In-Depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms

YOUNG-KYU CHOI, JASON CONG, ZHENMAN FANG[*], YUCHEN HAO, GLENN REINMAN, and PENG WEI[†],
Center for Domain-Specific Computing, University of California, Los Angeles

Conventional homogeneous multicore processors are not able to provide the continued performance and energy improvement that we have expected from past endeavors. Heterogeneous architectures that feature specialized hardware accelerators are widely considered a promising paradigm for resolving this issue. Among different heterogeneous devices, FPGAs that can be reconfigured to accelerate a broad class of applications with orders-of-magnitude performance/watt gains, are attracting increased attention from both academia and industry. As a consequence, a variety of CPU-FPGA acceleration platforms with diversified microarchitectural features have been supplied by industry vendors. Such diversity, however, poses a serious challenge to application developers in selecting the appropriate platform for a specific application or application domain.

This paper aims to address this challenge by determining which microarchitectural characteristics affect performance, and in what ways. Specifically, we conduct a quantitative comparison and an in-depth analysis on five state-of-the-art CPU-FPGA acceleration platforms: 1) the Alpha Data board and 2) the Amazon F1 instance that represent the traditional PCIe-based platform with private device memory; 3) the IBM CAPI that represents the PCIe-based system with coherent shared memory; 4) the first generation of the Intel Xeon+FPGA Accelerator Platform that represents the QPI-based system with coherent shared memory; and 5) the second generation of the Intel Xeon+FPGA Accelerator Platform that represents a hybrid PCIe (non-coherent) and QPI (coherent) based system with shared memory. Based on the analysis of their CPU-FPGA communication latency and bandwidth characteristics, we provide a series of insights for both application developers and platform designers. Furthermore, we conduct two case studies to demonstrate how these insights can be leveraged to optimize accelerator designs. The microbenchmarks used for evaluation have been released for public use.

CCS Concepts: • **Computer systems organization → Heterogeneous (hybrid) systems**.

Additional Key Words and Phrases: Heterogeneous Computing, CPU-FPGA Platform, Xeon+FPGA, CAPI, AWS F1

ACM Reference Format:
Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2018. In-Depth Analysis on Microarchitectures of Modern Heterogeneous CPU-FPGA Platforms. *ACM Trans. Reconfig. Technol. Syst.* 1, 1, Article 1 (January 2018), 21 pages. https://doi.org/10.1145/3294054

Extension of Conference Paper [10].
[*]Zhenman Fang is also an Adjunct Professor in Simon Fraser University, Canada
[†]Peng Wei is the corresponding author, email: peng.wei.prc@cs.ucla.edu

Authors' address: Young-kyu Choi; Jason Cong; Zhenman Fang; Yuchen Hao; Glenn Reinman; Peng Wei
Center for Domain-Specific Computing, University of California, Los Angeles.

1936-7406/2018/1-ART1 $15.00
https://doi.org/10.1145/3294054

1 INTRODUCTION

In today's datacenter designs, power and energy efficiency have become two of the primary constraints. The increasing demand for energy-efficient high-performance computing has stimulated a growing number of heterogeneous architectures that feature hardware accelerators or coprocessors, such as GPUs (graphics processing units), FPGAs (field-programmable gate arrays), and ASICs (application-specific integrated circuits). Among various heterogeneous acceleration platforms, the FPGA-based approach is considered one of the most promising directions, since FPGAs provide low power and high energy efficiency, and can be reprogrammed to accelerate different applications. Motivated by such advantages, leading cloud service providers have begun to incorporate FPGAs in their datacenters. For instance, Microsoft has designed a customized FPGA board called Catapult and integrated it into conventional computer clusters to accelerate large-scale production workloads, such as search engines [24] and neural networks [23]. In its Elastic Compute Cloud (EC2), Amazon also introduces the F1 compute instance [2] that equips a server with one or more FPGA boards. Intel, with its $16.7 billion acquisition of Altera, has predicted that approximately 30% of servers could have FPGAs in 2020 [1], indicating that FPGAs can play an important role in datacenter computing.

With the trend of adopting FPGAs in datacenters, various CPU-FPGA acceleration platforms with diversified microarchitectural features have been developed. We classify state-of-the-art CPU-FPGA platforms in Table 1 according to their physical integration and memory models. Traditionally, the most widely used integration is to connect an FPGA to a CPU via the PCIe interface, with both components equipped with private memories. Many FPGA boards built on top of Xilinx or Intel FPGAs use this way of integration because of its extensibility. The customized Microsoft Catapult board integration is such an example. Another example is the Alpha Data FPGA board [30] with the Xilinx FPGA fabric that can leverage the Xilinx SDAccel development environment [3] to support efficient accelerator design using high-level programming languages, including C/C++ and OpenCL. The Amazon F1 instance also adopts this software/hardware environment to allow high-level accelerator design. On the other hand, vendors like IBM tend to support a PCIe connection with a coherent, shared memory model for easier programming. For example, IBM has been developing the Coherent Accelerator Processor Interface (CAPI) on POWER8 [27] for such an integration, and has used this platform in the IBM data engine for NoSQL [5]. Meanwhile, the CCIX consortium has proposed the Cache Coherent Interconnect for Accelerators which can connect FPGAs with ARM processors through the PCIe interface with coherent shared memory as well [4]. More recently, closer CPU-FPGA integration becomes available using a new class of processor interconnects such as front-side bus (FSB) and the newer QuickPath Interconnect (QPI), and provides a coherent, shared memory, such as the FSB-based Convey machine [6] and the Intel Xeon+FPGA accelerator platform [22]. While the first generation of the Xeon+FPGA platform (Xeon+FPGA v1) connects a CPU to an FPGA only through a coherent QPI channel, the second generation of the Xeon+FPGA platform (Xeon+FPGA v2) adds two non-coherent PCIe data communication channels between the CPU and the FPGA, resulting in a hybrid CPU-FPGA communication model.

The evolution of various CPU-FPGA platforms brings up a challenging question: which platform should we choose to gain better performance and energy efficiency for a given application to accelerate? There are numerous factors that can affect the choice, such as platform cost, programming models and efforts, logic resource and frequency of FPGA fabric, CPU-FPGA communication latency and bandwidth, to name just a few. While some of them are easy to figure out, others are nontrivial, especially the communication latency and bandwidth between CPU and FPGA under different integration. One reason is that there are few publicly available documents for the newly announced platforms like the Xeon+FPGA family, CAPI and Amazon F1 instance. More importantly, those

Table 1. Classification of modern CPU-FPGA platforms

	Separate Private Memory	Shared Memory
PCIe Peripheral Interconnect	Alpha Data [30], Microsoft Catapult [24], Amazon F1 [2]	IBM CAPI [27], CCIx [4]
Processor Interconnect	N/A	Intel Xeon+FPGA v1 [22] (QPI), Convey HC-1 [6] (FSB)
Hybrid	N/A	Intel Xeon+FPGA v2 (QPI, PCIe)

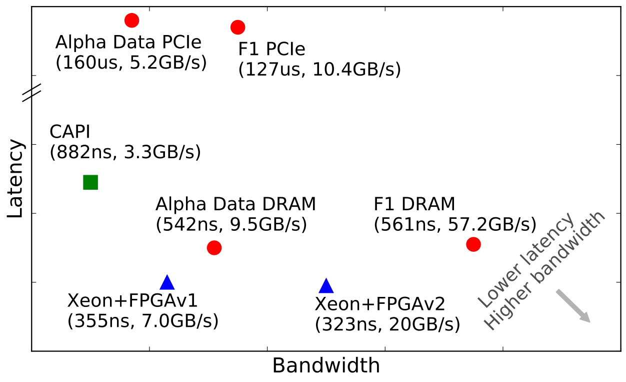


Fig. 1. Summary of CPU-FPGA communication bandwidth and latency (not to scale)

architectural parameters in the datasheets are often advertised values, which are usually difficult to achieve in practice. Actually, sometimes there could be a huge gap between the advertised numbers and practical numbers. For example, the advertised bandwidth of the PCIe Gen3 x8 interface is 8GB/s; however, our experimental results show that the PCIe-equipped Alpha Data platform can only provide 1.6GB/s PCIe-DMA bandwidth using OpenCL APIs implemented by Xilinx (see Section 3.2.1). Quantitative evaluation and in-depth analysis of such kinds of microarchitectural characteristics could help CPU-FPGA platform users to accurately predict the performance of a computation kernel to accelerate on various candidate platforms, and make the right choice. Furthermore, it could also benefit CPU-FPGA platform designers for identifying performance bottlenecks and providing better hardware and software support.

Motivated by those potential benefits to both platform users and designers, this paper aims to discover which microarchitectural characteristics affect the performance of modern CPU-FPGA platforms, and evaluate what that effect will be. We conduct our quantitative comparison on five state-of-the-art CPU-FPGA platforms: 1) the Alpha Data board and 2) Amazon F1 instance that represent the conventional PCIe-based platform with private device memory, 3) IBM CAPI that represents the PCIe-based system with coherent shared memory, 4) Intel Xeon+FPGA v1 that represents the QPI-based system with coherent shared memory, and 5) Xeon+FPGA v2 that represents a hybrid PCIe (non-coherent) and QPI (coherent) based system with shared memory. These five platforms cover various CPU-FPGA interconnection approaches, and different memory models as well.

In summary, this paper makes the following contributions.

1. The first quantitative characterization and comparison on the microarchitectures of state-of-the-art CPU-FPGA acceleration platforms—including the Alpha Data board and Amazon F1 instance, IBM CAPI, and Intel Xeon+FPGA v1 and v2—which covers the whole range of CPU-FPGA connections. We quantify each platform's CPU-FPGA communication latency and bandwidth and the results are summarized in Fig. 1.
2. An in-depth analysis of the big gap between advertised and practically achievable performance (Section 3), with step-by-step decomposition of the inefficiencies.
3. Seven insights for both application developers to improve accelerator designs and platform designers to improve platform support (Section 4). Specifically, we suggest that accelerator designers avoid using the advertised platform parameters to estimate the acceleration effect, which almost always leads to an overly optimistic estimation. Moreover, we analyze the trade-off between private-memory and shared-memory platforms, and analytically model the effective bandwidth with the introduction of the memory data reuse ratio r. We also propose the metric of computation-to-communication (CTC) ratio to measure when the CPU-FPGA communication latency and bandwidth are critical. Finally, we suggest that the complicated communication stack and hard-to-use coherent cache system may improve in the next-generation of CPU-FPGA platforms.
4. Two case studies in real applications to demonstrate how these insights can be leveraged, including matrix multiplication and matrix-vector multiplication. The former is a well-known compute-intensive application, and the latter is bounded by communication. We use these two applications to demonstrate how to choose the appropriate platform by applying the proposed insights.

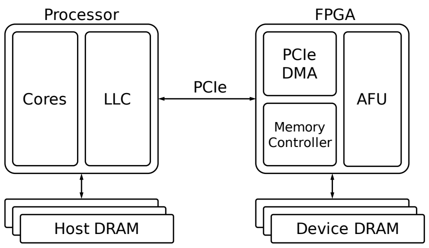
2 BACKGROUND

A high-performance interconnect between the host processor and FPGA is crucial to the overall performance of CPU-FPGA platforms. In this section we first summarize existing CPU-FPGA architectures with typical PCIe and QPI interconnects. Then we present the private and shared memory models of different platforms. Finally, we discuss related work.

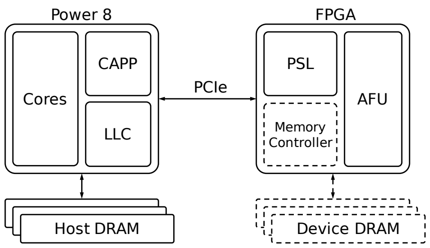
2.1 Common CPU-FPGA Architectures

Typical PCIe-based CPU-FPGA platforms feature direct memory access (DMA) and private device DRAM (Fig. 2(a)). To interface with the device DRAM as well as the host-side CPU-attached memory, a memory controller IP and a PCIe endpoint with a DMA IP are required to be implemented on the FPGA, in addition to user-defined accelerator function units (AFUs). Fortunately, vendors have provided hard IP solutions to enable efficient data copy and faster development cycles. For example, Xilinx releases device support for the Alpha Data card [30] in the SDAccel development environment [3]. As a consequence, users can focus on designing application-related AFUs and easily swap them into the device support to build customized CPU-FPGA acceleration platforms.

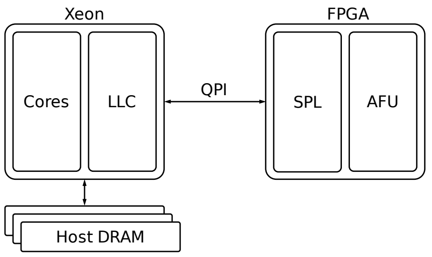
IBM integrates the Coherent Accelerator Processor Interface (CAPI) [27] into its Power8 and future systems, which provides virtual addressing, cache coherence and virtualization for PCIe-based accelerators (Fig. 2(b)). A coherent accelerator processor proxy (CAPP) unit is introduced to the processor to maintain coherence for the off-chip accelerator. Specifically, it maintains the directory of all cache blocks of the accelerator, and it is responsible for snooping the CPU bus for cache block status and data on behalf of the accelerator. On the FPGA side, IBM also supplies a power service layer (PSL) unit alongside the user AFU. The PSL handles address translation and coherency functions while sending and receiving traffic as native PCIe-DMA packets. With the



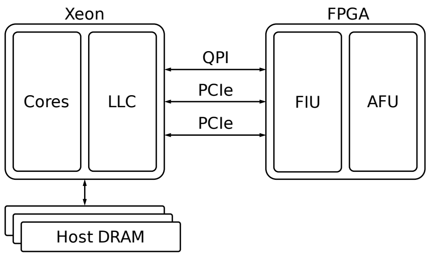
(a) Conventional PCIe-based platforms (e.g., Alpha Data board and F1 instance)



(b) PCIe-based CAPI platform



(c) QPI-based Xeon+FPGA v1



(d) Xeon+FPGA v2 with 1 QPI and 2 PCIe channels

Fig. 2. A tale of five CPU-FPGA platforms

ability to access coherent shared memory of the host core, device DRAM and memory controller become optional for users.

Intel Xeon+FPGA v1 [22] brings the FPGA one step closer to the processor via QPI where an accelerator hardware module (AHM) occupies the other processor socket in a 2-socket motherboard. By using QPI interconnect, data coherency is maintained between the last-level cache (LLC) in the processor and the FPGA cache. As shown in Fig. 2(c), an Intel QPI IP that contains a 64KB cache is required to handle coherent communication with the processor, and a system protocol layer (SPL) is introduced to provide address translation and request reordering to the user AFU. Specifically, a page table of 1024 entries, each associated with a 2MB page (2GB in total), is implemented in SPL, which will be loaded by the device driver during runtime. Though current addressable memory is limited to 2GB and private high-density memory for FPGA is not supported, this low-latency coherent interconnect has distinct implications for programming models and overall processing models of CPU-FPGA platforms.

Xeon+FPGA v2 co-packages the CPU and FPGA to deliver even higher bandwidth and lower latency than discrete forms. As shown in Fig. 2(d), the communication between CPU and FPGA is supported by two PCIe Gen3 x8 and one QPI (UPI in Skylake and later architectures) physical links. These are presented as virtual channels on the user interface. The FPGA logic is divided into two parts: the Intel-provided FPGA interface unit (FIU) and the user AFU. The FIU provides platform capabilities such as unified address space, coherent FPGA cache and partial reconfiguration of user AFU, in addition to implementing interface protocols for the three physical links. Moreover, a memory properties factory for higher-level memory services and semantics is supplied to provide a push-button development experience for end-users.

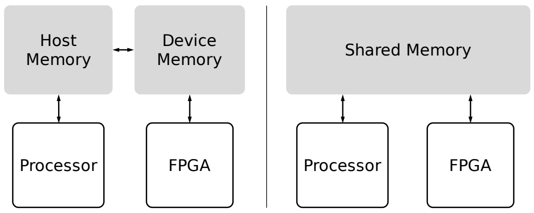


Fig. 3. Developer view of separate and shared memory spaces

2.2 CPU-FPGA Memory Models

Accelerators with physical addressing effectively adopt a separate address space paradigm (Fig. 3). Data shared between the host and device must be allocated in both the host-side CPU-attached memory and the private device DRAM, and explicitly copied between them by the host program. Although copying array-based data structures is straightforward, moving pointer-based data structures such as linked-lists and trees presents complications. Also, separate address spaces cause data replication, resulting in extra latency and overhead. To mitigate this performance penalty, users usually consolidate data movement into one upfront bulk transfer from the host memory to the device memory. In this paper the evaluated Alpha Data and Amazon F1 platforms fall into this category.

With tighter *logical* CPU-FPGA integration, the ideal case would be to have a unified shared address space between the CPU and FPGA. In this case (Fig. 3), instead of allocating two copies in both host and device memories, only a single allocation is necessary. This has a variety of benefits, including the elimination of explicit data copies, pointer semantics and increased performance of fine-grained memory accesses. CAPI enables unified address space through additional hardware module and operating system support. Cacheline-aligned memory spaces allocated using `posix_memalign` are allowed in the host program. Xeon+FPGA v1 provides the convenience of a unified shared address space using pinned host memory, which allows the device to directly access data on that memory location. However, users must rely on special APIs, rather than normal C or C++ allocation (e.g., `malloc/new`), to allocate pinned memory space.

Xeon+FPGA v2 supports both memory models by configuring the supplied memory properties factory, so that users can decide whether the benefit of having a unified address space outweighs the address translation overhead based on their use case.

2.3 Related Work

In this section we discuss three major categories of related work.

First, in addition to the commodity CPU-FPGA integrated platforms in Table 1, there is also a large body of academic work that focuses on how to efficiently integrate hardware accelerators into general-purpose processors. Yesil et al. [31] surveyed existing custom accelerators and integration techniques for accelerator-rich systems in the context of data centers, but without a quantitative study as we did. Chandramoorthy et al. [7] examined the performance of different design points including tightly coupled accelerators (TCAs) and loosely coupled accelerators (LCAs) customized for computer vision applications. Cotat et al. [15] specifically analyzed the integration and interaction of TCAs and LCAs at different levels in the memory hierarchy. CAMEL [12] featured reconfigurable fabric to improve the utilization and longevity of on-chip accelerators. All of these studies were done using simulated environments instead of commodity CPU-FPGA platforms.

Second, a number of approaches have been proposed to make accelerators more programmable by supporting shared virtual memory. NVIDIA introduced “unified virtual addressing” beginning with the Fermi architecture [21]. The Heterogeneous System Architecture Foundation announced heterogeneous Uniform Memory Accesses (hUMA) that will implement the shared address paradigm in future heterogeneous processors [25]. Cong et al. [11] propose supporting address translation using two-level TLBs and host page walk for accelerator-centric architectures. Shared virtual memory support for CPU-FPGA platforms has been explored in CAPI and the Xeon+FPGA family [22, 27]. This paper covers both the separate memory model (Alpha Data and F1 instance) and shared memory model (CAPI, Xeon+FPGA v1 and v2).

Third, there is also numerous work that evaluates modern CPU and GPU microarchitectures. For example, Fang et al. [16] evaluated the memory system microarchitectures on commodity multicore and many-core CPUs. Wong et al. [29] evaluated the microarchitectures on modern GPUs. This work is the first to evaluate the microarchitectures of modern CPU-FPGA platforms with an in-depth analysis.

3 CHARACTERIZATION OF CPU-FPGA MICROARCHITECTURES

This work aims to reveal how the underlying microarchitectures, i.e., processor or peripheral interconnect, and shared or private memory model, affect the performance of CPU-FPGA platforms. To achieve this goal, in this section we quantitatively study those microarchitectural characteristics, with a key focus on the effective bandwidth and latency of CPU-FPGA communication on five state-of-the-art platforms: Alpha Data, CAPI, Xeon+FPGA v1 and v2, and Amazon F1 instance.[1]

3.1 Experimental Setup

To measure the CPU-FPGA communication bandwidth and latency, we design and implement our own microbenchmarks, based on the Xilinx SDAccel SDK 2017.4 [3] for Alpha Data and F1 instance, Alpha Data CAPI Design Kit [17] for CAPI, and Intel AALSDK 5.0.3 [18] for Xeon+FPGA v1 and v2. Each microbenchmark consists of two parts: a host program and a computation kernel. Following each platform's typical programming model, we use the C language to write the host programs for all platforms, and describe the kernel design using OpenCL for Alpha Data and F1 instance, and Verilog HDL for the other three platforms.

The hardware configurations of Alpha Data, CAPI, Xeon+FPGA v1 and v2, and Amazon F1 in our study are listed in Table 2.

Table 2. Platform configurations of Alpha Data, F1, CAPI, Xeon+FPGA v1 and v2

Platform	Alpha Data	CAPI	Xeon+FPGA v1	Xeon+FPGA v2	Amazon EC2 F1
Host CPU	Xeon E5-2620v3 @2.40GHz	Power8 Turismo @4GHz	Xeon E5-2680v2 @2.80GHz	Xeon E5-2600v4 @2.4GHz	Xeon E5-2686v4 @2.3GHz
Host Memory	64GB DDR3-1600	16GB DDR3-1600	96GB DDR3-1600	64GB DDR4-2133	64GB DDR4-2133
FPGA Fabric	Xilinx Virtex 7 @200MHz	Xilinx Virtex 7 @250MHz	Intel Stratix V @200MHz	Intel Arria 10 @400MHz‡	Xilinx UltraScale+ @250MHz
CPU ↔ FPGA	PCIe Gen3 x8, 8GB/s	PCIe Gen3 x8, 8GB/s	Intel QPI, 12.8GB/s	1× Intel QPI & 2× PCIe Gen3 x8, 25.6GB/s	PCIe Gen3 x16, 16GB/s
Device Memory	16GB DDR3-1600	16GB DDR3-1600†	N/A	N/A	64GB DDR4-2133

† The device memory in CAPI is not used in this work.
‡ The user clock can be easily configured to 137/200/273 MHz using the supplied SDK, in addition to max 400MHz frequency.

3.2 Effective Bandwidth

3.2.1 Effective Bandwidth for Alpha Data. Traditional CPU-FPGA platforms like Alpha Data contain two communication phases: 1) PCIe-based direct memory access (DMA) between host memory

[1]Results in this publication were generated using pre-production hardware or software, and may not reflect the performance of future products.

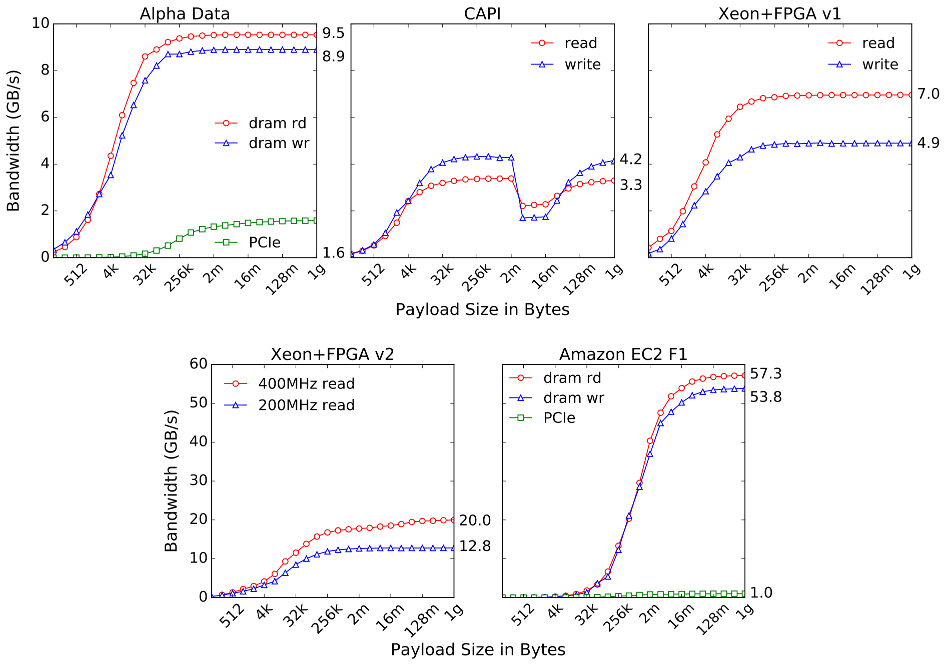
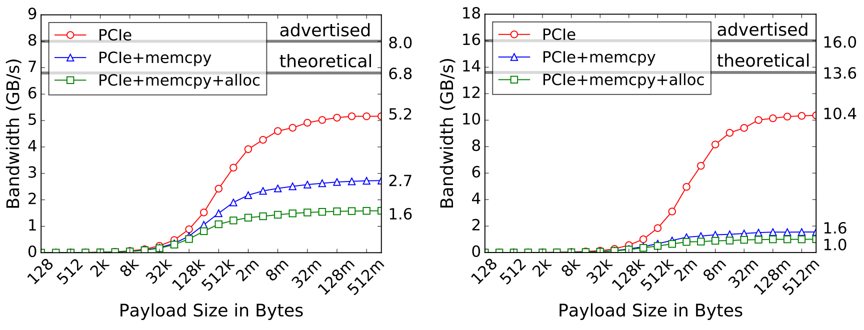


Fig. 4. Effective bandwidth of Alpha Data, CAPI, Xeon+FPGA v1 and v2, and F1 instance



(a) Alpha Data (b) Amazon EC2 F1

Fig. 5. PCIe-DMA bandwidth breakdown

and device memory, and 2) device memory access. We measure the effective bandwidths with various payload sizes for both phases. The measurement results are illustrated in Fig. 4. Since the bandwidths for both directions of PCIe-DMA transfer are almost identical (less than 4% difference), we only present the unidirection PCIe-DMA bandwidth in Fig. 4.

While Fig. 4 illustrates a relatively high private DRAM bandwidth (9.5GB/s for read, 8.9GB/s for write[2]), the PCIe-DMA bandwidth (1.6GB/s) reaches merely 20% of PCIe's advertised bandwidth (8GB/s). That is, the expectation of a high DMA bandwidth with PCIe is far from being fulfilled.

The first reason is that there is non-payload data overhead for the useful payload transfer [20]. In a PCIe transfer, a payload is split into small packets, each packet equipped with a header. Along with the payload packets, there are also a large number of packets for control purposes transferred through PCIe. As a result, the maximum supplied bandwidth for the actual payload, which we call the *theoretical* bandwidth, is already smaller than the advertised value.

Another important reason is that a PCIe-DMA transaction involves not only PCIe transfer, but also host buffer allocation and host memory copy [14]. The host memory stores user data in a pageable (unpinned) space from which the FPGA cannot directly retrieve data. A page-locked (pinned), physically contiguous memory buffer is in the operating system kernel space that serves as a staging area for PCIe transfer. When a PCIe-DMA transaction starts, a pinned buffer is first allocated in the host memory, followed by a memory copy of pageable data to this pinned buffer. The data is then transferred from the pinned buffer to device memory through PCIe. These three steps—buffer allocation, host memory copy, and PCIe transfer—are sequentially processed in Alpha Data, which significantly decreases the PCIe-DMA bandwidth.

Moreover, there could be some implementation deficiencies in the vendor-provided environment which serve as another source of overhead.[3] One possibility could be the data transfer overhead between the endpoint of the PCIe channel, i.e., the vendor-provided FPGA DMA IP, and the FPGA-attached device DRAM. Specifically, the device DRAM does not directly connect to the PCIe channel; instead, the data from the host side first reach the on-chip DMA IP and then are written into the device DRAM through the vendor-provided DRAM controller IP. If this extra step were not well overlapped with the actual data transfer via the PCIe channel through careful pipelining, it would further reduce the effective bandwidth. Our experiments show that a considerable gap still exists between the measured bandwidth and the theoretical value, indicating that the vendor-provided environment could be potentially improved with further performance tuning.

Next, we quantitatively evaluate the large PCIe-DMA bandwidth gap step by step, with results shown in Fig. 5.

1. The non-payload data transfer lowers the theoretical PCIe bandwidth to 6.8GB/s from the advertised 8GB/s [20].
2. Possible implementation deficiencies in the vendor-provided environment prevent the 6.8GB/s PCIe bandwidth from being fully exploited. As a result, the highest achieved effective PCIe-DMA bandwidth without buffer allocation and host memory copy decreases to 5.2GB/s.
3. The memory copy between the pageable and pinned buffers further degrades the PCIe-DMA bandwidth to 2.7GB/s.
4. The buffer allocation overhead degrades the final effective PCIe-DMA bandwidth to only 1.6GB/s. This is the actual bandwidth that end-users can obtain.

3.2.2 Effective Bandwidth for CAPI. CPU-FPGA platforms that realize the shared memory model, such as CAPI, Xeon+FPGA v1 and v2, allow the FPGA to retrieve data directly from the host memory. Such platforms therefore contain only one communication phase: host memory access through the communication channel(s). For the PCIe-based CAPI platform, we simply measure the effective read and write bandwidths of its PCIe channel for a variety of payload sizes, as shown in Fig. 4.

[2]If not specifically indicated, the bandwidth appearing in the remainder of this paper refers to the maximum achievable bandwidth.
[3]The Xilinx SDAccel environment is close-sourced, so we were not able to pin-point this overhead.

Compared to the Alpha Data board, CAPI supplies end-users with a much higher effective PCIe bandwidth (3.3GB/s vs. 1.6GB/s). This is because CAPI provides efficient API support for application developers to directly allocate and manipulate pinned memory buffers, eliminating the memory copy overhead between the pageable and pinned buffers. However, Alpha Data's private *local* memory read and write bandwidths (9.5GB/s, 8.9GB/s) are much higher than those of CAPI's shared *remote* memory access. This phenomenon offers opportunities for both platforms. As will be discussed in Section 4, if an accelerator is able to efficiently use the private memory as a "shortcut" for accessing the host memory, it will probably obtain a similar or even more effective CPU-FPGA communication bandwidth in a traditional platform like Alpha Data than in a shared-memory platform like CAPI or Xeon+FPGA.

Another remarkable phenomenon shown in Fig. 4 is the dramatic falloff of the PCIe bandwidth at the 4MB payload size. This could be the side effect of CAPI's memory coherence mechanism. CAPI shares the last-level cache (LLC) of the host CPU with the FPGA, and the data access latency varies significantly between LLC hit and miss. Therefore, one possible explanation for the falloff is that CAPI shares 2MB of the 8MB LLC with the FPGA. Payloads of a size that is not larger than 2MB can fit into LLC, resulting in a low LLC hit latency that can be well amortized by a few megabytes of data. Nevertheless, when the payload size grows to 4MB and cannot fit into LLC, the average access latency of the payload data will suddenly increase, leading to the observed falloff. With the payload size continuing to grow, this high latency is gradually amortized, and the PCIe bandwidth gradually reaches the maximum value.

3.2.3 Effective Bandwidth for Xeon+FPGA v1. The CPU-FPGA communication of Xeon+FPGA v1 involves only one step: host memory access through QPI; therefore, we just measure a set of effective read and write bandwidths for different payload sizes, as shown in Fig. 4. We can see that both the read and write bandwidths (7.0GB/s, 4.9GB/s) are much higher than the PCIe bandwidths of Alpha Data and CAPI. That is, the QPI-based CPU-FPGA integration demonstrates a higher effective bandwidth than the PCIe-based integration. However, the *remote* memory access bandwidths of Xeon+FPGA v1 are still lower than those of Alpha Data's *local* memory access. Thus, similar to CAPI, Xeon+FPGA v1 can possibly be outperformed by Alpha Data if an accelerator keeps reusing the data in the device memory as a "shortcut" for accessing the host memory.

We need to mention that Xeon+FPGA v1 provides a 64KB cache on its FPGA chip for coherency purposes [22]. Each CPU-FPGA communication will first go through this cache and then go to the host memory if a cache miss happens. Therefore, the CPU-FPGA communication of Xeon+FPGA v1 follows the classic cache access pattern. Since the bandwidth study mainly focuses on large payloads, our microbenchmarks simply flush the cache before accessing any payload to ensure all requests go through the host memory. The bandwidths illustrated in Fig. 4 are, more accurately, miss bandwidths. Section 3.3.1 discusses the cache behaviors in detail.

3.2.4 Effective Bandwidth for Xeon+FPGA v2. While the CPU-FPGA communication of Xeon+FPGA v2 involves only one step as well, Xeon+FPGA v2 allows the user accelerator to operate at different clock frequencies. Therefore, we measure the effective bandwidths of various payloads sizes at 200MHz and 400MHz. 200MHz is the frequency that is also used by Alpha Data and Xeon+FPGA v1; 400MHz is the maximal frequency supported by Xeon+FPGA v2. Note that this change does not affect the frequency of the internal logic of the communication channels, but just the interface between the channels and the user accelerator. Fig. 4 illustrates the measurement results. We can see that Xeon+FPGA v2 outperforms the aforementioned three CPU-FPGA platforms in terms of effective bandwidth (20GB/s at 400MHz, 12.8GB/s at 200MHz). This is because Xeon+FPGA v2 connects the CPU and the FPGA through three communication channels—one QPI channel and two PCIe channels—resulting in a significantly high aggregate bandwidth.

Like its first generation, Xeon+FPGA v2 also provides a 64KB on-chip cache for coherency purposes. However, this cache maintains coherence only for the QPI channel, and the two PCIe channels have no coherence properties. As a consequence, Xeon+FPGA v2 actually delivers a partially coherent memory model. We will discuss the cache behaviors of the Xeon+FPGA family in Section 3.3.1, and the coherence issue in Section 4.

3.2.5 Effective Bandwidth for F1 Instance. The Amazon EC2 F1 instance represents the state-of-the-art advance of the canonical PCIe-based platform architecture. Like the Alpha Data board, it connects the CPU with the FPGA through the PCIe channel, with private memory attached to both components. It is powered by the Xilinx SDAccel environment as well to achieve the behavior-level hardware accelerator development. Both the PCIe channel and the private DRAM are upgraded to supply higher bandwidths. However, the end-to-end bandwidth delivered to the end user is rather surprising. As illustrated in Fig. 4, the effective PCIe bandwidth of the F1 instance turns out to be even worse than that of the Alpha Data board which adopts an old-generation technique. The breakdown in Fig. 5 shows that the F1 PCIe bandwidth is twice the amount of the Alpha Data bandwidth if the buffer allocation and the memory copy overhead are not considered. This suggests that the buffer allocation and memory copy impose more overhead on the F1 instance than on the Alpha Data board. It might be due to the virtualization overhead of the F1 instance.

As mentioned before, the fact that CPU-FPGA bandwidth of Xeon+FPGA v1 lies between the PCIe bandwidth and the private device DRAM bandwidth of the Alpha Data board provides opportunities for both platforms. The F1 instance and Xeon+FPGA v2 form another (more advanced) pair of CPU-FPGA platforms that follows such a relation. We expect that this relation between a private-memory platform and a shared-memory platform will continue to exist in future CPU-FPGA platforms, and the platform suitability will depend on the characteristics of each application. This will be discussed in Section 4.

3.3 Effective Latency

3.3.1 Coherent Cache Behaviors. As described in Sections 3.2.3 and 3.2.4, the QPI channel of the Xeon+FPGA family includes a 64KB cache for coherence purposes, and the QPI-based communication thus falls into the classic cache access pattern. A cache transaction is typically depicted by its hit time and miss penalty. We follow this traditional methodology for cache study and quantify the hit time and miss latency of the Xeon+FPGA coherent cache, as shown in Table 3.

Table 3. CPU-FPGA access latency in Xeon+FPGA

Access Type	Latency (ns)
Read Hit	70
Write Hit	60
Read Miss	avg: 355
Write Miss	avg: 360

A noteworthy phenomenon is the long hit time—70ns (14 FPGA cycles) for read hit and 60ns (12 FPGA cycles) for write hit in this 64KB cache. We investigate this phenomenon by decomposing the hit time into three phases—address translation, cache access and transaction reordering—and measuring the elapsed time of each phase, as shown in Table 4.

The data demonstrate a possibly exorbitant price (up to 100% extra time) paid for address translation and transaction reordering. Worse still, the physical cache access latency is still prohibitively high—35ns (7 FPGA cycles). Given this small but long-latency cache, it is extremely hard, if not impossible, for an accelerator to harness the caching functionality.

Table 4. Hit latency breakdown in Xeon+FPGA

Access Step	Read Latency (ns)	Write Latency (ns)
Address Translation	20	20
Cache Access	35	35
Transaction Reordering	15	5

It is worth noting that the Xeon+FPGA v2 platform supports higher clock frequencies than its first generation, and thus can potentially lead to a lower cache access latency. However, this does not fundamentally change the fact that the latency of accessing the coherent cache is still much longer than that of accessing the on-chip BRAM blocks. Therefore, Xeon+FPGA v2 does not fundamentally improve the usability of the coherent cache. As discussed in Section 4, we still suggest accelerator designers sticking to the conventional FPGA design principle that explicitly manages the on-chip BRAM resource.

3.3.2 Communication Channel Latencies. We now compare the effective latencies among the PCIe transfer of Alpha Data, F1 instance and CAPI, the device memory access of Alpha Data and F1 instance, and the QPI transfer of the Xeon+FPGA family.[4] Table 5 lists the measured latencies of all five platforms for transferring a single 512-bit cache block (since all of them have the same 512-bit interface bitwidth). We can see that the QPI transfer expresses orders-of-magnitude lower latency compared to the PCIe transfer, and is even smaller than that of Alpha Data or Amazon F1's private DRAM access. This rather surprising observation is largely due to the implementation of the vendor-provided environment. In particular, Xilinx SDAccel connects the accelerator circuit to the FPGA-attached DRAM through not only the DRAM controller but also an AXI interface that is implemented on the FPGA chip. The data back and forth through the AXI interface impose a significant overhead to the effective device DRAM access latency,[5] resulting in the fact that the *local* DRAM access latency of Alpha Data is even longer than the *remote* memory access latency of Xeon+FPGA. This phenomenon implies that a QPI-based platform is preferable for applications with fine-grained CPU-FPGA interaction. In addition, we can see that CAPI's PCIe transfer latency is much lower than that of the Alpha Data board. This is because the Alpha Data board harnesses the SDAccel SDK which enables accelerator design and integration through high-level programming languages. Such a higher level of abstraction introduces an extra CPU-FPGA communication overhead in processing the high-level APIs.

Table 5. Latencies of transferring a single 512-bit cache block

Platform	Alpha Data	CAPI	Xeon+FPGA v1	Xeon+FPGA v2	Amazon EC2 F1
Latency	PCIe: 160μs DRAM: 542ns	882ns	355ns	323ns	PCIe: 127μs DRAM: 561ns

4 ANALYSIS AND INSIGHTS

Based on our quantitative studies, we now analyze how these microarchitectural characteristics can affect the performance of CPU-FPGA platforms and propose seven insights for platform users (to optimize their accelerator designs) and platform designers (to improve the hardware and software support in future CPU-FPGA platform development).

[4]For simplicity, we mainly discuss the CPU-to-FPGA read case; the observation is similar for the FPGA-to-CPU write case.
[5]While not being able to perfectly reason the long latency of the Xilinx platforms, we have confirmed with Xilinx that the phenomenon is observed by Xilinx as well, and the AXI bus is one of the major causes.

4.1 Insights for Platform Users

Insight 1: *Application developers should never use the advertised communication parameters, but measure the practically achievable parameters to estimate the CPU-FPGA platform performance.*

Experienced accelerator designers are generally aware of the data transfer overhead from the non-payload data, e.g., packet header, checksum, control packets, etc., and expect approximately 10% to 20% or even less bandwidth degradation. Quite often, this results in a significant overestimation of the end-to-end bandwidth due to the unawareness of the overhead generated by the system software stack, like the host-to-kernel memory copy discussed in this paper. As analyzed in Section 3, the effective bandwidth provided by a CPU-FPGA platform to end users is often *far* worse than the advertised value that reflects the physical limit of the communication channel. For example, the PCIe-based DMA transfer of the Alpha Data board fulfills only 20% of the 8GB/s bandwidth of the PCIe Gen3 x8 channel; the Amazon F1 instance that adopts a more advanced data communication technique delivers an even worse effective bandwidth to the end user. Evaluating a CPU-FPGA platform using these advertised values will probably result in a significant overestimation of the platform performance. Worse still, the relatively low effective bandwidth is not always achievable. In fact, the communication bandwidth for a small payload is up to two orders of magnitude smaller than the maximum achievable effective bandwidth. A specific application may not always be able to supply each communication transaction with a sufficiently large payload to reach a high bandwidth. Platform users need to consider this issue as well in platform selection.

Insight 2: *In terms of effective bandwidth, both the private-memory and shared-memory platforms have opportunities to outperform each other. The key metric is the device memory reuse ratio* r.

Bounded by the low-bandwidth PCIe-based DMA transfer, the Alpha Data board generally reaches a lower CPU-FPGA effective bandwidth than that of a shared-memory platform like CAPI or Xeon+FPGA v1. The higher private memory bandwidth, however, does provide opportunities for a private-memory platform to perform better in some cases. For example, given 1GB input data sent to the device memory through PCIe, if the FPGA accelerator iteratively reads the data for a large number of times, then the low DMA bandwidth will be amortized by the high private memory bandwidth, and the effective CPU-FPGA bandwidth will be nearly equal to the private memory bandwidth, which is higher than that of the shared-memory platform. Therefore, the data reuse of FPGA's private DRAM determines the effective CPU-FPGA bandwidth of a private-memory platform, and whether it can achieve higher effective bandwidth than a shared-memory platform.

Quantitatively, we define the device memory reuse ratio, r, as:

$$r = \frac{\sum_{dev} S_{dev}}{\sum_{dma} S_{dma}}$$

where $\sum_{dev} S_{dev}$ denotes the aggregate data size of all device memory accesses, and $\sum_{dma} S_{dma}$ denotes the aggregate data size of all DMA transactions between the host and the device memory.

Then, the effective CPU-FPGA bandwidth for a private-memory platform, $bw_{cpu-fpga}$, can be defined as:

$$bw_{cpu-fpga} = \frac{1}{\frac{1}{r * bw_{dma}} + \frac{1}{bw_{dev}}}$$

where bw_{dma} and bw_{dev} denote the bandwidths of the DMA transfer and the device memory access, respectively.

The above formula suggests that larger r leads to higher effective CPU-FPGA bandwidth. It is worth noting that since the FPGA on-chip BRAM data reuse is typically important for FPGA design optimization, the above finding suggests that accelerator designers using a private-memory platform need to consider both on-chip BRAM data reuse and off-chip DRAM data reuse. Moreover, by comparing this effective CPU-FPGA bandwidth of a private-memory platform to the DRAM bandwidth of a shared-memory platform, we can get a threshold of device memory reuse ratio, $r_{threshold}$. If the r value of an application is larger than $r_{threshold}$, the private-memory platform will achieve a higher bandwidth; and vice versa. This could serve as an initial guideline for application developers to choose the appropriate platform for a specific application. One example is the logistic regression application whose computation kernel is a series of matrix-vector multiplication operations that iterate over the same input matrix. Our matrix-vector multiplication case study in Section 5.2 demonstrates that the Alpha Data board starts to outperform Xeon+FPGA v1 when the iteration number grows beyond 7, which is the value of $r_{threshold}$ for Alpha Data and Xeon+FPGA v1. Also, this phenomenon continues to exist between the next-generation platforms, i.e., the Amazon EC2 F1 instance and Xeon+FPGA v2. This suggests that the proposed device memory reuse ratio is not merely applicable to the evaluated platforms in this paper, but also provides guidance to the selection of the private-memory versus shared-memory CPU-FPGA platforms across generations.

Fundamentally, the device memory reuse ratio quantifies the trade-off between private-memory and shared-memory platforms based on the following two observations. First, local memory access is usually faster than remote memory access. Using the same technology, the private-memory platform achieves a higher device memory access bandwidth compared to the shared-memory platform which retrieves data from the CPU-attached memory. Second, the end-to-end CPU-FPGA data transfer routine of the shared-memory platform is a subset of that of the private-memory platform. Specifically, the routine of the private-memory platform contains data transfers 1) from CPU-attached memory to FPGA, 2) from FPGA to FPGA-attached memory, and 3) from FPGA-attached memory to FPGA; whereas the shared-memory platform performs only the first step. Since these observations are not likely to change over time, we expect that the trade-off between these two types of platforms will continue to exist, and the device memory reuse ratio will remain a critical parameter.

Insight 3: *In terms of effective latency, the shared-memory platform generally outperforms the private-memory platform, and the QPI-based platform outperforms the PCIe-based platform.*

As shown in Table 5, the shared-memory platform generally achieves a lower communication latency than the private-memory platform with the same communication technology (CAPI vs. Alpha Data). This is because the private-memory platform first caches the data in its device memory, and then allows the FPGA to access the data; this results in a longer communication routine. This advantage, together with an easier programming model, motivates the new trend of CPU-FPGA platforms with a PCIe connection and coherent shared memory, such as CAPI and CCIx.

Meanwhile, compared to the PCIe-based platform, the QPI-based platform brings the FPGA closer to the CPU, leading to a lower communication latency. Therefore, a QPI-based, shared-memory platform is preferred for latency-sensitive applications, especially those that require frequent (random) fine-grained CPU-FPGA communication. Some examples like high-frequency trading (HFT), online transaction processing (OLTP), or autonomous driving might benefit from the low communication latency of the QPI channel. Compared to Xeon+FPGA systems, the major drive of the PCIe-based shared memory system is its extensibility for more FPGA boards in large-scale systems.

Insight 4: *CPU-FPGA communication is critical to some applications, but not all. The key metric is the computation-to-communication ratio* (CTC).

Double buffering and dataflow are well-used techniques in accelerator design optimizations. Such techniques can realize a coarse-grained data processing pipeline by overlapping the computation and data communication processes. As a result, the performance of the FPGA accelerator is generally bounded by the coarse-grained pipeline stage that consumes more time. Based on this criterion, FPGA accelerators can be roughly categorized into two classes: 1) computation-bounded ones where the computation stage takes a longer time, and 2) communication-bounded ones where the communication stage takes longer time.

If an accelerator is communication-bounded, then a better CPU-FPGA communication stack will greatly improve its overall performance. As demonstrated in our matrix-vector multiplication case study (Section 5.2), the overall performance of the FPGA accelerator is determined by the effective CPU-FPGA communication bandwidth that the platform can provide. In this case, the high-bandwidth F1 instance and Xeon+FPGA v2 platform are preferred. On the other hand, if an accelerator is computation-bounded, then switching to another platform with a better communication stack does not make a considerable difference. As a computation-bounded accelerator example, the matrix multiplication accelerator performs almost the same in Alpha Data, CAPI, the Xeon+FPGA family, and the F1 instance when scaling to the same 200MHz frequency. In this case, the traditional PCIe-based private-memory platform may be preferred because of its good compatibility. One may even prefer not to choose a platform with the best CPU-FPGA communication technology for cost efficiency. Application developers should find out whether the application to accelerate is compute-intensive or communication-intensive in order to select the appropriate platform.

Quantitatively, we use the computation-to-communication (C2C) ratio [32] (which is also named "memory intensity" in [28]) to justify whether a computation kernel is computation/communication bounded. Specifically, the C2C ratio is defined as the division of the computation throughput and the data transfer throughput:

$$C2C\ ratio = \frac{Throughput_{compute}}{Throughput_{transfer}}$$

The computation throughput is referred to as the speed of processing a certain size of input for a given FPGA accelerator; the data transfer throughput is referred to as the speed of transferring this certain size of input into or out of the FPGA fabric. When the C2C ratio of a kernel is above 1, this kernel is then computation bounded; otherwise, it is communication bounded. In general, the data transfer throughput is linearly proportional to the input size. Therefore, a computation kernel with super-linear time complexity, such as matrix multiplication, is computation bounded. Meanwhile, computation kernels, like matrix-vector multiplication that is of linear time complexity, are often bounded by the CPU-FPGA communication. For computation bounded kernels, the CPU-FPGA communication bandwidth is not the performance bottleneck, so the accelerator designers do not need to chase for high-end communication interfaces. On the other hand, the CPU-FPGA communication is critical to communication bounded kernels with C2C ratio less than 1, and the efficiency of the communication interface is then a key factor in platform selection.

Insight 5: *CPU-FPGA memory coherence is promising, but impractical to be used in accelerator design, at least for now.*

The newly-announced CPU-FPGA platforms, including CAPI, CCIx and the Xeon+FPGA family, attempt to provide memory coherence support between the CPU and the FPGA—either through PCIe or QPI. Their implementation methodology is similar: constructing a coherent cache on the FPGA fabric to realize the classic snoopy protocol with the last-level cache of the host CPU. However, although the FPGA fabric supplies a few megabytes of on-chip BRAM blocks, only 64KB (the Xeon+FPGA family) or 128KB (CAPI) of them are organized into the coherent cache. That is, these platforms maintain memory coherence for less than 5% of the on-chip memory space, leaving the majority as scratchpads of which the coherence needs to be maintained by application developers. Although users may choose to ignore the 95% scratchpads and store data on chip only through the coherent cache to obtain transparent coherence maintenance, this approach is apparently inefficient. For one thing, the coherent cache has a much longer access latency than that of the scratchpads. Also, the coherent cache provides much less parallel access capability compared to the scratchpads that can potentially feed thousands of data per cycle. As a consequence, application developers may still have to stick to the conventional FPGA accelerator design principle to explicitly manage the coherence of the scratchpad data.

While the current implementation of the CPU-FPGA memory coherence seems to be impractical due to the aforementioned prohibitively high overhead, the methodology does create a great potential for reducing FPGA programming effort. The coherent cache is particularly beneficial for computation kernels with unforeseeable memory access patterns, such as hashing. As will be discussed in Insight 7, implementing the coherent cache on the FPGA fabric considerably restricts its capacity, latency, and bandwidth. If the cache is implemented as a dedicated ASIC memory block in future FPGA platforms, application developers could harness the power of the cache coherence.

4.2 Insights for Platform Designers

Insight 6: *There still exists a large room for improvement in order to bridge the gap between the practically achieved bandwidth and the physical limit of the communication channel.*

For example, none of Alpha Data, CAPI or Amazon F1 instance fulfill the 8GB/s PCIe bandwidth, even without considering the overhead of pinned memory allocation and pageable-pinned memory copy. Meanwhile, it proves to be a good alternative to alleviate the communication overhead by allowing direct pageable data transfer through PCIe, which is realized in the CAPI platform. Another alternative is to offer end users the capability to directly manipulate pinned memory space. For example, both the Xeon+FPGA family and unified virtual addressing (CUDA for GPU) provide efficient and flexible API support to allow software developers to operate on allocated pinned memory arrays or objects just like those allocated by `malloc/new` [21]. Nevertheless, these solutions result in "fragmented" payloads, i.e., the payload data may be stored in discrete memory pages, causing reduced communication bandwidth.

Another alternative optimization is to form the CPU-FPGA communication stack into a coarse-grained pipeline, like the CUDA streams technique in GPUs. This may slightly increase the communication latency for an individual payload, but could significantly boost the throughput of CPU-FPGA communication for concurrent transactions.

Both approaches should solve the problem, and the solution has been verified by two of our follow-up studies. Guided by this insight, [26] proposes a new programming environment for streaming applications that achieves a much higher effective bandwidth in the Amazon F1 instance; [13] proposes a deep pipeline stack to overlap the communication and computation steps. More discussions are provided in Section 5.3.

Insight 7: *The coherent cache design could be greatly improved.*

The coherent cache of the recently-announced CPU-FPGA platforms aims to provide the classic functionalities of CPU caches: data caching and memory coherence that are transparent to programmers. However, the long latency and small capacity make this component impractical to be used efficiently in FPGA accelerator design.

One important reason for such a long-latency, small-capacity design is that the coherent cache is implemented on the FPGA fabric. Therefore, compared to the CPU cache counterpart, the FPGA-based coherent cache has a much lower frequency and thus a much worse performance. One possible approach to address this issue is to move the coherent cache module out of the FPGA fabric as a hard ASIC circuit instead. This could potentially reduce the difference between the FPGA's cache latency and the scratchpad latency and also enlarge the cache capacity. As a result, cache can be more efficiently utilized in FPGA accelerator designs.

Another important reason is that the cache structure generally has a very limited number of data access ports. In contrast, the BRAM blocks on the FPGA fabric can be partitioned to supply thousands of data in parallel. To exploit such a parallel data supply, it is also a common practice to assign a dedicated BRAM buffer for multiple processing elements in FPGA accelerator design (e.g., [9]). Since massive parallelism is a widely adopted way for FPGA acceleration, the future cache design may also need to take this into consideration; e.g., a distributed, many-port cache design might be preferred to a centralized, single-port design.

5 CASE STUDIES

To demonstrate the usefulness of our proposed insights, we conduct two case studies that utilize those insights (for platform users) to optimize the accelerator designs: matrix multiplication and matrix-vector multiplication. These two applications share the same basic operations, floating-point addition and multiplication, but belong to different categories. The former is computation-bounded, and the latter is communication-bounded. We use them to demonstrate how to leverage our two proposed metrics—device memory reuse ratio r and the computation-to-communication ratio CTC—to choose the right platform.

The basic settings of the two cases are almost identical. The dimension of the matrices is 4096×4096, and the dimension of the vectors is 4096. The input data are initially stored in the CPU-attached memory, and the output data will be stored back to the CPU-attached memory. The same systolic-array-based matrix multiplication accelerator design presented in [19] is implemented on all five platforms, and the matrix-vector multiplication design has a similar architecture. For comparison purposes, the computation architecture has been almost identically designed on all platforms; thus the capacity of the FPGA fabric has no effect on our case evaluation.

5.1 Matrix Multiplication

Given two $N \times N$ floating-point matrices as input and a $N \times N$ matrix as output, the computation time complexity ($O(N^3)$) is higher than that of data communication ($O(N^2)$), resulting in the algorithm being computation-bounded. Fig. 6 illustrates the accelerator performances on the five platforms. We can see that the performance is proportionate to the frequency of the accelerator design. In detail, since Alpha Data and Xeon+FPGA v1 have the same frequency of operation, 200MHz, the same accelerator design leads to almost identical performance. For the Xeon+FPGA v2 platform that supports multiple frequencies, we configure the accelerator under both 200MHz and 273MHz to 1) make a fair comparison with other platforms, 2) demonstrate the impact of frequency. The results show that the 200MHz design on the Xeon+FPGA v2 platform does not achieve any superiority over the ones on Alpha Data and Xeon+FPGA v1, even though it has a much higher CPU-FPGA communication bandwidth. Meanwhile, the CAPI platform and the F1 instance with the 250MHz frequency delivers a better performance over the 200MHz designs on the other platforms.

The fundamental reason for such results is that the matrix multiplication design is computation-bounded. The data communication is therefore completely overlapped by the accelerator computation, which is determined by the frequency of operation given the same circuit design. This verifies our insight that a carefully designed accelerator may diminish the impact of CPU-FPGA communication if it is computation-bounded. Application developers could thus focus on the other factors in platform selection, like compatibility, cost, etc.

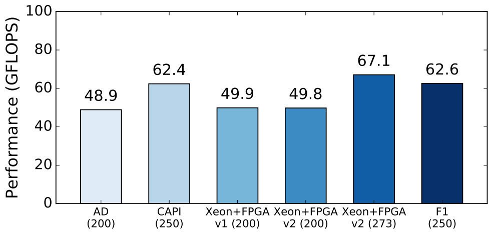


Fig. 6. Matrix multiplication kernel execution time

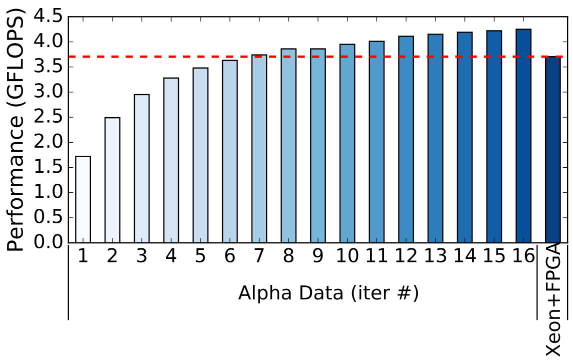


Fig. 7. Matrix-vector multiplication kernel execution time

5.2 Matrix-Vector Multiplication

Given an $N \times N$ floating-point matrix and an N-dimension vector as input, and an N-dimension vector as output, the computation time complexity ($O(N^2)$) is the same as that of the data communication ($O(N^2)$). The algorithm is generally communication-bounded. To avoid the interference of the accelerator design and the frequency of operation, we use the same design on Alpha Data and Xeon+FPGA under the same 200MHz frequency. In order to demonstrate the impact of the device memory reuse ratio, we iteratively perform the matrix-vector multiplication with the same matrix, but the updated vector that is generated from the last iteration. This is the typical computation pattern of the widely used logistic regression application.

Fig. 7 illustrates the performances on different platforms with various iteration numbers. Note that the accelerator performance on Xeon+FPGA is not affected by the iteration number, so we just show one value for it. We can see that the accelerator performance on the Alpha Data board improves with the increase of the iteration number, because the one-time, low-bandwidth PCIe transfer is amortized by the high-bandwidth device memory access. After the iteration number (device memory reuse ratio r) grows beyond 7, the value of $r_{threshold}$ between Alpha Data and Xeon+FPGA, the Alpha Data board starts to outperform Xeon+FPGA.

5.3 Optimization of Communication Overhead

While the case studies demonstrate how the insights work, there are a set of follow-up studies that demonstrate the effectiveness of the insights in acceleration design and platform optimization. In Section 3.2, we claimed that the effective host-to-FPGA bandwidth will be reduced by various factors such as PCIe packet header overhead, host buffer allocation, and host memory copy. In addition to the analysis we provided, this claim may also be verified by optimizing away the aforementioned overheads and measuring the performance improvement. We summarize them as follows.

In ST-Accel [26], a new communication library is proposed to provide direct connection between IO and FPGA, therefore bypassing host DRAM and FPGA DRAM data copy. This has improved the effective host-to-FPGA bandwidth from 2.59 GB/s to 11.89 GB/s. In [13], the data transfer from FPGA to JVM is pipelined with FPGA computation and host-JVM communication, effectively hiding the low PCIE bandwidth. This improves the throughput by 4.9× on average. The work in [8] provides another case study on DNA sequencing where the host-to-FPGA bandwidth is improved by batch processing and sharing the FPGA accelerator among multiple threads. These techniques reduce the communication overhead from 1000× of the computation time down to only 16% of the overall execution time.

6 CONCLUSION AND FUTURE WORK

To the best of our knowledge, this is the first comprehensive study that aims to evaluate and analyze the microarchitectural characteristics of state-of-the-art CPU-FPGA platforms in depth. The paper covers all the latest-announced, shared-memory platforms, as well as the traditional private-memory Alpha Data broad and the Amazon EC2 F1 instance, with detailed data published (most of which not available from public datasheets). We found that the advertised communication parameters are often too ideal to be delivered to end users in practice, and suggest application developers avoiding overestimation of the platform performance by considering the effective bandwidth and the communication payload. Moreover, we demonstrate that the communication-bounded accelerators can be significantly affected by different platform implementations, and propose the device memory reuse ratio as a metric to quantify the boundary of platform selection between a private-memory platform and a shared memory platform. Also, we demonstrate that the CPU-FPGA communication may not matter for computation-bounded applications where the data movement can be overlapped by the accelerator computation, and propose to use the computation-to-communication ratio *CTC* to measure it. In addition, we point out that the transparent data caching and memory coherence functionalities may be impractical in the current platforms because of the low-capacity and high-latency cache design.

We believe these results and insights can aid platform users in choosing the best platform for a given application to accelerate, and facilitate the maturity of CPU-FPGA platforms. To help the community measure other platforms, we have also released our microbenchmarks at http://vast.cs.ucla.edu/ubench-cpu-fpga.

ACKNOWLEDGMENTS

This research is supported in part by Intel and the National Science Foundation (NSF) Joint Research Center on Computer Assisted Programming for Heterogeneous Architectures (CAPA) (CCF-1723773); NSF/Intel InTrans Program (CCF-1436827); CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA (GI18518.156870); A Community Effort to Translate Protein Data to Knowledge: An Integrated Platform (U54GM114833-01); UCLA Institute for Digital Research and Education Postdoc Fellowship; Fujitsu and Huawei under the CDSC industrial partnership; and C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA. We thank Amazon for the AWS F1 credit, Intel and Xilinx for the hardware donation, and Janice Wheeler for proofreading this paper.

REFERENCES

[1] 2016. Intel to Start Shipping Xeons With FPGAs in Early 2016. (2016). http://www.eweek.com/servers/intel-to-start-shipping-xeons-with-fpgas-in-early-2016.html

[2] 2017. Amazon EC2 F1 Instance. (2017). https://aws.amazon.com/ec2/instance-types/f1/

[3] 2017. SDAccel Development Environment. (2017). http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[4] 2018. Cache Coherent Interconnect for Accelerators. (2018). https://www.ccixconsortium.com/

[5] Brad Brech, Juan Rubio, and Michael Hollinger. 2015. *IBM Data Engine for NoSQL - Power Systems Edition.* Technical Report. IBM Systems Group.

[6] Tony M Brewer. 2010. Instruction set innovations for the Convey HC-1 computer. *IEEE Micro* 2 (2010), 70–79.

[7] Nanchini Chandramoorthy, Giuseppe Tagliavini, Kevin Irick, Antonio Pullini, Siddharth Advani, Sulaiman Al Habsi, Matthew Cotter, John Sampson, Vijaykrishnan Narayanan, and Luca Benini. 2015. Exploring architectural heterogeneity in intelligent vision systems. In *HPCA-21.*

[8] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. 2016. When Apache Spark meets FPGAs: A case study for next-generation DNA sequencing acceleration. In *HotCloud.*

[9] Young-kyu Choi and Jason Cong. 2016. Acceleration of EM-based 3D CT reconstruction using FPGA. *IEEE Transactions on Biomedical Circuits and Systems* 10, 3 (2016), 754–767.

[10] Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2016. A quantitative analysis on microarchitectures of modern CPU-FPGA platforms. In *DAC-53.*

[11] Jason Cong, Zhenman Fang, Yuchen Hao, and Glenn Reinman. 2017. Supporting Address Translation for Accelerator-Centric Architectures. In *HPCA-23.*

[12] Jason Cong, Mohammad Ali Ghodrat, Michael Gill, Beayna Grigorian, Hui Huang, and Glenn Reinman. 2013. Composable accelerator-rich microprocessor enhanced for adaptivity and longevity. In *ISLPED.*

[13] Jason Cong, Peng Wei, and Cody Hao Yu. 2018. From JVM to FPGA: Bridging abstraction hierarchy via optimized deep pipelining. In *HotCloud.*

[14] Shane Cook. 2012. *CUDA programming: a developer's guide to parallel computing with GPUs.* Newnes.

[15] Emilio G Cota, Paolo Mantovani, Giuseppe Di Guglielmo, and Luca P Carloni. 2015. An analysis of accelerator coupling in heterogeneous architectures. In *DAC-52.*

[16] Zhenman Fang, Sanyam Mehta, Pen-Chung Yew, Antonia Zhai, James Greensky, Gautham Beeraka, and Binyu Zang. 2015. Measuring microarchitectural details of multi-and many-core memory systems through microbenchmarking. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 4 (2015), 55.

[17] IBM 2015. *Coherent Accelerator Processor Interface UserâĂŹs Manual Xilinx Edition.* IBM. Rev. 1.1.

[18] Intel 2016. *BDW+FPGA Beta Release 5.0.3 Core Cache Interface (CCI-P) Interface Specification.* Intel. Rev. 1.0.

[19] J. Jang, S. Choi, and V. Prasanna. 2005. Energy-and time-efficient matrix multiplication on FPGAs. *IEEE TVLSI* 13, 11 (2005), 1305–1319.

[20] Jason Lawley. 2014. *Understanding Performance of PCI Express Systems.* Xilinx. Rev. 1.2.

[21] NVIDIA 2009. *NVIDIA's Next Generation CUDA Compute Architecture: FERMI.* NVIDIA. Rev. 1.1.

[22] Neal Oliver, Rahul R Sharma, Stephen Chang, Bhushan Chitlur, Elkin Garcia, Joseph Grecco, Aaron Grier, Nelson Ijih, Yaping Liu, Pratik Marolia, and others. 2011. A reconfigurable computing system based on a cache-coherent fabric. In *ReConFig.*

[23] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. 2015. Toward accelerating deep learning at scale using specialized hardware in the datacenter. In *Hot Chips.*

[24] Andrew Putnam, Adrian M Caulfield, Eric S Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, and others. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *ISCA-41.*

[25] Phil Rogers. 2013. Heterogeneous system architecture overview. In *Hot Chips.*

[26] Zhenyuan Ruan, Tong He, Bojie Li, Peipei Zhou, and Jason Cong. 2018. ST-Accel: A high-level programming platform for streaming applications on FPGA. In *FCCM.*

[27] J Stuecheli, Bart Blaner, CR Johns, and MS Siegel. 2015. CAPI: A Coherent Accelerator Processor Interface. *IBM Journal of Research and Development* 59, 1 (2015), 7–1.

[28] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2009), 65–76.

[29] Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos. 2010. Demystifying GPU microarchitecture through microbenchmarking. In *ISPASS.*

[30] Xilinx 2017. *ADM-PCIE-7V3 Datasheet.* Xilinx. Rev. 1.3.

[31] Serif Yesil, Muhammet Mustafa Ozdal, Taemin Kim, Andrey Ayupov, Steven Burns, and Ozcan Ozturk. 2015. Hardware Accelerator Design for Data Centers. In *ICCAD.*

[32] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '15)*. 161–170.