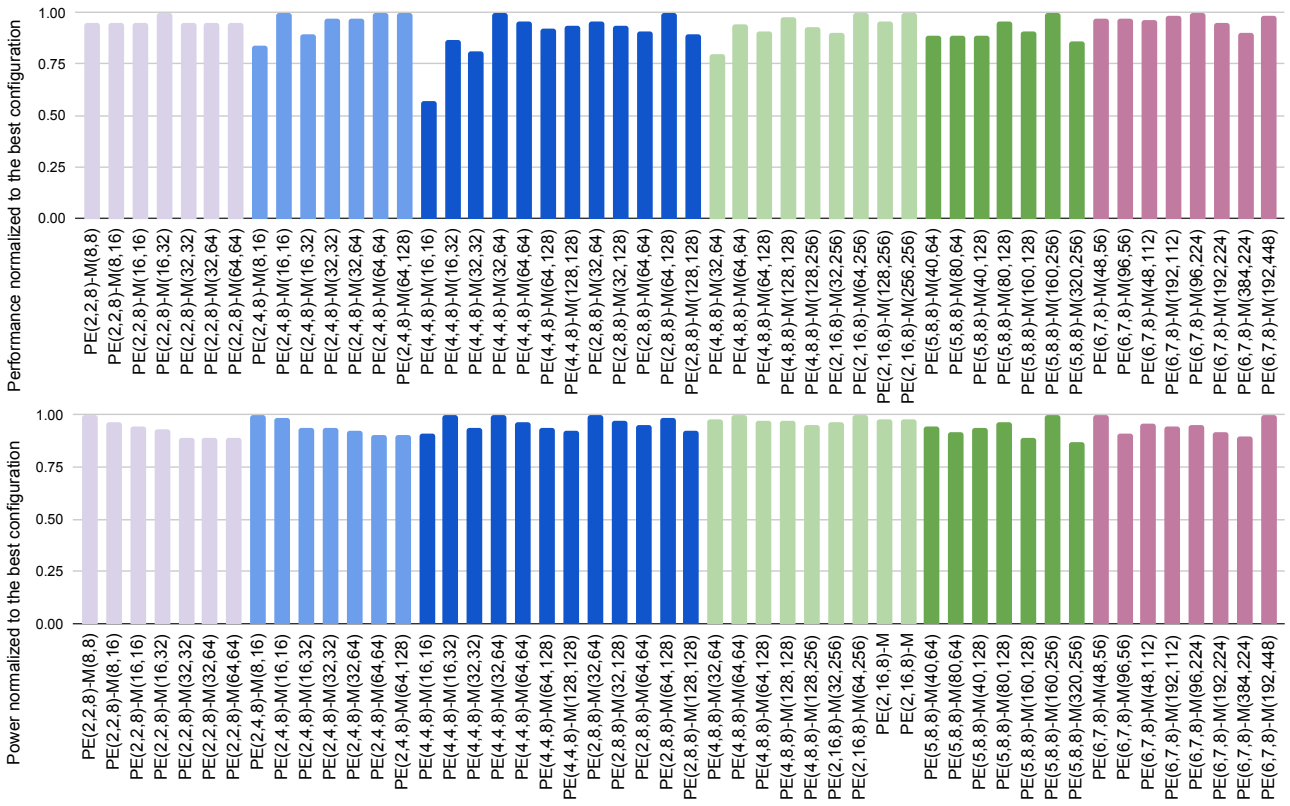


FIGURE 23 DGEMM design space exploration on on the Arria 10: *PE(R,C,VL)* denotes a 2D systolic array of size R×C processing elements, each performing SIMD dot products of length VL. *M(BR,BC)* indicates SRAM buffer of size $BR \times BC \times VL \times sizeof(double)$. Performance (top) and power (bottom) are normalized to the best configuration that use the same number of DSP blocks.

Ultimately, FPGA 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 scratchpads 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.

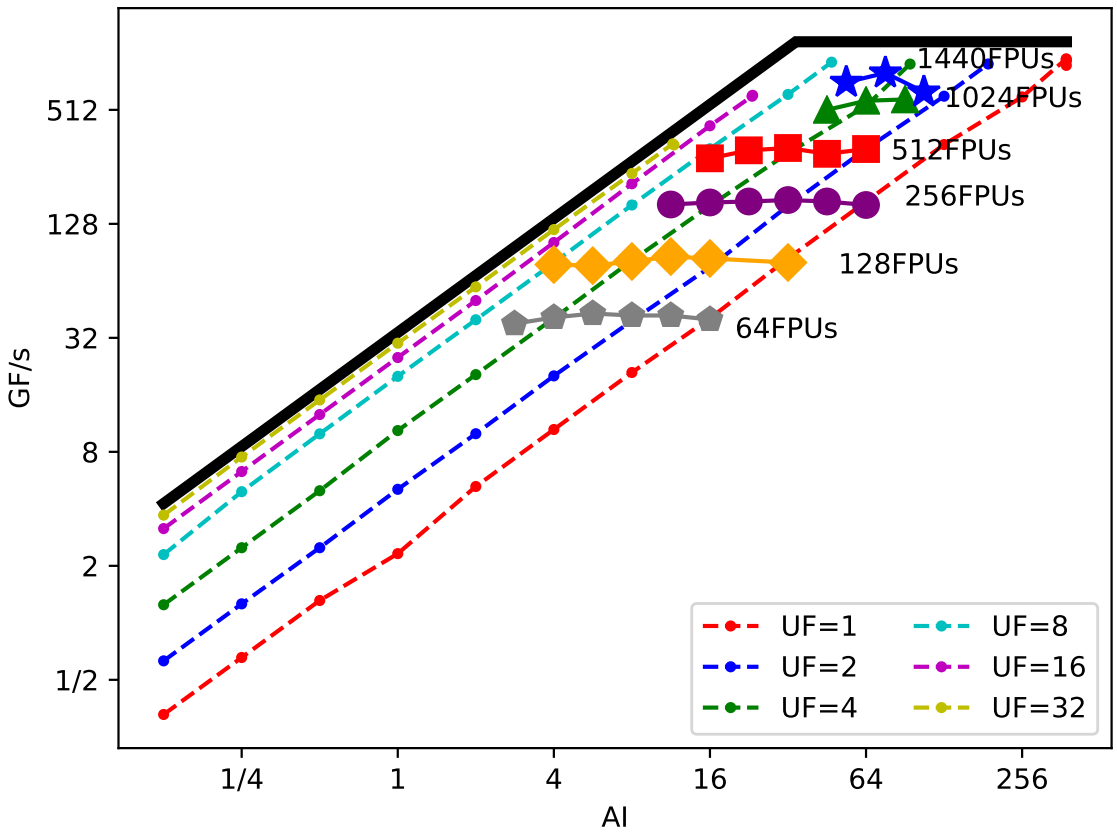


FIGURE 24 Arria 10 SGEMM arithmetic intensity and performance as a function of FPUs (symbol shape/color) and SRAM capacity (trendline of symbols) in the systolic array. SGEMM performance is capped by the number of FPUs and ultimately close to the ERT Roofline limit, while the requisite unrolling factor (UF) used to move the SRAM to/from DRAM decreases with increasing SRAM capacity.

References

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

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

3. Nguyen T, Williams S, Siracusa M, MacLean C, Doerfler D, Wright NJ. The Performance and Energy Efficiency Potential of FPGAs in Scientific Computing. In: 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS). ; 2020: 8-19

4. Williams S, Waterman A, Patterson D. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Commun. ACM* 2009; 52(4): 65–76. doi: 10.1145/1498765.1498785

5. Van Essen B, Macaraeg C, Gokhale M, Prenger R. Accelerating a Random Forest Classifier: Multi-Core, GP-GPU, or FPGA?. In: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines. ; 2012: 232-239

6. Nurvitadhi E, Sheffield D, Jaewoong Sim , Mishra A, Venkatesh G, Marr D. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In: 2016 International Conference on Field-Programmable Technology (FPT). ; 2016: 77-84

7. Nurvitadhi E, Venkatesh G, Sim J, et al. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?. In: FPGA '17. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. Association for Computing Machinery; 2017; New York, NY, USA: 5–14

8. Zhang Y, Shalabi YH, Jain R, Nagar KK, Bakos JD. FPGA vs. GPU for sparse matrix vector multiply. In: 2009 International Conference on Field-Programmable Technology. ; 2009: 255-262.

9. Kestur S, Davis JD, Williams O. BLAS Comparison on FPGA, CPU and GPU. In: 2010 IEEE Computer Society Annual Symposium on VLSI. ; 2010: 288-293.

10. Asano S, Maruyama T, Yamaguchi Y. Performance comparison of FPGA, GPU and CPU in image processing. In: 2009 International Conference on Field Programmable Logic and Applications. ; 2009: 126-131.

11. Birk M, Zapf M, Balzer M, Ruiter N, Becker J. A Comprehensive Comparison of GPU- and FPGA-Based Acceleration of Reflection Image Reconstruction for 3D Ultrasound Computer Tomography. *J. Real-Time Image Process.* 2014; 9(1): 159–170. doi: 10.1007/s11554-012-0267-4

12. Che S, Li J, Sheaffer JW, Skadron K, Lach J. Accelerating Compute-Intensive Applications with GPUs and FPGAs. In: 2008 Symposium on Application Specific Processors. ; 2008: 101-107.

13. Véstias M, Neto H. Trends of CPU, GPU and FPGA for high-performance computing. In: 2014 24th International Conference on Field Programmable Logic and Applications (FPL). ; 2014: 1-6.

14. Krommydas K, Feng W, Owaida M, Antonopoulos CD, Bellas N. 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: 153-160.

15. Zohouri HR, Maruyama N, Smith A, Matsuda M, Matsuoka S. 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: 409-420

16. Cong J, Fang Z, Lo M, Wang H, Xu J, Zhang S. Understanding Performance Differences of FPGAs and GPUs. In: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). ; 2018: 93-96.

17. Che S, Boyer M, Meng J, et al. Rodinia: A Benchmark Suite for Heterogeneous Computing. In: IISWC '09. Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC). IEEE Computer Society; 2009; USA: 44–54

18. Meyer M, Kenter T, Plessl C. Evaluating FPGA Accelerator Performance with a Parameterized OpenCL Adaptation of the HPCChallenge Benchmark Suite. *arXiv preprint arXiv:2004.11059* 2020.

19. Wang Z, Huang H, Zhang J, Alonso G. Shuhai: Benchmarking High Bandwidth Memory On FPGAS. In: 2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). ; 2020: 111-119.

20. Siracusa M, Rabozzi M, Del Sozzo E, Di Tucci L, Williams S, Santambrogio MD. A CAD-based methodology to optimize HLS code via the Roofline model. In: 2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). ; 2020.

21. Cardoso J, Silva dB, Braeken A, D'Hollander EH, Touhafi A. Performance Modeling for FPGAs: Extending the Roofline Model with High-Level Synthesis Tools. *International Journal of Reconfigurable Computing* 2013; 2013: 428078. doi: 10.1155/2013/428078

22. Choi JW, Bedard D, Fowler R, Vuduc R. A Roofline Model of Energy. In: 2013 IEEE 27th International Symposium on Parallel and Distributed Processing. ; 2013: 661-672.

23. Ilic A, Pratas F, Sousa L. Beyond the Roofline: Cache-Aware Power and Energy-Efficiency Modeling for Multi-Cores. *IEEE Transactions on Computers* 2017; 66(1): 52-58.

24. Calore E, Schifano SF. Energy-Efficiency Evaluation of FPGAs for Floating-Point Intensive Workloads. In: ADVANCES IN PARALLEL COMPUTING. International Conference on Parallel Computing (ParCo). IOS Press; 2020; PAESI BASSI: 555–564

25. Roberts SI, Wright SA, Fahmy SA, Jarvis SA. The Power-Optimised Software Envelope. *ACM Trans. Archit. Code Optim.* 2019; 16(3). doi: 10.1145/3321551

26. Intel . Intel Arria 10 Device Overview. [Online] https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf; . Accessed: 2020-09-12.

27. XILINX . Alveo U280 Data Center Accelerator Card. [Online] https://www.xilinx.com/products/boards-and-kits/alveo/u280.html; .

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

29. Geijn v. dRA, Watts J. SUMMA: Scalable Universal Matrix Multiplication Algorithm. tech. rep., University of Texas at Austin; USA: 1995.

30. ORNL . Titan Supercomputer. [Online] https://www.olcf.ornl.gov/olcf-resources/compute-systems/titan/; . Accessed May 25th, 2021.

31. ORNL . Summit User Guide. [Online] https://docs.olcf.ornl.gov/systems/summit_user_guide.html; . Accessed May 25th, 2021.

32. NERSC . Perlmutter Supercomputer. [Online] https://www.nersc.gov/systems/perlmutter/; . Accessed May 25th, 2021.

33. NVIDIA . NVIDIA HGX Platform. [Online] https://www.nvidia.com/en-us/data-center/hgx/; . Accessed May 30th, 2021.

34. Empirical Roofline Toolkit (ERT). [Computer Software]; . Accessed: 2020-08-01.

35. Yang C, Gayatri R, Kurth T, et al. An Empirical Roofline Methodology for Quantitatively Assessing Performance Portability. In: P3HPC Workshop at SC. ; 2018.

36. Kamil S, Husbands P, Oliker L, Shalf J, Yelick K. Impact of modern memory subsystems on cache optimizations for stencil computations. In: Workshop on Memory System Performance (MSP). ; 2005.

37. XILINX . AXI Interconnect. [Online] https://www.xilinx.com/products/intellectual-property/axi_interconnect.html; . Accessed May 25th, 2021.

38. Yelick KA, Oliker L, Awan MG. GPU accelerated Smith-Waterman for performing batch alignments (GPU-BSW) v1.0. [Computer Software] https://doi.org/10.11578/dc.20191223.1; 2019.

39. Farrar M. Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics* 2006; 23(2): 156-161. doi: 10.1093/bioinformatics/btl582

40. Kung SY. *VLSI Array Processors*. USA: Prentice-Hall, Inc. . 1987.

41. Fine Licht dJ, Kwasniewski G, Hoefler T. Flexible Communication Avoiding Matrix Multiplication on FPGA with High-Level Synthesis.. *CoRR* 2019; abs/1912.06526.

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

43. Bergman K, Hendry G, Hargrove P, et al. Let There Be Light! The Future of Memory Systems is Photonics and 3D Stacking. In: MSPC '11. Proceedings of the 2011 ACM SIGPLAN Workshop on Memory Systems Performance and Correctness. Association for Computing Machinery; 2011; New York, NY, USA: 43–48

44. Georganas E, Buluç A, Chapman J, Oliker L, Rokhsar D, Yelick KA. 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: 561–570

45. Intel . Intel® FPGA SDK for OpenCL™ Pro Edition. [Online] https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf; . Accessed May 30th, 2021.

46. XILINX . Vitis Unified Software Development Platform 2020.2 Documentation. [Online] https://www.xilinx.com/html_docs/xilinx2020_2/vitis_doc/index.html; . Accessed May 30th, 2021.

47. Intel . Intel® oneAPI DPC++ Compiler(Beta). [Online] https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/dpc-compiler.html; . Accessed September 8th, 2020.