A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms

Young-kyu Choi Jason Cong Zhenman Fang
Yuchen Hao Glenn Reinman Peng Wei

Center for Domain-Specific Computing, University of California, Los Angeles
{ykchoi, cong, zhenman, haoyc, reinman, peng.wei.prc}@cs.ucla.edu

ABSTRACT

CPU-FPGA heterogeneous acceleration platforms have shown great potential for continued performance and energy efficiency improvement for modern data centers, and have captured great attention from both academia and industry. However, it is nontrivial for users to choose the right platform among various PCIe and QPI based CPU-FPGA platforms from different vendors. This paper aims to find out what microarchitectural characteristics affect the performance, and how. We conduct our quantitative comparison and in-depth analysis on two representative platforms: QPI-based Intel-Altera HARP with coherent shared memory, and PCIe-based Alpha Data board with private device memory. We provide multiple insights for both application developers and platform designers.

1. INTRODUCTION

In today's data center design, 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 to be one of the most promising directions, since FPGAs provide low power and high energy efficiency, and can be reprogrammed to accelerate different applications. For example, Microsoft has designed a customized FPGA board called Catapult and deployed it in its data center [16], which improved the ranking throughput of the Bing search engine by 2x. In other words, the number of servers can be reduced by 2x with each new CPU-FPGA server consuming only 10% more power.

Motivated by FPGAs' high energy efficiency and reprogrammability, industry vendors, such as Microsoft, Intel/Altera, Xilinx, IBM and Convey, are providing various ways to integrate high-performance FPGAs into data center servers. We have classified modern CPU-FPGA acceleration 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 PCIe, with both components equipped with private memory. Many FPGA boards built on top of Xilinx or Altera 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 [1] with Xilinx FPGA fabric that can leverage the Xilinx SDAccel development environment [2] to support efficient accelerator design using high-level programming languages, including C/C++ and OpenCL. Moreover, vendors like IBM tend to support a PCIe connection with a coherent shared memory for easier programming. For example, IBM has been developing the Coherent Accelerator Processor Interface (CAPI) on POWER8 [19] for such an integration, and has used this platform in the IBM data engine for NoSQL [3]. More recently, closer 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 [4] and, the latest Intel-Altera Heterogeneous Architecture Research Platform (HARP) [13] that targets data centers.

Table 1: Classification of modern CPU-FPGA platforms

	Separate Private Memory	Coherent Shared Memory
PCIe Peripheral Interconnect	Alpha Data board [2], Microsoft Catapult [16]	IBM CAPI [19]
Processor Interconnect	N/A	Intel-Altera HARP (QPI) [13], Convey HC-1 (FSB) [4]

The evolvement of various CPU-FPGA platforms brings up two challenging questions: 1) which platform should we choose to gain better performance and energy efficiency? and 2) how can we design an optimal accelerator on a given platform? There are numerous factors that can affect the choice and optimization, 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 newly announced platforms like HARP[1]. More importantly, those 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 aid CPU-FPGA platform users to accurately predict (and optimize) the performance of a candidate accelerator design on a platform, 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 what microarchitectural characteristics affect the performance of modern CPU-FPGA platforms, and evaluate how they will affect that performance. We conduct our quantitative comparison on two representative modern CPU-FPGA platforms to cover both integration dimensions, i.e., PCIe-based vs. QPI-based, and private vs. shared memory model. One is the recently announced QPI-based HARP with coherent shared memory, and the other is the commonly used PCIe-based Alpha Data with separate private memory. We quantify each platform's CPU-FPGA communication latency and bandwidth with in-depth analysis, and

[1]For simplicity, in the rest of this paper, we will use HARP for the Intel-Altera HARP platform, and Alpha Data for the Alpha Data board integrated with a Xeon CPU.

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2897972

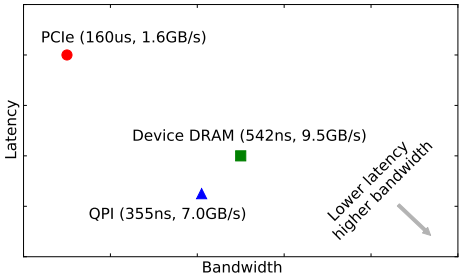


Figure 1: Summary of CPU-FPGA communication bandwidth and latency (not to scale) for PCIe-based and QPI-based platforms

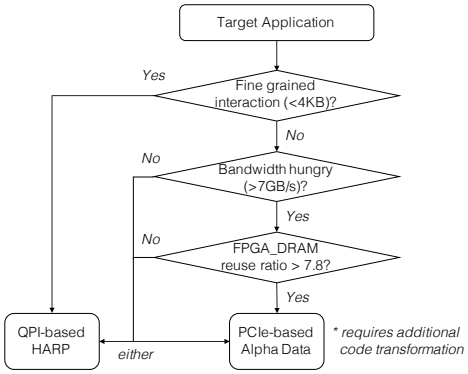


Figure 2: Decision tree to choose the right CPU-FPGA platform

derive insights for both platform users and designers. To demonstrate how theses insights can be leveraged to optimize accelerator designs, we further conduct two case studies in real applications like matrix multiplication and high-throughput sequencing [18].

In summary, this paper makes the following contributions.

1. The first quantitative study on the microarchitectures of two representative modern PCIe and QPI based CPU-FPGA acceleration platforms, with results summarized in Fig. 1.
2. An in-depth analysis of the big gap between advertised and practically achievable performance (Section 3), and five insights for both application developers to improve accelerator designs and platform designers to improve platform support (Section 4).
3. A decision tree (Fig. 2) for easy choice of the platforms: 1) QPI-based HARP has lower latency, and is better for applications with fine-grained CPU-FPGA interaction; 2) nevertheless, PCIe-based Alpha Data can catch up for coarse-grained interaction with additional code transformations to hide the latency; 3) furthermore, PCIe-based Alpha Data is better for bandwidth-hungry applications with high FPGA_DRAM reuse ratio.

2. BACKGROUND

A high-performance interconnect between 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 interconnect. 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. 3(a)). To interface with external network and 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 [1] in the SDAccel development environment [2]. 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.

Intel HARP 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. 3(b), an Intel QPI IP that contains a 64KB cache is necessary to handle coherent communication with the processor, and a system protocol layer (SPL) is introduced to provide address translation and request reordering to user-defined AFUs with virtual addressing. Specifically, a page table of 1024 entries, each associated with a 2MB page (2GB in total), is implemented in SPL, which is loaded by the kernel driver. 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.

2.2 CPU-FPGA Memory Models

Accelerators with physical addressing effectively adopt a separate private address space paradigm (Fig. 3(c)). Data shared between the host and device must be allocated in both memories, 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 private address spaces cause data replication, resulting in extra latency and overhead. In this paper the evaluated Alpha Data platform falls into this category.

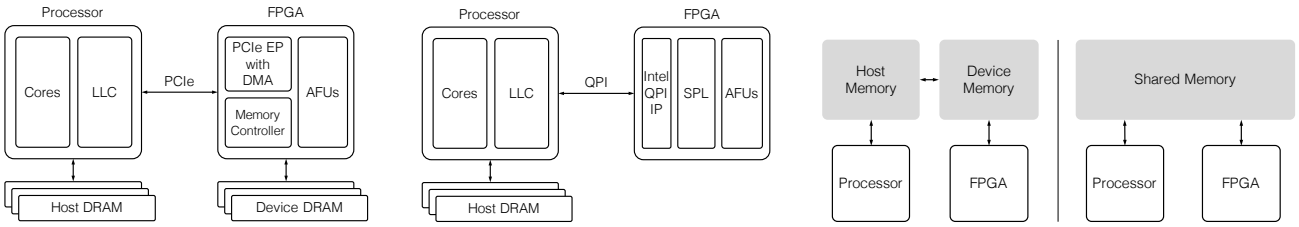
With tighter FPGA integration to the processor, the ideal case would be to have a unified shared address space between the CPU and FPGA. In this case (Fig. 3(c)), instead of allocating two copies in both host and device memories, only a single allocation is necessary. This has a variety of benefits, including the elimination of explicit data copies, pointer semantics and increased performance of fine-grained memory accesses. HARP provides some of the convenience of a unified shared address space. It enables *zero-copy* using pinned host memory which allows a device to directly access and process data on that memory location. However, developers must rely on special APIs, rather than normal C or C++ allocation (e.g., `malloc/new`), to allocate pinned memory space. Also, pinned memory is much more expensive to allocate and de-allocate, which can lead to poor performance on small transfer sizes. In general, full support for shared virtual address requires architecture and operating system support for address translation on FPGAs, which has yet to be fully explored on CPU-FPGA platforms.

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. [21] 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. [5] 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. [9] specifically analyzed the integration and interaction of TCAs and LCAs at different levels in the memory hierarchy. CAMEL [7] featured reconfigurable fabric to improve the utilization and longevity of on-chip accelerators. All 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 virtual shared memory. NVIDIA introduced "unified virtual addressing" beginning with the Fermi architecture [15]. The Heterogeneous System Ar-



(a) PCIe-based CPU-FPGA platform (b) QPI-based CPU-FPGA platform (c) Developer view of private and shared memory

Figure 3: CPU-FPGA system architectures and memory models

chitecture Foundation announced heterogeneous Uniform Memory Accesses (hUMA) that will implement the shared virtual address paradigm in future heterogeneous processors [17]. Virtual shared memory support for CPU-FPGA platforms has been explored in [19, 13]. CAPI [19] enables the FPGA to participate as a cache coherent peer to general-purpose cores. However, the underlying PCIe bus favors bandwidth-oriented applications instead of applications with low-latency and fine-grained interactions. This paper covers both the separate private memory model (Alpha Data) and virtual shared memory model (HARP). In our future work, we will further evaluate the CAPI platform as well.

Third, there is also numerous work that evaluates modern CPU and GPU microarchitectures. For example, Fang et al. [10] evaluated the memory system microarchitectures on commodity multicore and many-core CPUs. Wong et al. [20] 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 these 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 two representative platforms: HARP[2] and Alpha Data.

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 2015.1.5 [2] for Alpha Data and Intel AALSDK 4.1.7 [12] for HARP. Each microbenchmark consists of two parts: a host program and a computational kernel. Following each platform's typical programming model, we use the C language to write the host programs for both platforms, and describe the kernel design using OpenCL for Alpha Data and Verilog HDL for HARP.

The hardware configurations of Alpha Data and HARP in our study are listed in Table 2.

Table 2: Configurations of Alpha Data and HARP

Platform	Alpha Data	HARP
Host CPU	Xeon E5-2620v3@2.40GHz	Xeon E5-2680v2@2.80GHz
Host Memory	64GB DDR3-1600	96GB DDR3-1600
FPGA Fabric	Xilinx Virtex 7@200MHz	Altera Stratix V@200MHz
CPU ↔ FPGA	PCIe Gen3 x8, 8GB/s	Intel QPI, 8GT/s
Device Memory	16GB DDR3-1600	N/A

3.2 Effective Bandwidth

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

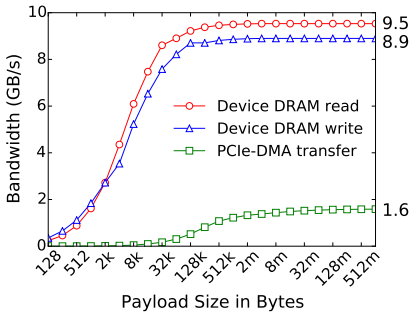


Figure 4: Bandwidth of PCIe-DMA and device DRAM in Alpha Data

3.2.1 *Effective Bandwidth for Alpha Data*

Alpha Data's CPU-FPGA communication includes two processes: PCIe-DMA transfer between host memory and device (FPGA) memory, and device memory access. We measure the effective bandwidths with various payload sizes for both processes, as shown 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[3]); 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 away from being fulfilled.

The first reason is that there is non-payload data overhead for the useful payload transfer [11]. In a PCIe transfer, a payload is split into small packages, each one equipped with a header. Along with the payload packages, there are also a large number of packages for control purposes transferred through PCIe.

Another important reason is that a PCIe-DMA transaction involves not only PCIe transfer, but also host buffer allocation and host memory copy [8]. The host memory stores user data in a pageable (unpinned) space from which the FPGA cannot directly retrieve data. A page-locked (pinned) memory buffer then 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 deceases the PCIe-DMA bandwidth.

Next, we quantitatively evaluate the huge 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 [11].
2. Nevertheless, the highest achieved effective PCIe-DMA bandwidth without buffer allocation and host memory copy is only 3.8GB/s. The possible reason is that the device-side DMA FPGA IP which connects the device memory to the PCIe bus

[3]If not specifically indicated, the bandwidth appearing in the rest of this paper refers to the maximum achievable bandwidth.

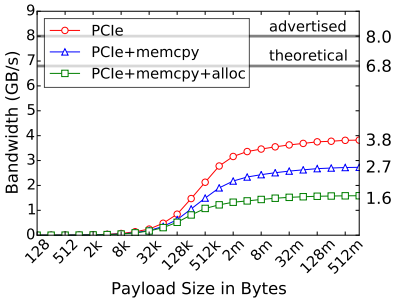


Figure 5: Alpha Data PCIe-DMA bandwidth

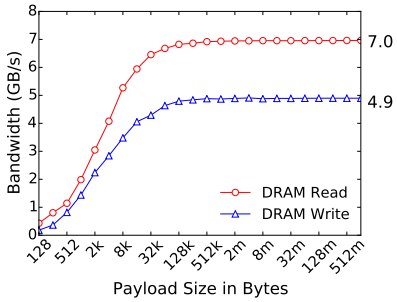


Figure 6: HARP QPI bandwidth

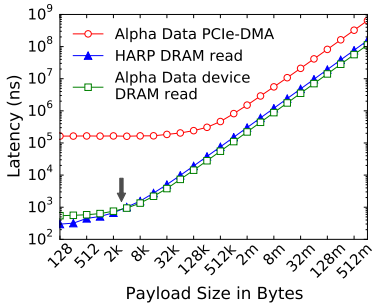


Figure 7: Latency in HARP and Alpha Data

might prevent the PCIe bandwidth from being fully exploited, thus further decreasing the PCIe-DMA bandwidth in practice.

3. The host memory copy overhead 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.

3.2.2 *Effective Bandwidth for HARP*

HARP's CPU-FPGA communication 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. 6. We can see that both the read and write bandwidths of HARP (7.0GB/s, 4.9GB/s) are much higher than the PCIe-DMA bandwidth of Alpha Data (1.6GB/s). Therefore, the QPI-based platform does demonstrate a higher effective bandwidth than the PCIe-based platform in terms of directly retrieving a payload from the host memory to the FPGA. However, Alpha Data's private *local* memory read and write bandwidths (9.5GB/s, 8.9GB/s) are higher than those of HARP's shared *remote* memory access (7.0GB/s, 4.9GB/s). This phenomenon indicates an opportunity for a platform with separate private memory to outperform another platform with coherent shared memory. In other words, if an accelerator is able to smartly use the private memory as a shortcut for accessing the host memory, it will probably obtain a similar or even higher effective bandwidth in a separate private memory platform than in a coherent shared memory platform (see insight 1 in Section 4).

We need to mention that HARP's FPGA contains a 64KB cache for coherency purposes [13]. Each CPU-FPGA communication will first go through this cache and then go to the host memory if a cache miss happens. Therefore, HARP's CPU-FPGA communication 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. 6 are, more accurately, miss bandwidths. Section 3.3 discusses the cache behaviors in detail.

3.3 Effective Latency

As described in Section 3.2.2, HARP's CPU-FPGA communication falls into the classic cache access pattern. A cache transaction is typically depicted by its hit time and miss penalty. This depiction, however, does not take the payload size into account. In our study, we first follow the traditional methodology for cache study to quantify the hit time and the miss penalty of the cache. Table 3 lists the hit and miss latencies for cache read and write accesses. 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 parts — address translation, cache access and transaction reordering — and measuring the elapsed time of each step, as shown in Table 4. The data demonstrate a possibly exorbitant price (up to 100% extra time) paid for address translation and transaction reordering. Given this small but long-latency cache, it is extremely hard, if not impossible, for an accelerator to harness the caching functionality.

Then we compare the effective latencies among Alpha Data's DMA access (through PCIe), Alpha Data's device memory access and HARP's host memory access (through QPI)[4]. As shown in Fig. 7, for fine-grained data transfers (< 4KB), the QPI transfer expresses two orders-of-magnitude lower latency compared to the PCIe transfer. Also, when a payload is smaller than 4KB, the QPI latency will be even smaller than that of Alpha Data's private DRAM access. This phenomenon implies that a QPI-based platform is preferable for applications with fine-grained CPU-FPGA interaction.

In addition, we can see that the PCIe-DMA latency is almost constant (160μs) when the payload size is small; it increases linearly when the payload size grows above a certain threshold (around 64KB). This indicates that a PCIe-DMA transaction has a setup overhead of approximately 160μs, which dominates the overall latency for small payload transfers.

Table 3: CPU-FPGA access latency in HARP

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

Table 4: Hit latency breakdown in HARP

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

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 five insights for both platform users to optimize their accelerator designs, as well as platform designers to improve the hardware and software support in future CPU-FPGA platform development.

4.1 Insights for Platform Users

Insight 1: In terms of effective bandwidth, *PCIe Gen3 x8 (1.6GB/s) < QPI-based shared (remote) memory (7GB/s) < FPGA private device memory (9.5GB/s)*. Bounded by the low PCIe bandwidth, a PCIe-based platform generally reaches lower CPU-to-FPGA effective bandwidth than a QPI-based one. The higher private memory bandwidth, however, does provide opportunities for a PCIe-based platform 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 PCIe bandwidth will be amortized by the high private memory bandwidth, and the effective CPU-to-FPGA bandwidth will be nearly equal to the private memory bandwidth which is higher than the QPI-based remote memory access bandwidth. Therefore, the

[4]For simplicity, we mainly discuss the CPU-to-FPGA read case, the observation is similar for the FPGA-to-CPU write case.

data reuse of FPGA's private DRAM determines the effective CPU-to-FPGA bandwidth of a PCIe-based platform (Alpha Data), and whether it can achieve higher effective bandwidth than a QPI-based platform (HARP).

Quantitatively, we define the FPGA_DRAM reuse ratio, *r*, as:

$$r = \frac{\text{FPGA\_DRAM data access size}}{\text{PCIe transfer data size}}$$

Then, the effective CPU-to-FPGA bandwidth for a PCIe-based platform can be defined as:

$$\text{BW\_CPU\_FPGA} = \frac{1}{\frac{1}{r * BW_{PCIe}} + \frac{1}{BW_{FPGA\_DRAM}}}$$

According to the above formula, the higher *r* is, the better the CPU-to-FPGA bandwidth is, and the better the performance is. This could serve as an guideline to optimize the accelerator design in a PCIe-based platform. It is worth noting that since FPGA_BRAM reuse is typically important for FPGA design optimizations, the above finding suggests that accelerator designers using a PCIe-based platform need to consider both FPGA BRAM reuse and DRAM reuse. Moreover, by comparing this effective CPU-to-FPGA bandwidth of a PCIe-based platform to the DRAM bandwidth of a QPI-based platform, we can get a threshold of FPGA_DRAM reuse ratio, *r_threshold* (7.8 in our case). If the ratio is larger than *r_threshold*, a PCIe-based platform achieves higher bandwidth; otherwise a QPI-based platform wins. This could serve as an initial guideline for accelerator designers to choose a more suitable platform when accelerating bandwidth hungry applications. We will demonstrate the impact of the FPGA_DRAM reuse ratio in the matrix multiplication case study in Section 5.1.

Insight 2: In terms of effective latency, *PCIe Gen3 x8 > FPGA private device memory > QPI-based shared (remote) memory* for fine-grained (< 4KB) data access. Particularly, QPI-based memory access is two orders-of-magnitude faster than PCIe-based access due to the cumbersome PCIe-DMA setup overhead (160μs).

Therefore, a QPI-based 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 HARP's low communication latency. We will conduct a case study in Section 5.1 to demonstrate this benefit even if in the not-so latency-sensitive matrix multiplication application.

Insight 3: A PCIe-based platform can still catch up with a (much) lower-latency QPI-based platform with careful designs in many cases if not all cases. However, this requires additional code transformations to hide the CPU-FPGA communication latency, providing that the communication can be hidden by computation. Two techniques we study are double buffering and batch processing.

First, CPU-FPGA communication in a PCIe-based platform includes multiple steps: PCIe transfer, FPGA device DRAM access, and BRAM access. A well-known technique to hide FPGA DRAM access latency is double-buffer: while the FPGA kernel is processing one BRAM buffer, DRAM is transferring data to another BRAM buffer. We further extend this to hide the PCIe transfer overhead as well. The performance improvement of these two double-buffers together will be demonstrated in the matrix multiplication study in Section 5.1.

Second, we propose the batched processing technique to amortize the significant PCIe-DMA setup overhead. For computational kernels that run in very short time but are frequently invoked, the significant PCIe-DMA setup overhead (160μs) can kill the total benefit and even degrade the system performance. High-throughput DNA sequencing is one example with this characteristic: its computational kernels take less than 1ms to execute, but are invoked billions of times. Therefore, we propose to batch a number of short requests into one kernel to reduce the PCIe communication overhead. Detailed results are shown in Section 5.2.

4.2 Insights for Platform Designers

Insight 4: There is still large room for improvement to bridge the gap between the practically achieved PCIe bandwidth (1.6GB/s) and the theoretical one (6.8GB/s).

On one hand, the host-side PCIe driver and the device-side DMA IP might still have room to be improved (from 3.8GB/s to 6.8GB/s or close). On the other hand, the OpenCL buffer allocation and host memory copy processes also have room for improvement (from 1.6GB/s to 3.8GB/s). Next we discuss how to solve the latter one in more detail.

One possible solution is to grant users fine-grained control to directly manipulate pinned memory space. For example, both HARP SPL 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` [15].

Currently we are working closely with Xilinx to provide feedback for their SDAccel development environment improvement.

Insight 5: The long latency (70ns) and small size (64KB) of the FPGA-side cache post a serious challenge for users to take advantage of it. There is great opportunity for improvement of FPGA-side cache in future HARP products.

Conventionally, people only use software-managed scratchpad/buffer (BRAM) in an FPGA to achieve fast access (1 FPGA cycle or close). It is rewarding to see hardware-managed cache support in HARP since caches have proven to be very useful in conventional CPU and recent GPU architectures (like FPGAs, GPUs used to have scratchpad only, but now have both), in the sense of reducing programming efforts while achieving high performance. However, current FPGA cache access latency is too long compared to the 1-cycle scratchpad access. First, the virtual-to-physical address translation introduces a 20ns overhead, which should be further optimized. Second, even without this overhead, the pure cache access latency is 7 FPGA cycles (35ns), which should be improved to around 2 or 3 cycles. And these two could happen in parallel.

These optimizations have been done extensively in a conventional CPU and the old wisdom should be leveraged in the new HARP. Considering such a long cache access latency, this 64KB cache size is too cumbersome and almost useless compared to the MB-size BRAM with 1-cycle access. Currently we are working closely with Intel to provide feedback for their future HARP product improvements.

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: one is the well-known dense matrix multiplication kernel, and the other is a real-world application—high-throughput DNA sequencing [18].

5.1 Dense Matrix Multiplication

We study a systolic-array-based dense matrix multiplication (MM) [14] on both HARP and Alpha Data to demonstrate insights 1 to 3. To demonstrate the difference in CPU-FPGA communication, we make the computation kernel on two FPGAs the same: both have 128 floating-point MM processing elements (PE), and each PE handles tiled matrix of size 128^2 and runs at 200MHz.

First, to demonstrate the impact of FPGA_DRAM reuse ratio proposed in insight 1, we vary the matrix size as shown in the x-axis (bottom) in Fig. 8. Since MM kernel decomposes the matrix into sub-blocks of size 128^2, matrix data is read more repeatedly from DRAM as the size of the whole matrix becomes larger. The PCIe transfer, on the other hand, only needs to be done once regardless of the matrix size. Thus, larger matrix size leads to larger FPGA_DRAM reuse ratio, shown in the x-axis (top) in Fig. 8. As shown in Fig. 8, in the initial Alpha Data implementation (AD), the performance (y-axis, measured as Giga FLoating-point OPerations per Second, GFLOPS) increases with larger FPGA_DRAM reuse

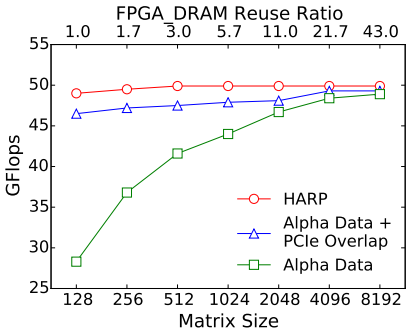


Figure 8: Matrix multiplication kernel execution time

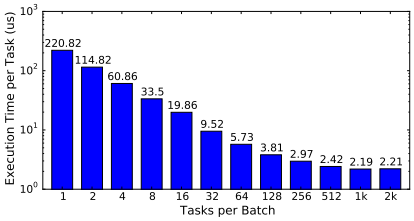


Figure 9: Average computation kernel execution time with different task batch sizes

ratio (matrix size).

Second, as shown in Fig. 8, the HARP implementation has a similar performance regardless of the matrix size, and most of the cases it has better performance than the Alpha Data one (insight 2). The reason is that the data access latency is faster and almost uniform in the QPI-based shared memory model. We should note that even with FPGA_DRAM reuse ratio as high as 43, Alpha Data almost achieves similar (but not better) performance to HARP. The reason is that even under this ratio, the bandwidth requirement itself is only about 1.6GB/s (i.e., not bandwidth hungry), which can be satisfied by HARP as well.

Third, the gap between Alpha Data and HARP can be solved by overlapping the PCIe transfer with the kernel execution (insight 3). By allocating two sets of buffers for the matrix (double buffer), the kernel may perform MM on one set of buffer, while OS transfers the next matrix data through the PCIe to the other buffer. Fig. 8 shows that the reference design with this optimization almost retains its highest performance even with small FPGA_DRAM reuse ratio. And the performance of this reference design is competitive with the HARP implementation.

5.2 High-Throughput DNA Sequencing

To demonstrate the usefulness of insight 3, i.e., batching to hide the FPGA kernel invocation overhead including OpenCL buffer allocation and PCIe setup, we further conduct a case study on a real-world application: high-throughput DNA sequencing. High-throughput DNA sequencing is a typical application that has the following two features: 1) the aggregate computational kernel execution time is extremely long; 2) the execution time of an individual kernel is extremely short. From the perspective of computer science, an instance of high-throughput DNA sequencing is to solve billions of approximate string matching problems, where each takes less than 1 millisecond on average, but tens of hours in total, on a 24-thread CPU. Apparently, if we naively design an accelerator to solve one approximate string matching problem and invoke the accelerator billions of times in Alpha Data (160μs setup overhead), it takes over 40 hours to just invoke Alpha Data's FPGA kernel.

As mentioned in insight 3, we can apply batch-processing, i.e., for each invocation, it processes not only one but many matching problems. We port the FPGA design proposed in [6] into Alpha Data and evaluate the average execution time of each computation kernel for different batch sizes. As shown in Fig. 9, a larger batch size does alleviate the invoked FPGA kernel overhead and significantly shortens the execution time of each computation kernel. The real computation kernel execution time should only be around 2.19 μs, when the batch size is around 1024. However, when there is no batching, i.e., batch size is 1, the computation kernel time is around 220 μs, where the overhead is far larger than the real FPGA kernel execution time. This demonstrates that batch-processing can greatly hide the latency of computation kernel invocation overhead.

6. CONCLUSION AND FUTURE WORK

To the best of our knowledge, this is the first paper to evaluate and analyze the microarchitectural characteristics of state-of-the-art PCIe and QPI based CPU-FPGA platforms in depth. We found that a QPI-based platform expresses impressive advantage on fine-grained (< 4KB) communication latency, and the FPGA_DRAM reuse ratio determines which platform supplies larger effective bandwidth. With a careful design, a PCIe-based platform can catch up with a QPI-based platform in many cases. We also observed that both platforms still have large room for improvement, and discussed possible solutions. We believe these insights can aid platform users in designing better accelerators, and facilitate the maturity of CPU-FPGA platforms. In our future work, we will conduct a similar study on the PCIe-based CAPI platform with coherent shared memory. To help the community measure other platforms, we also plan to release our microbenchmarks in the near future at http://vast.cs.ucla.edu/ubench-cpu-fpga.

Acknowledgments

This work was supported in part by C-FAR, one of the six SRC STARnet Centers, sponsored by MARCO and DARPA, and by NSF/Intel Innovation Transition (InTrans) Grant awarded to the Center for Domain-Specific Computing (CDSC). We thank Intel and Xilinx for their equipment donations. We also specially thank George Powley from Intel and Jim Wu from Xilinx for their assistance in this work.

7. REFERENCES

[1] "Alpha Data ADM-PCIE-7V3 datasheet." [Online]. Available: http://www.alpha-data.com/pdfs/adm-pcie-7v3.pdf
[2] "SDAccel development environment." [Online]. Available: http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html
[3] B. Brech *et al.*, "IBM Data Engine for NoSQL - Power Systems Edition," IBM Systems Group, Tech. Rep., 2015.
[4] T. M. Brewer, "Instruction set innovations for the convey hc-1 computer," *IEEE micro*, no. 2, pp. 70–79, 2010.
[5] N. Chandramoorthy *et al.*, "Exploring architectural heterogeneity in intelligent vision systems," in *HPCA*, 2015.
[6] Y.-T. Chen *et al.*, "A novel high-throughput acceleration engine for read alignment," *FCCM*, 2015.
[7] J. Cong *et al.*, "Composable accelerator-rich microprocessor enhanced for adaptivity and longevity," in *ISLPED*, 2013.
[8] S. Cook, *CUDA programming: a developer's guide to parallel computing with GPUs*. Newnes, 2012.
[9] E. G. Cota *et al.*, "An analysis of accelerator coupling in heterogeneous architectures," in *DAC*, 2015.
[10] Z. Fang *et al.*, "Measuring microarchitectural details of multi- and many-core memory systems through microbenchmarking," *ACM TACO*, 2015.
[11] A. Goldhammer *et al.*, "Understanding performance of PCI express systems," *Xilinx WP350, Sept*, vol. 4, 2008.
[12] Intel, "Accelerator abstraction layer software programmer's guide."
[13] Intel, "Intel quickpath interconnect fpga core cache interface specification."
[14] J. Jang *et al.*, "Energy-and time-efficient matrix multiplication on FPGAs," *IEEE TVLSI*, vol. 13, no. 11, pp. 1305–1319, 2005.
[15] Nvidia, "Nvidia's next generation CUDA compute architecture: FERMI," *Comput. Syst*, vol. 26, pp. 63–72, 2009.
[16] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *ISCA*, 2014.
[17] P. Rogers, "Heterogeneous system architecture overview," in *Hot Chips*, 2013.
[18] J. Shendure *et al.*, "Next-generation dna sequencing," *Nature biotechnology*, vol. 26, no. 10, pp. 1135–1145, 2008.
[19] J. Stuecheli *et al.*, "CAPI: A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 7–1, 2015.
[20] H. Wong *et al.*, "Demystifying gpu microarchitecture through microbenchmarking," in *ISPASS*, 2010.
[21] S. Yesil *et al.*, "Hardware accelerator design for data centers," in *ICCAD*, 2015.