Received August 8, 2018, accepted October 2, 2018, date of publication October 17, 2018, date of current version November 14, 2018.
Digital Object Identifier 10.1109/ACCESS.2018.2876597

A Hybrid Architecture With Low Latency Interfaces Enabling Dynamic Cache Management

MICHEL GÉMIEUX[1], MENG LI[1,2], YVON SAVARIA[1], (Fellow, IEEE), JEAN-PIERRE DAVID[1], (Member, IEEE), AND GUCHUAN ZHU[1], (Senior Member, IEEE)
[1]Department of Electrical Engineering, Polytechnique Montréal, Montreal, QC H3C 3A7, Canada
[2]School of Information and Science Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China

Corresponding author: Meng Li (lmbuaa@gmail.com)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada and in part by Huawei Technologies Canada Co., Ltd.

ABSTRACT The main focus of the dominant technologies in the high performance computation (HPC) market, such as GPU and multicore systems, is put on processing power, while much less attention has been paid to communication delays inside hybrid architectures. To fill this gap, this paper presents an experimental study on Intel's Broadwell Xeon multicore processor with integrated Arria 10 FPGA capabilities to characterize the communication delays between CPUs and the FPGA, using both the low latency cache coherent interface and the two PCIe links offered by this platform. The obtained results show that an FPGA cache access latency can be as low as 25 cycles at 400 MHz and that the platform is capable of reaching a bandwidth over 20 GB/s using an aggregate of the three available links. Furthermore, an FPGA-based cache management mechanism is proposed and implemented in this paper. A case study on a Merkle tree hash function shows that a hardware accelerator can achieve a fivefold data access acceleration in the worst case scenario. This scheme takes advantage of the QPI cache coherency and queuing theory to achieve a low latency and efficient memory management. In addition, design recommendations regarding the use of the CPU-FPGA platform for the implementation of fine-grained memory management schemes are suggested.

INDEX TERMS CPU-FPGA, multi-chip package, low latency, QPI, cache management.

I. INTRODUCTION

The advent of "Big Data" increased the demand for high performance computing systems to handle an overwhelming amount of information circulating in data centers around the world. Due to the volume of the data consumed at any time, throughput aware acceleration remains a bottleneck in HPCs [1]. This need motivated the development of heterogeneous computing clusters, which are mostly populated by accelerator cards, powered by GPUs, DSPs, and more recently add-on cards with large Field Programmable Gate Arrays (FPGAs). Some mainstream systems have been deployed by leading industrial companies, such as Amazon, which provides FPGA-accelerated clusters [2] for energy efficient and cost effective data processing.

Meanwhile, new applications may introduce new concerns that need to be addressed. Deep learning, for instance, stresses two prevalent requirements, namely the need for high memory bandwidth during the training of a neural network [3] and the need for low latency calculations for the inference of a trained application [4]. Other applications, such as High Frequency Trading (HFT) and 5G communication systems, impose also strict latency constraints [5]. There is no doubt that one of the key technological bottlenecks to solve is to guarantee low-latency and high-performance communications, while providing a high throughput.

Considered as a major step in High Performance Reconfigurable Computing (HPRC), Intel provides a platform with integrated FPGAs in CPU packages for data-center-based applications, called Multi-Chip Package (MCP) [6]. Specifically, this platform features a cache coherent interface between the CPU cache and a memory structure on the FPGA used as cache, which gives the opportunity for low latency hardware acceleration through its proprietary links. The cache coherent MCP allows for a cache-based communication with the FPGA. Such an interface enables cache manipulations of x86 processor caches, which is usually

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

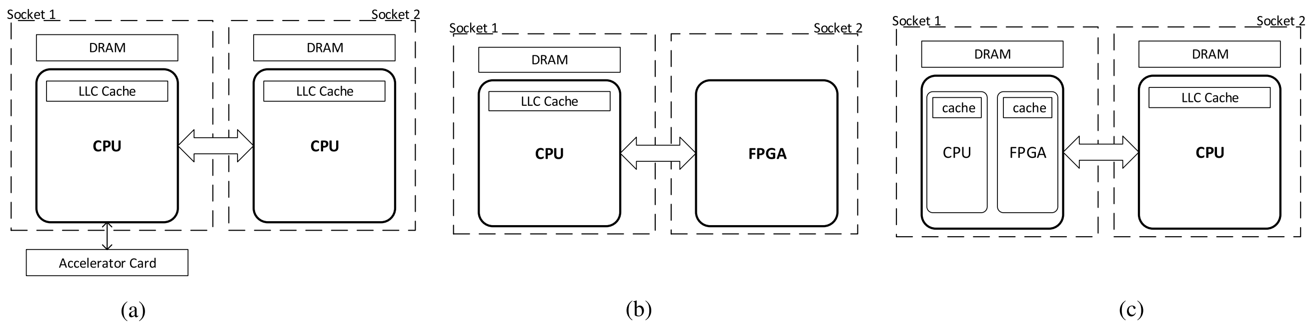


FIGURE 1. **Accelerator Architectures. (a) Accelerator Card. (b) Socketed FPGA. (c) MCP.**

reserved to inaccessible hardware system interfaces embedded in the CPU. This feature provides the end-users with more control over the processor's cache through the use of an FPGA, which can ultimately reduce the latency, increase the throughput, and enhance the determinism of co-designed applications.

By exploiting the new capabilities mentioned above, the aim of the present work is put on providing a solution for low-latency resource allocation acceleration for a class of hybrid architectures, especially cache management schemes, and an enhanced execution determinism. Due to the versatility and the parallelism provided by FPGAs, the present work explores some widely adopted CPU-FPGA architectures. Note that communication interfaces can significantly affect the transfer efficiency of a given architecture. Therefore, we investigate two prevalent CPU-FPGA interfaces from the literature, namely the Peripheral Component Interconnect Express (PCIe) [7] and the Quick Path Interconnect (QPI) [8].

The contributions of the present work include:

- the development of an accelerator-controlled cache management scheme that allows for a more efficient use of CPU cores for computations;
- an in-depth empirical characterization of the HARPv2 CPU-FPGA platform, which enables low latency resource allocation and application acceleration;
- the implementation of an accelerator for the hashing of a blockchain application to validate the proposed cache management scheme.

The experience gained in this work suggested guidelines on how to efficiently use socketed FPGA architectures, in order to identify means of manipulating CPU caches in a deterministic manner, with low latency accesses to shared memory in a multi-system architecture.

The rest of the paper is organized as follows: Section II reviews some communication standards widely adopted in hybrid architectures, as well as the architectures in which they are embedded. It also states the problem of real-time resource allocation on hybrid architectures. Section III presents the methods used to characterize low latency interfaces. In Section IV, we introduce a proposed queue-based cache management scheme. Section V presents the experimental characterization results of the MCP, along with a case study of a blockchain application conducted to validate the proposed cache management mechanism. Section VI concludes the paper and elaborates on future work.

II. BACKGROUND

This section presents some popular CPU-FPGA microarchitectures and makes a brief comparison of some most prevalent interfaces associated with memory models (hierarchy). It also presents a review of related work on latency optimization for hardware acceleration of real-time applications.

A. MICROARCHITECTURES

In high performance computing, device locality and fast interconnects between the host and its accelerator are performance critical. To a certain extent, regardless of the interface used in CPU-FPGA platforms, accelerator locality has a significant impact on communication delays, for both computation acceleration and data transmission.

A typical FPGA-accelerated server is composed of a daughter card in a PCIe slot on the motherboard, as shown in Figure 1a. This configuration induces usually a high latency. In addition, the complexity of the CPU's root complex can add extra overhead in cases where the data coming from the accelerator have to go through bridges, switches or chipset to reach the processing units. The accelerator's bandwidth is usually bounded by the physical implementation of its communication interfaces. Important improvements have been made in recent years by hardware platforms providers, such as Intel FPGA (previously Altera) and Xilinx, to enable the use of FPGA acceleration in HPC for "Big data" type workloads. Microsoft has successfully integrated Virtex 6 FPGAs in its first launch of their Catapult hardware [9], and then outperformed it with its new upgrade using Stratix D5 FPGAs [10]. Bing also used a hardware-accelerated search engine in its data centers for massive data analysis. The data flow is then highly optimized to favor massive processing over latency as explained above.

Socketed FPGAs, as shown in Figure 1b, aim at offering low latency, high throughput hardware acceleration. As discussed in [11] and [12], socketed FPGAs can offer low latency communication using the Front Side

Bus [13], or its new upgrade, the QPI bus. The earlier socketed FPGAs came as custom-made adapter cards fitting a specific socket type (i.e. LGA2011) [14]. In our previous work [14], we implemented a dynamic resource allocator using a QPI-based CPU-FPGA platform equipped with a Virtex 7 socketed FPGA. The introduction of QPI allows for a cache coherent interface with the accelerator. Using the accelerator system combined with an efficient parallel algorithm can achieve a speedup of two orders of magnitude, while reducing the latency overhead. With the same configuration, Intel launched the Heterogeneous Accelerator Research Program (HARP), through which the first QPI-based CPU-FPGA architecture using Stratix FPGAs was introduced [15].

A year later, Intel introduced in the second iteration of the HARP platform a Multi-Chip Package containing a Xeon-CPU and an FPGA [16], as described in Figure 1c. This new iteration coming out from the HARP offers improvements on the QPI management of the accelerator side and software development kits (SDKs), as well as the addition of two new communication interfaces exploring the PCIe Gen3 x8 standard. For a given implementation on the MCP, the interfaces can be used separately or combined. To the best of our knowledge, no latency bounded application exploiting the HARP MCP platform has been reported, while most publications using the HARP platform are related to machine learning. Unlike the work reported in [17], which provides information on processing performance per watt and/or accuracy for given models on specific configurations, the work reported in this paper focuses on latency metrics and characterization of the pre-market MCP.

Remarkably, most of the aforementioned platforms follow similar design flows for software and hardware co-design. The typical flow is to use APIs provided with a SDK to program the application on the CPU side, the Accelerator Abstraction layer (AALSDK) [18] and the Open Programmable Acceleration Engine (OPAE) [19] for the new Intel MCP. The FPGA part can be designed at various abstraction levels, ranging from low-level Hardware Description Languages (HDLs), such as VHDL and Verilog, to higher level languages, such as OpenCL. The OpenCL design environment varies according to the FPGA manufacturer, e.g., SDAccel for Xilinx [20] and FPGA Runtime Environment for Intel [21].

An important bottleneck in such systems is the interface between the host CPU and the FPGA. In this subsection we focus on the two types of interfaces available on the MCP: PCIe and QPI. The throughput of QPI is proportional to the processor clock frequency. A QPI has unidirectional send and receive link pairs, which can be activated simultaneously or independently. For a 2.4 GHz CPU, a QPI link has a 2 bit/Hz **rate** (double data rate), 20 data bits per **direction**, and **duplex** operation, with 8 **bits**/byte, which yields a theoretical throughput of 19 GB/s. An interesting feature of QPI is that the link comes with a cache coherency mechanism allowing for a unified memory space between the accelerator and the processor. Such mechanism allows for cache based communication between the host and the FPGA. It typically provides lower latency than other link types, and is equivalent to a 64B cache line access on x86 platforms. Given the information above, we can calculate the theoretical bandwidth of the QPI link as follows:

$$Bandwidth = \frac{Freq \times Rate \times Directions \times Duplex}{Bits} \quad (1)$$

The typical implementation of PCIe for FPGA accelerators is PCIe 8x Generation 3, which gives a link running at up to 8 GB/s. A PCIe link typically has a larger overhead than a transfer induced by QPI. As PCIe interfaces are throughput efficient, they are usually recommended for data transfers larger than 4 KB. However, PCIe is not cache coherent, even if it is allowed to access the unified address space of a process in the case of a CPU-FPGA MCP. Moreover PCIe links do not always deliver their theoretical performance. The actual bandwidth depends on the connection with the processor. It was shown in [22] that, in a topology similar to the one presented in Figure 1a, PCIe under-performed. However, in this work, a study of PCIe transfers in various scenarios allowed us to observe a close to the maximum theoretical performance. The main differences between PCIe and QPI are summarized in Table 1.

TABLE 1. Communication interfaces.

	QPI	PCIe Gen3 x16
Topology	Link	Link
Bus Width (bits)	20	16
Bi-directional Bus	Yes	No
Rate (GTransfers/s)	12.6	16
Theoretical Bandwidth (GB/s)	25.6	16
Memory	Coherent/MESIF	No

Cache coherent interfaces enabled by **unified memory spaces** in, e.g., QPI and CAPI interfaces provided by IBM [23] allow for a better memory management in accelerator architectures. Whereas non-coherent memory spaces imply that the host and/or the accelerator need to explicitly copy and allocate the data. In the case of message transmission acceleration towards the host with a topology similar to Figure 1b or Figure 1c, data copy and allocation are needed only on the host side. Simpler data structures will result in lower data allocation latency (new/alloc). For the case of **private memory spaces**, typically with a topology similar to Figure 1a, data allocation needs to be performed (copied) on both the host and accelerator main memories.

The main advantage of coherent unified memory spaces is that it allows for a device to directly access and consume the data without reallocation and duplication, as both the accelerator and the host have a direct access to the memory space. The MCP is one of the first platforms to support a unified memory-mapped I/O (MMIO) space between the host and the accelerator for x86 high performance server-based processors, allowing for zero-copy transactions. In this paper we use the combination of the cache coherent MCP and

low latency interfaces for real-time hardware acceleration of resource allocation.

B. REAL-TIME RESOURCE ALLOCATION

This subsection reviews some notable results on accelerators for real time resource allocation, among which latency reduction is one of the most attractive subjects. A typical way to get a reduced latency is to use a Real Time Operating system (RTOS), such as FreeRTOS [24]. It is explained in [25] how to reduce the latency inferred by resource management with smart assembly insertions at the kernel level. However, a problem associated with RTOSs is their rigidity and their lack of support for new computation libraries. A solution consists in the use of a new method and architecture. Indeed, there exist many solutions for hardware-base implementation of scheduling algorithms, such as [26]–[29]. In a previous work [30], we demonstrated that using StarPU with an adequate scheduling algorithm enabled the virtualization of a latency sensitive application (LTE) on a computing platform. It is demonstrated in [31] that gains can be obtained in terms of latency reduction for a RTOS supported by hardware implemented scheduling algorithms. Compared to the work reported in [31], where a RTOS on Xilinx Zynq is used, we plan in the present work to leverage hardware acceleration on a much larger scale based on a MCP FPGA on an HPC compliant OS.

III. CHARACTERIZATION OF LOW LATENCY INTERFACES ON A HYBRID CPU-FPGA ARCHITECTURE

In this section, we provide a detailed description on the method used to characterize low latency interfaces, including the specifications of the platform used, the test environment, and the means used to control the state of the memory hierarchy during the experiments.

A. HARDWARE CONFIGURATION

The detailed configuration of the hybrid architecture is built on a multichip package (MCP), as presented in Section II-A. This MCP contains a Broadwell 14 core 2.4 GHz Xeon processor integrated with an Arria 10 GX 1150 FPGA, both are server grade processing units. The FPGA has one hardware single precision floating point unit embedded in each DSP block (1518 blocks) and 54000 M20k memory blocks for high performance applications. Such a large FPGA coupled with a powerful general purpose processor allows performing large scale computations, and the cache coherent interface enables the implementation of fine grain accelerators.

Intel's HARPv2 connects the host to the accelerator (FPGA) with three links: two PCIe interfaces and one QPI as shown in Figure 2. This provides flexibility in data transfer adapted workloads. Intel's Core Cache Interface (CCI-P) [32] allows for leveraging all the links simultaneously or profiting from the use of a specific interface. Normally, the QPI link can achieve a maximum bandwidth of 12.8 GB/s. The platform also offers two PCIe 8x Gen 3 links, each with a theoretical maximum throughput of 8 GB/s giving a combined PCIe bandwidth of 16 GB/s. Combining all three links gives the system a theoretical maximum throughput of 28.8 GB/s.

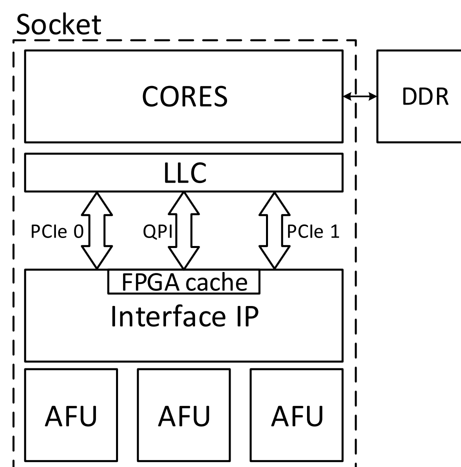


FIGURE 2. The QPI and PCIe connections between CPUs and FPGA. Last Level of Cache (LLC), Double Data Rate (DDR), Accelerated Function Unit (AFU).

B. CHARACTERIZATION APPROACH OF LOW LATENCY INTERFACES

This subsection details the method used for MCP characterization, while the related experimental results will be reported in Section V. It should be noted that the results are obtained on a pre-production platform and may not reflect the best performance that a production-ready MCP can offer. Efficiency and optimization analysis of the interfaces should be carried out once more when the production platform is available. This is justified as the pre-production devices might have some unresolved issues to be corrected in the final product.

Latency and bandwidth measurements of the MCP reported later were performed using a loopback application. We first explain the test environment and our workflow, followed by a detailed report of the method used for characterization.

We used the AALSDK 5.0.3 for the host program coded in C, and the Accelerated Function Unit (AFU) in the FPGA is coded in Verilog. Due to the scarce availability of the MCP platform, the current workflow imposes to work remotely on HPC servers [33]. For efficiency purpose, we first simulated the entire environment using the Intel's Accelerator Functional Unit Simulation Environment (ASE) [34]. The ASE is linked to Questasim 10.6a [35] (or Synopsys) for code compilation and simulations, as well as to Quartus 17.0 [36] for the synthesis and place-and-route features. Once a program is validated in the simulation environment, the entire program and the bit file are transferred to the HARP servers for testing. At runtime, the overall FPGA clock is bounded at 400 MHz by the CCI-P maximum frequency, giving a 2.5 ns period.

As the QPI link is cache coherent, the cache initial state needs to be defined. To have reproducible results, this experimentation starts at a state where the cache is not full nor

configured to maximize collisions. To optimize the experiments, the current implementation of the loopback enables the system to have a warmed up cache in order to facilitate cache hits [37].

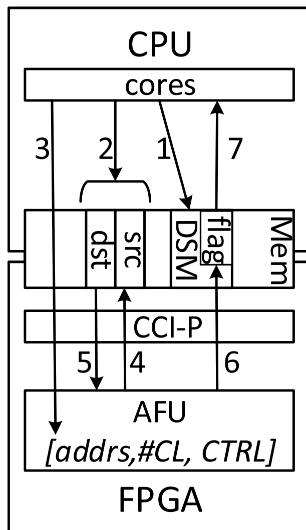


FIGURE 3. The loop back application.

Figure 3 illustrates an in-depth representation of the proposed loopback test. At high level, a loopback test consists simply in sending a message and waiting until the message is sent back to the transmitter. The CPU starts the test, then the FPGA initiates a write request and starts the time-stamping. As soon as the CPU receives the data, they are written back to another location in the shared (cached) memory. When the FPGA receives an acknowledge that the data were written, time-stamping is stopped. At the end, the FPGA verifies the data transferred.

Note that on the software side, Intel's AALSDK provides an API that can be used to setup a service between the CPU and the FPGA, as well as MMIO bounded functions to use and align virtual and physical addresses. Table 2 shows some useful functions used in the considered loopback test.

TABLE 2. Useful AALSDK API functions.

Function Name	Utility
ALIBuffer	Align Virtual to Physical addresses
ALIMMIO	MMIO Access
ALIUMSG	Low Latency message
ALIReset	Reset AFU

The hardware implementation of the loopback application is straightforward. The AFU contains three main blocks: a **read** Finite State Machine (FSM), a **write** FSM, and a free running counter based on which important events will be time stamped. The remaining Verilog code assists and enables the AFU and CCI-P communication. Each FSM can receive and send requests to and from the CCI-P.

Typically, for the case of an FPGA write loopback initiated test, the free running counter starts when the processor start flag is received by the AFU and stops when the AFU reads its last cache line. As an example, the operation initiated by the AFU would go as follows:

- Starts a free running counter at an AFU start;
- Timestamps an event at a write request to the CCI-P;
- Timestamps an event at a response of the request from the CCI-P;
- Stops the free running counter at the last read response.

At the end of each test, the timestamps are sent back to the CPU for evaluation.

Figure 3 shows the detailed interactions of the processor and the FPGA for platform characterization. Each arrow is associated with a number, which corresponds to the action described by the bullet of the same number listed below. The list corresponds to a typical FPGA initiated loopback test, where the FPGA writes the source buffer in cache, and the CPU copies that source buffer to the destination buffer. The FPGA then validates the data to confirm the test validity.

1) The processor sets up the Domain Specific Memories (DSMs). The DSMs are a combination of host specific memory spaces as well as cache aligned spaces intended to be used for commands to be passed to the AFU, also called Control Specific Registers (CSRs).
2) The host creates and initializes the source and destination memory spaces. Each space is set to a specific size of 1024 cache lines, which is 64KB. It also corresponds to the size of the FPGA cache on the AFU side.
3) The CPU sets the MMIO CSRs. It creates and asserts a start flag for the test in the CSR. It dictates the AFU to start.
4) The CPU listens until the done flag is set.
5) The AFU writes in the source buffer and then reads the content copied by the CPU in the destination buffer.
6) The done flag is set.
7) The CPU acknowledges the termination of the test, receives the events timestamps, and the AFU verifies that the data in the SRC and DST buffers are the same.

The acknowledgement received in Step 7 concludes a typical execution of the loopback application. The above test is initiated from the FPGA, however the loopback test can also be initiated from the CPU side, which should yield similar results. The low latency interfaces enable us to have a cache management mechanism implemented on an FPGA. The following section provides the details of the implementation of the proposed cache management. The characterization results are presented in Section V.

IV. PERFORMANCE IMPROVEMENTS WITH QUEUE-BASED CACHE MANAGEMENT ON AN MCP

In this section, we propose a novel approach using an FPGA to enable efficient cache management and reduce the waiting time caused by data fetching. In general, cache management is performed by dedicated CPU hardware, such as the Cache Allocation Technology (CAT) [38] for Intel processors, which aims at enhancing the system. However, it is not trivial to implement cache management as well as

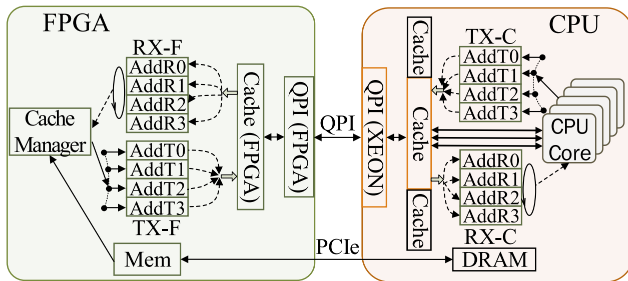


FIGURE 4. Proposed hybrid architecture composed of CPUs and an FPGA.

task scheduling on the CPU side in an efficient manner. Benefiting from the hybrid CPU-FPGA platform shown in Figure 4 and its low latency transmission delay through the QPI interface, we propose a queue-based cache management technique inside the FPGA, which can be used in real-time applications, such as task allocation and scheduling. Indeed, cache management with deterministic performance can be implemented with the help of modules in the FPGA. The immediate benefits of this hybrid architecture are summarized as follows:

- Deterministic execution and performance guarantee of a cache manager provided by the FPGA for real-time applications;
- Low latency QPI and high throughput PCIe buses for communication between a CPU and the FPGA;
- Reconfigurable implementation of cache management and upgraded capacity for function acceleration with the FPGA over the CPU alone.

As the main processor in the MCP is cache coherent with its accelerator, FPGA cache can be regarded as a copy of a designated part of the CPU's Last Level of Cache (LLC). Therefore, any change in either cache will be replicated to the other side via the low latency QPI bus. This mechanism enables the management of the shared cache with an FPGA. It is known that an FPGA offers parallel computation capability. Thus, the FPGA can provide deterministic performance cache management without interference from an operating system (OS). Furthermore, in practice, it is possible to boost execution performance and hide transmission latency by pre-fetching data from main memory to cache before task execution. However, it may be impossible to pre-fetch large amounts of data for a given execution, due to cache current state and overall size. Thus, efficient management schemes are required.

In its basic implementation, the management scheme is represented as content queues written by the FPGA to be replicated or read from the CPU through the cache. This mechanism can be explained with a task scheduler using our cache management scheme. Therefore, for a given Directed Acyclic Graph, a queue-based cache manager is proposed and implemented in an FPGA.

As shown in Figure 5, the shared cache is divided in two main memory blocks. The first block is used to receive the control signals for and to the FPGA, such as done, ready,

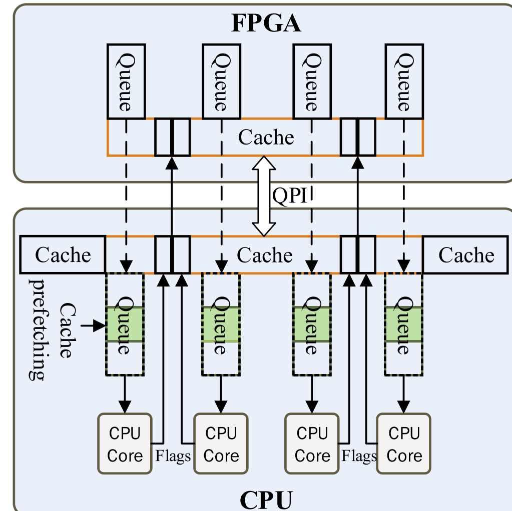


FIGURE 5. Queue-based cache management.

send and acknowledge flags, represented as the sub-blocks arrow in the figure. The other block is used to write the fetched data, in order to have it ready for task execution by a CPU core. In a typical flow, the FPGA containing ready task queues for each CPU execution units fetches the necessary data ahead of task execution if it is ready into the corresponding CPU queues. The fetched data are therefore ready for execution. In order for the FPGA to keep track of tasks execution and to maintain its ready task queues, the CPU sends control information to the FPGA such as task ID, ready and done flags.

The handshake between the processor and the accelerator ensures an efficient resource management. Using the handshake and the FPGA in a well configured OS environment enables a near deterministic hardware accelerated cache management, improving the underlying system's performance. Multiple steps, such as adding kernel patches [39], running on a bare metal OS [40], disabling hyperthreading [41], as well as using core isolation techniques, can be taken in order to improve the determinism. Even though taking all these steps should produce a much more deterministic cache management, it remains to be confirmed experimentally whether other sources of performance variability would remain.

Note that in order for the previously exposed scheme to be efficient, it is assumed that there are enough tasks ready for execution. Besides, task execution time needs to be longer or comparable to the time required for prefetching data. Thus, normally, benefiting from queue-based cache management and cache prefetching, a continuous task processing can be realized without any waiting delay due to input data loading from main memory to cache.

To validate the expected benefits from the queue-based cache management and cache prefetching scheme introduced above, we present a "case study" in the following section. It serves as a proof of concept on which future work may expand.

V. VALIDATION OF LOW LATENCY COMMUNICATION AND CASE STUDY OF CACHE MANAGEMENT

In this section, the low latency interfaces are characterized and utilization recommendations of the hybrid platform are provided. Then, we implement the proposed cache management scheme in an FPGA embedded in the MCP architecture to validate the improvements it can bring to system performance. Note that due to the scarcity or non-availability of socketed and MCP type CPU-FPGA architectures from other vendors, such as Xilinx, this paper focuses on the comparison of delay and bandwidth for all three architectures mentioned in Section II, based on the Intel platforms.

TABLE 3. MCP access latency.

Access	Latency (FPGA cycles)
Read Hit	25
Write Hit	28
Read Miss	119
Write Miss	129

A. CHARACTERIZATION RESULTS OF LOW LATENCY INTERFACES

This subsection presents and dives in depth in all the results obtained on the MCP platform. It also provides some recommendations on how the platform may be used to generate low overhead applications. The MCP CPU-FPGA QPI communication has typical cache memory access patterns, where a cache transaction is described as either a ''cache hit'' or a ''cache miss''. As shown in Table 3, the QPI's cache hit latencies are similar to those exposed in [22]. However the average read and write miss latencies are reduced by around 65%. We also observed that unlike [22], the empirical bandwidths are close to their theoretical values for QPI, about 12.6 GB/s as seen in Figure 7b, but still lacking for PCIe with about 5.2 GB/s for PCIe, as shown in Figure 6c.

To better understand the intricacies of the different links, we first compare their performances in the cold cache environment, then with a warm cache to conclude with a discussion on which link is adequate for what situation. Figure 6a shows the write throughput of PCIe alongside QPI's, for transfers from 1 to 65535 cache lines in a cold cache environment. We observe that PCIe has a higher write throughput than QPI for small transfers, i.e., 1 to 32 cache lines. However for transfers over 32 cache lines up to the cache capacity, QPI's throughput is better than that of PCIe, exhibiting a peak at around 10 GB/s, almost twice PCIe's throughput and dropping down towards the level of PCIe after cache capacity is reached. For all transfers over the 64 KB cache capacity, throughput for QPI and PCIe are similar around 8192 cache lines, while QPI lacks behind for larger transfers, due to the cache coherency protocol (evictions and cache misses). For the case of read throughput, Figure 6b shows that QPI has a slight advantage over PCIe for read throughput up to 128 cache lines, at which point PCIe and QPI have a similar read throughput of about 5.2 GB/s, equivalent to PCIe's measured limit. Figure 6c and Figure 6d, show the detailed evaluations of PCIe and QPI for their read and write throughput in the aforementioned cold cache environment. As for the links latency, we observe in Figure 6e that QPI has a smaller latency of about 100 cycles for transfers from 1 to 32 caches lines reaching PCIe's latency around 128 cache lines. However the inverse happens for PCIe, where its write latency in this cache environment is smaller than QPI's for the same small transfer interval, as shown by comparing Figure 6f and Figure 6g. Figure 6h compares PCIe and QPI loopback latencies. It shows that they are similar, meaning that the PCIe write affinities and QPI's read affinities cancel out in a cold cache environment. It indicates that for all applications requiring quick loopback performance in a cold cache environment, either links are suitable.

As for results in the warmed cache environment, we observe that QPI is greatly affected by the cache state. It can be observed that its results in a cold cache environment are both worst in read and write latencies over the warmed cache results. Figure 7a shows that we can achieve the lowest read and write latencies for QPI, i.e., 25 and 28 cycles respectively. The warmed environment allows for QPI to reach its announced maximum throughput of 12.6 GB/s for reads and writes, as shown in Figure 7b. Due to the multiple links available on the MCP, using an aggregate of the two PCIe and the QPI link is possible. Figure 7c and Figure 7d, show the overall latency of read and write transfers when the links are used simultaneously. The histograms on each figure display the number of reads and writes done by each type of links (QPI and 2x PCIe). For instance, it can be observed that, in Figure 7b, the aggregate read latency is equivalent to that of QPI in a warmed cache; that is due to the aggregated protocol using only QPI, as its histogram display 1 count for QPI and 0 for the combination of both PCIe links. For a bigger transfer of 256 cache lines, we observe that the read latency of the aggregate is better than PCIe alone but worse than QPI alone for the same transfer size, which can also be explained by the transfer count of the aggregate favoring QPI over PCIe, with 137 QPI reads and 119 PCIe reads. Lastly, Figure 7e shows that the highest throughput can only be achieved using aggregated transfers, with an average of 19 GB/s and a max of 20.634 GB/s for a 1024 cache lines writes.

To summarize, in the cold cache environment, the QPI link does not have a clear advantage over PCIe. PCIe has a small latency over QPI for writes when transferring less than 32 cache lines, which is also evident for write throughput. On the other hand, QPI displays better read latencies and throughput across the board for all transfers under the FPGA cache size. Once the cache size limit is reached, PCIe and QPI have similar performances. Lastly, in the warmed cache environment, QPI read and writes are better than PCIe in all instances where the data fit in cache, where the read latency reaches as low as 25 FPGA cycles, as well as reaching the link's theoretical max bandwidth of 12.6 GB/s. Note that as PCIe is a non-cache coherent link, its performance is not affected by the cache state. The results show that as long as the

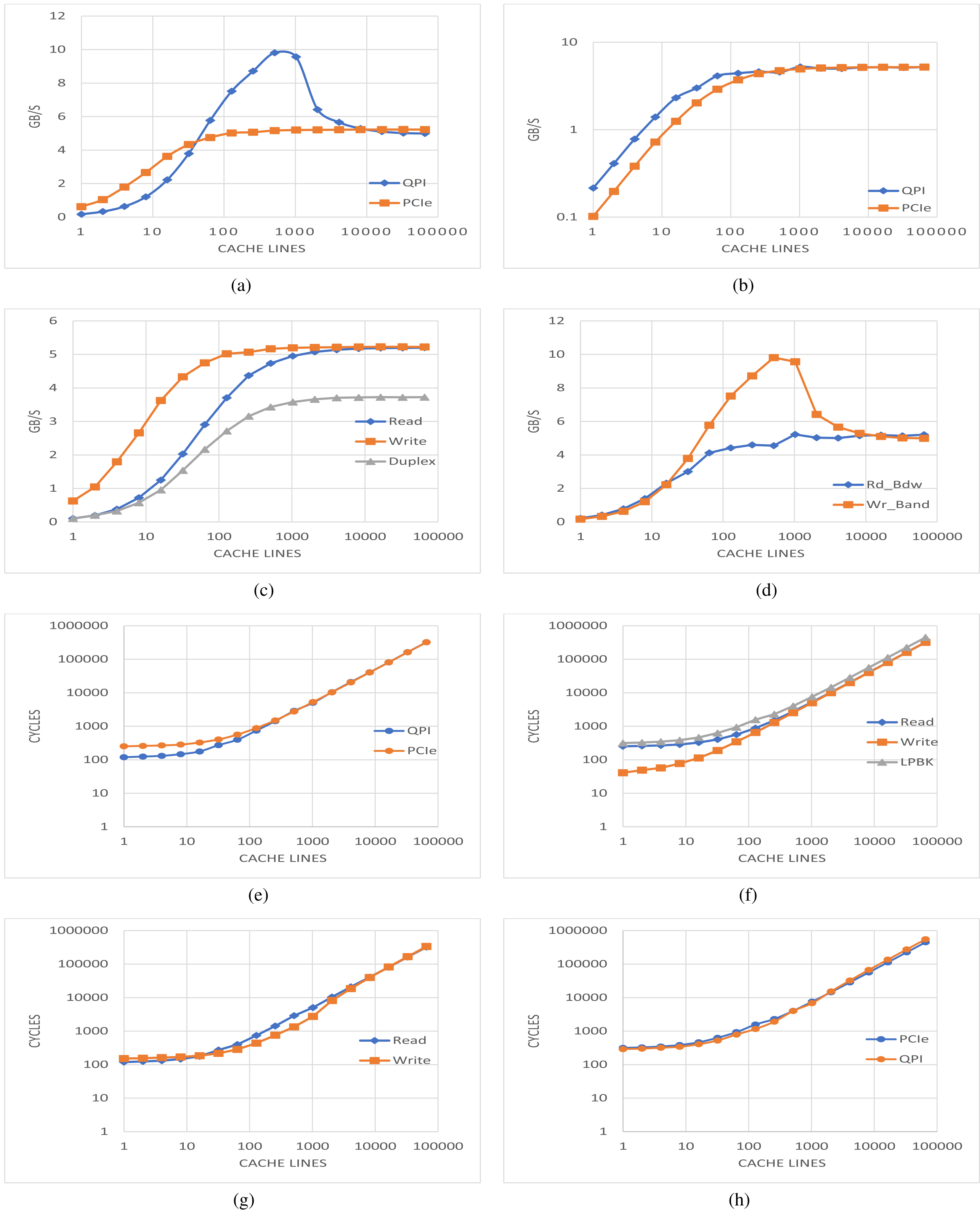


FIGURE 6. Cold cache characterization results. (a) QPI vs PCIe write throughput. (b) QPI vs PCIe read throughput. (c) PCIe throughput results. (d) QPI throughput results. (e) PCIe throughput results. (f) QPI throughput results. (g) QPI Read and Writes latency. (h) QPI vs PCIe loopback latency.

transfers fit in the available cache, QPI is the preferred interface to use for low latency communication with the processor, while it cannot deliver a high throughput for under 32 cache line writes. In most of the results, PCIe comes second to QPI in performance, except in write performance for small data transfers due to the PCIe write affinity to main memory.

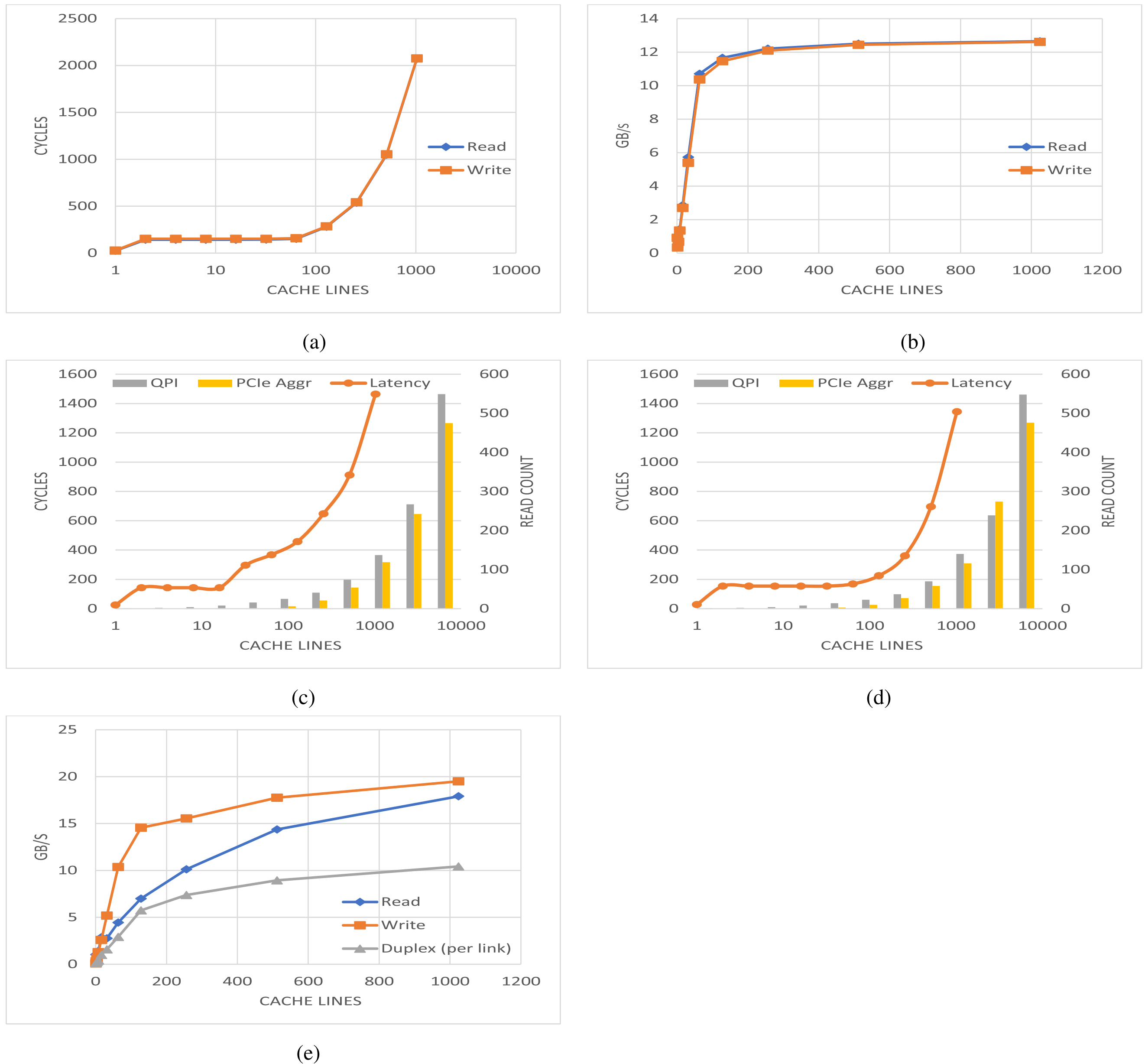


FIGURE 7. **Warmed cache characterization results. (a) QPI read and write latency. (b) QPI throughput results. (c) Aggregate Link Read latency. (d) Aggregate Link Write Latency. (e) Aggregate Link Throughput.**

Note that as QPI is a cache coherent link, data transfers are cached, allowing for better performing applications based on data locality alone. A good compromise for latency and bandwidth is the use of aggregated transfers on the three available links. Note that in both read and write latencies, the QPI link is favored over the PCIe links. This ensures the lowest possible latencies for reads and writes, while getting the best overall throughput. Spreading the transfer load on multiple interfaces could have a benefit on memory contention with long running applications. However, more experiments are necessary to verify this hypothesis. We also noted that in most of our warmed-up cache experiments, we would get a cache miss on the second cache-line transfer. More investigations are needed to find the source of the problem, although it does not invalidate the insights and the results provided in this paper.

In light of the results in Figure 6 and Figure 7, conclusions on how to better use the MCP can be drawn. Most of the insights provided in [22] on offload to an accelerator with an architecture similar to Figure 1a hold true, especially those related to data movements. However, some assumptions and recommendations targeting the PCIe in particular are invalidated by the use of the unified memory space of the MCP. A decision tree to choose interfaces in different circumstances is provided in [22]. The following tips and suggestions should be taken as recommendations for MCP users and designers.

Insight 1: For the lowest latency transfers, QPI should be prioritized. We would recommend that as long as the amount of data to be transferred is less than the remaining available cache size, the QPI interface should be used over the PCIe. For a cacheline, the typical latency of a QPI transfer is 64 ns (25 cycles) versus 625 ns (250 cycles) of a PCIe transfer. A QPI cache miss penalty of a cacheline is still more latency efficient than a PCIe transfer.

Insight 2: QPI offers a higher bandwidth than one PCIe x8 Gen3 for payloads up to 64 KB (for an empty cache). In order to have a higher bandwidth irrespective of latency, both high latency links should be used, offering up to 16 GB/s throughput.

Insight 3: If latency variability is not an issue, using the Intel automated aggregate interface outperforms the PCIe interface alone for most cases. Using both interfaces provides the "best of both worlds" as exposed by Figure 7. Whenever the payload is small enough, QPI will be the prioritized interface, while PCIe will be for larger payloads and both for the in-between. Compared to PCIe only, even with the latency added by memcopy, the aggregate transfers are more efficient. The same applies for high throughput needs, and Figure 7 shows a throughput up to 20 GB/s using the automatic aggregate interfaces, out of a theoretical 28 GB/s. However using all the links at once, a well-crafted manual implementation might allow for a higher bandwidth.

Insight 4: The MCP accelerator may not be the solution for all use cases of FPGA acceleration. For example, if the data from the functions to be accelerated are to be transferred to another machine, the multi-interface transfer latencies might invalidate the use of the integrated accelerator. That would be the case if the latency to perform an additional back and forth transfer between the CPU and FPGA adds more time to the execution than the processing time saved. In this case a NIC equipped daughter card FPGA might be preferable.

Insight 5: On the accelerator side, using the FPGA cache might still be ill-advised due to its long access latency (64 ns for reads and 70 ns for writes) compared to a BRAM or register access (typically 1 to 2 cycles). The cache is better used as an interface between the CPU-FPGA than as a storage medium for FPGA's AFUs.

Insight 6: The MCP accelerator is better used in the unified memory space of the system. Whenever offsite memory accesses are needed, a DMA enabled daughter accelerator card might be preferred.

B. A CASE STUDY - MERKLE TREE ACCELERATION

Since the inception of Bitcoin by a still unknown creator going under the pseudonym of "Satoshi Nakamoto" [42] in January 2009, hash-based cryptography for decentralized applications gained a widespread popularity. The growing cryptocurrencies realm counts over 1624 different "active" tokens over multiple exchanges and platforms. Most of these tokens rely on a hash-based tree structure for their digital signature schemes. This tree based generalization of a hash list is called a "Merkle tree" [43]. Each leaf node is a hash of a block of data and a non-leaf node is the hash of its children. Typically, a Merkle tree has a branching factor of two.

An integral part of decentralized trust-less transaction systems, such as the first pseudo-anonymous Bitcoin [44], the enterprise aware Ethereum peer-to-peer networks [45] or the privacy token Monero [46], is called a "Block". The block represents the main scalability feature of cryptocurrencies, where it is part of a multi-level data structure within the "Blockchain". The block records all the transaction data for the case of Bitcoin. Therefore the sequence of multiple blocks over time represents the blockchain. Figure 8 shows the relationship between the block and the Merkle tree, and also displays a typical hash-based tree structure.

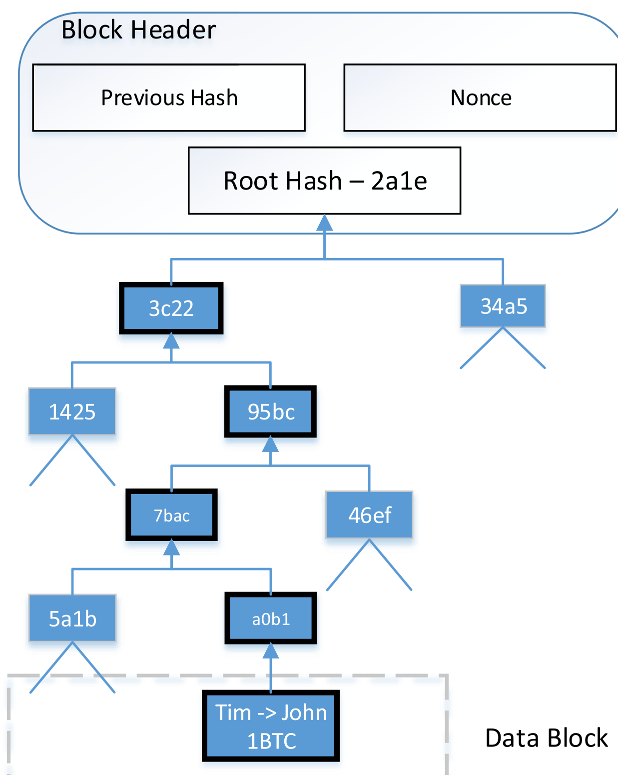


FIGURE 8. Merkle Tree in Blockchain applications.

The case study considered in the present work includes a simple implementation of a blockchain with a SHA-256 hash function. Within the Bitcoin network, the SHA-256 hash function serves for two mains purposes [42]:

- Mining, where a "miner" creates a new block to the block chain with a process called "proof-of-work" to ensure that all the blocks added to the blockchain are legitimate and free of fraudulent information. The proof-of-work mechanism solves a SHA-256 hash function.
- Creation of new addresses, where every new address needs a public key to be hashed. SHA-256 is the hash function used for the creation of new addresses because it allows for greater security as well as shorter addresses.

The advantage of creating Merkle trees within blocks in the case study is that the size of a SHA-256 hash function

is 256 bits, i.e., 32 bytes [47], which is coincidentally half a cacheline long. It allows for our implementation of hardware prefetching to combine data, addresses, and any other possibly relevant data in one cacheline transfer between the CPU and the AFU. Another advantage is that the computation of a more complex hash function on the CPU side allows for longer computation, and hence, it hides some of the communication time implied by the accelerator prefetching on the MCP.

This makes use of a simple Merkle tree with a SHA-256 cryptographic hash function reminiscent of cryptocurrencies, such as Bitcoin, representing an adequate application to test our cache-based management scheme for hash computation acceleration using an accelerator prefetching. The following subsection dives into the details of the method and implementation, as well as giving the results of this proof of concept.

C. IMPLEMENTATION OF MERKLE TREE AND EXPERIMENT RESULTS

In order to show that accelerator prefetching could be beneficial, we developed a simple Bitcoin-esc blockhain chain with an oversized Merkle tree. For each block, we created a 30 MB Merkle tree to make sure that during the tests the CPU would not fetch the entire tree when needing to compute the hash of a given node. For each test, a block is created and it would iterate multiple times to compute the hash of random nodes until 64 000 nodes are computed. The use of the random function allows for low to no compiler optimization. Listing 1 shows the order in which the application runs.

```
  merkt_t *merkt = merkt_create();

  prevAddr = startAddr;

for (uint32_t i = 0; i < 64000; ++i) {
  nextAddr = rand_node(merkt, seed,addrSpan);
  send_Addr(getNextAddr,m_inputVirt);
  compute_hash(prevAddr);
  prevAddr = nextAddr;
}

merkt_delete(merkt);

print(``End Test'')
```

Listing 1. Merkle tree implementation pseudo code.

Each time we access a specific node of the Merkle tree to compute its hash, we use a pseudo random function that chooses the next branch to compute. The predicted address is relayed to the FPGA in order to prefetch the data associated with the address, while the processors compute the previous node's hash. In order to transfer the address from the CPU to the FPGA, a simple service is implemented. A virtual address is linked to a cache-aligned physical address. Listing 2 shows the use of pointers to the virtual address to copy the next node to compute via the communication service implemented with the AALSDK.

```
//Save size for input virtual address
m_pALIBufferService->
bufferAllocate(ALLOCATION_SIZE, &m_InputVirt)

m_InputSize = BUFFER_SIZE;

m_InputPhys = m_pALIBufferService->
bufferGetIOVA(m_InputVirt);

// Set the input address for communication
m_pALIMMIOService->
mmioWrite64(SRC_ADDR, CL_ALIGN_ADDR(m_InputPhys));
```

Listing 2. Address transfer through a service..

On the FPGA side a simple memcopy type application is implemented. It receives the address and reads the data associated with the address in order to keep the data in cache. The hardware interface is coded. The AFU "read" FSM is shown in Listing 3.

```
begin
   case(read_fsm)
   2'h0:
   begin // Waits for go
    RdAddr <= 0;
    RdLen <= CL_len;
    RdSop <= 1'b1;
    Num_Read <= 20'h0 + CL_len + 1'b1;

    if(go)
      if(NumLines!=0)
       read_fsm <= 2'h1;
      else
       read_fsm <= 2'h2;
  end

  2'h1:
  begin // Send read requests
   if(RdSent)
   begin
     RdAddr <= RdAddr + CL_len + 1'b1;
     RdLen <= RdLen;
     RdSop <= 1'b1;
     Num_Read <= Num_Read + CL_len + 1'b1;

      if(Num_Read_req == NumLines)
      if(Cont) read_fsm <= 2'h0;
      else read_fsm <= 2'h2;
    end
  end

  default: read_fsm <= read_fsm;
  endcase

end
```

Listing 3. Simple application.

Essentially, once the AFU reads the next address to hash, the data becomes available in the cache for the processor resulting in fewer accesses to the main memory for computation, which should allow the application to run faster.

One thing to note is that the current implementation of the MCP provides a MMIO space accessible by the FPGA, but it has a limited size of only 4MB. This means that the current case study only runs with addresses in that 4MB span. Given that span, the theoretical maximum number of 64B accesses

is 64,000. Given the information above, we can calculate the theoretical improvement limit of accelerator prefetching as the maximum latency (data is not in cache) divided by the minimum latency (data is in cache):

$$Acceleration = \frac{T_{Hash} + T_{MainAccess}}{T_{Hash} + T_{L3Acces}}, \quad (2)$$

where "$T_{MainAccess}$" is the typical main memory access time, "$T_{L3Acces}$" the typical L3 cache read access time from the CPU, and "T_{Hash}" the typical time to compute one SHA-256 hash on one CPU core. This equation is true only if the hash computation takes longer than the time it takes for the AFU to receive the next node's address and read. It is important that AFU processing time is masked by the hash computation, otherwise it would add time to the overall execution, thus invalidating the acceleration. For the equation to be effective, it also needs a typical main memory access longer than a L3 cache access. The Xeon E5 provides a typical per core hash rate of 2.06 Mh/s. This implies that it would take about 490 ns to compute one SHA-256 hash, which is 4.9x longer than the 100 ns of worst case main memory access on a CPU. The values in Table 3 and Table 4 show that a typical send address and read address from the CPU to the AFU would be around 72.5 ns in the best case and in the event of an AFU cache miss and the longest L3 access, 317 ns would be the worst case. The formula is therefore true for our test, and the theoretical acceleration limit for the considered application is 1.157 for the worst scenario, where the respective L3 and main memory accesses would take the most time.

TABLE 4. Typical memory hierarchy access times.

Memory Level	Access Time (ns)
L1 cache	1-5
L2 cache	5-10
L3 cache	10-20
Main Memory	50-100

A remarkable outcome is a faster access time implied by the efficient prefetching. As shown in Table 4 the AFU-based prefetching reduces a main memory access to a LLC access, giving a 5-fold latency improvement in the worst case scenario. This implies that for a more memory bounded application, the overall application speed up would be much more significant. For an application in which time would be consumed 90% by memory transfers, Amdahl's law expresses:

$$S_{max} = \frac{1}{(1-p) + (\frac{p}{s})}, \quad (3)$$

where s is the performance improvement factor and p the part that can be improved. The maximum theoretical speedup of the application would be of a factor of 3.6, where the improvable portion of the application is 90% and the improvement is 5-fold.

After running the Merkle Tree test for 1 million different blocks, we get an average acceleration of 1.118 with a min of 1.10 and a maximum of 1.133, which is close to the theoretical limit of 1.157 computed by (3). After some investigation, we may improve the acceleration using a low latency messaging API provided by the AALSDK. Another possible improvement would be to limit the Merkle tree and the random node accesses to a specific 4MB space. It would allow the AFU to "prefetch" all the data to be processed. However such an implementation could induce more evictions and cache misses in the FPGA cache, which could result in a lower performance, thus negating the need for accelerator prefetching.

The case study described above serves as a proof of concept for accelerator-based cache management. The results show that the prefetching can be done using an FPGA embedded in the MCP. Other cache management techniques would be feasible using the MCP, which will be addressed in our future work.

VI. CONCLUSION AND FUTURE WORK

In this paper, we provided an in depth characterization of the new CPU-FPGA MCP. We introduced a new cache management technique, where the accelerator shares the control and is able to prefetch data to make it readily available to the CPU cores. The study of the MCP exposed a new CPU-FPGA platform which enables low latency and high bandwidth resource management, while being able to be used to accelerate latency bounded applications. The MCP's characterization shows that it can achieve access latencies as low as 64 ns and throughput as high as 20.6 GB/s. A fine-grained acceleration became possible due to the coherent FPGA cache with the LLC through the QPI interface, as well as being able to deal with larger datasets using the CPU's main memory's address space through the PCIe. We made a proof of concept using the accelerator of the MCP to prefetch the following hash to be computed of a Merkle tree, which resulted in a typical application speedup of 1.118 and a 5x data access speedup in section V-B. We also confirm the recommendations provided in [22], while adding newer ones for the use of the new MCP platform.

This research is a first step towards low latency resource allocation in HPC environments. As future work, it is planned to use the on-board FPGA of the MCP to study cache managements techniques on top of using lock-free and wait-free programming to obtain lower latency and more deterministic processing for more real-time applications.

REFERENCES

[1] G. E. Moore, "Cramming more components onto integrated circuits," in *Readings in Computer Architecture*, M. D. Hill, N. P. Jouppi, and G. S. Sohi, Eds. San Francisco, CA, USA: Morgan Kaufmann, 2000, pp. 56–59.
[2] *F1 Instances*. Aug. 2018. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/
[3] Y. Umuroglu *et al.*, "FINN: A framework for fast, scalable binarized neural network inference," *CoRR*, vol. abs/1612.07119, pp. 1–10, Dec. 2016.
[4] M. Courbariaux and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," *CoRR*, vol. abs/1602.02830, pp. 1–11, 2016.
[5] J. W. Lockwood, A. Gupte, N. Mehta, M. Blott, T. English, and K. Vissers, "A low-latency library in FPGA hardware for high-frequency trading (HFT)," in *Proc. IEEE 20th Annu. Symp. High-Perform. Interconnects (HOTI)*, Aug. 2012, pp. 9–16.

[6] (Apr. 2016). *Intel Begins Shipping Xeon Chips With FPGA Accelerators*. [Online]. Available: http://www.eweek.com/servers/intel-begins-shipping-xeon-chips-with-fpga-accelerators.html

[7] PCI-SIG. (Nov. 2010). *PCI Express Base Specification Revision 3.0*. [Online]. Available: http://composter.com.ua/documents/PCI_Express_Base_Specification_Revision_3.0.pdf

[8] Intel. (2009). *An Introduction to the Intel QuickPath Interconnect*. [Online]. Available: http://www.intel.com/content/www/us/en/io/quickpath-technology/quick-path-interconnect-introduction-paper.html

[9] A. Putnam *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, Jun. 2014.

[10] A. M. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2016, pp. 1–13.

[11] L. Ling *et al.*, "High-performance, energy-efficient platforms using in-socket FPGA accelerators," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA)*, New York, NY, USA, 2009, pp. 261–264.

[12] P. Chow, "Why Put FPGAs in your CPU socket?" in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2013, p. 3.

[13] C. Steffen and G. Genest, "Nallatech in-socket FPGA front-side bus accelerator," *Comput. Sci. Eng.*, vol. 12, no. 2, pp. 78–83, Mar. 2010.

[14] M. Gémieux, Y. Savaria, J.-P. David, and G. Zhu, "A cache-coherent heterogeneous architecture for low latency real time applications," in *Proc. IEEE 20th Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2017, pp. 176–184.

[15] D. Koeplinger, R. Prabhakar, Y. Zhang, C. Delimitrou, C. Kozyrakis, and K. Olukotun, "Automatic generation of efficient accelerators for reconfigurable hardware," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 115–127.

[16] P. Colangelo, E. Luebbers, R. Huang, M. Margala, and K. Nealis, "Application of convolutional neural networks on Intel Xeon processor with integrated FPGA," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2017, pp. 1–7.

[17] S. Krishnan, P. Ratusziak, C. Johnson, D. Moss, and S. Subhaschandra. *Accelerator Templates and Runtime Support for Variable Precision CNN*. [Online]. Available: https://parasol.tamu.edu/pact17/Sri-architecture-templates-runtime_final.pdf

[18] Intel. (2007). *Intel QuickAssist Technology Accelerator Abstraction Layer (AAL)*. [Online]. Available: https://blog-assets.oss-cn-shanghai.aliyuncs.com/18951/6103fadf4dd3a0dfbd0d637308a94b8e99e799d2.pdf

[19] Intel. (Feb. 2017). *Open Programmable Acceleration Engine (OPAE) C API Programming Guide*. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ug/opae-programming-guide.pdf

[20] *SDAccel Development Environment*. [Online]. Available: https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[21] Intel. *Intel FPGA SDK for OpenCL—Overview*. [Online]. Available: https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html

[22] Y.-K. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "A quantitative analysis on microarchitectures of modern CPU-FPGA platforms," in *Proc. 53nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[23] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel, "CAPI: A coherent accelerator processor interface," *IBM J. Res. Develop.*, vol. 59, no. 1, pp. 7:1–7:7, Jan. 2015.

[24] *FreeRTOS—Market Leading RTOS (Real Time Operating System) for Embedded Systems with Internet of Things Extensions*. [Online]. Available: http://www.freertos.org/index.html

[25] P. Laplante, *Real-Time Systems Design and Analysis*. Hoboken, NJ, USA: Wiley, 2004.

[26] B. K. Kim and K. G. Shin, "Scalable hardware earliest-deadline-first scheduler for ATM switching networks," in *Proc. 18th IEEE Real-Time Syst. Symp.*, Dec. 1997, pp. 210–218.

[27] P. Kuacharoen, M. A. Shalan, and V. J. Mooney, III, "A configurable hardware scheduler for real-time systems," in *Engineering of Reconfigurable Systems and Algorithms*. 2003, pp. 96–101.

[28] N. Gupta, S. K. Mandal, J. Malave, A. Mandal, and R. N. Mahapatra, "A hardware scheduler for real time multiprocessor system on chip," in *Proc. 23rd Int. Conf. VLSI Design*, Jan. 2010, pp. 264–269.

[29] D. Gregorek, C. Osewold, and A. Garcia-Ortiz, "A scalable hardware implementation of a best-effort scheduler for multicore processors," in *Proc. Euromicro Conf. Digit. Syst. Design*, Sep. 2013, pp. 721–727.

[30] M. Gémieux, "Analyse de faisabilité de l'implantation d'un protocole de communication sur processeur multicoeurs," M.S. thesis, École Polytech. Montréal, Montreal, QC, Canada, Apr. 2015.

[31] R. Mancuso, P. Srivastava, D. Chen, and M. Caccamo, "A hardware architecture to deploy complex multiprocessor scheduling algorithms," in *Proc. IEEE 20th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 2014, pp. 1–10.

[32] Intel. (Sep. 2017). *Intel FPGA IP Core Cache Interface (CCI-P)*. [Online]. Available: https://01.org/sites/default/files/downloads/opae/intel-fpga-ip-cci-p-inter-spec-external-0.5.pdf

[33] *Universitat Paderborn—University*. [Online]. Available: http://www.uni-paderborn.de/en/university/

[34] Intel, "Intel accelerator functional unit (AFU) simulation environment (ASE)-user guide," Intel, Santa Clara, CA, USA, Tech. Rep. UG-20091. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug-ase.pdf

[35] (Oct. 2017). *Questa Advanced Simulator—Mentor Graphics*. [Online]. Available: https://www.mentor.com/products/fv/questa/

[36] *Intel Quartus Prime Software—What's New in Quartus Prime*. [Online]. Available: https://www.altera.com/products/design-software/fpga-design/quartus-prime/what-s-new.html

[37] J. W. Haskins and K. Skadron, "Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Mar. 2003, pp. 195–203.

[38] *Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 V4 Family | Intel Software*. [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[39] F. Cerqueira and B. Brandenburg, "A comparison of scheduling latency in linux, PREEMPT-RT, and LITMUS RT," in *Proc. 9th Annu. Workshop Operating Syst. Platforms Embedded Real-Time Appl.*, Paris, France, 2013, pp. 19–29.

[40] M. Aichouch J.-C. Prévotet, and F. Nouvel, "Evaluation of the overheads and latencies of a virtualized RTOS," in *Proc. 8th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2013, pp. 81–84.

[41] X. Lu, H. Shi, D. Shankar, and D. K. D. K. Panda, "Performance characterization and acceleration of big data workloads on OpenPOWER system," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 213–222.

[42] S. Nakamoto. (2009). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[43] G. Becker. *Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis*. [Online]. Available: https://www.emsec.rub.de/media/crypto/attachments/files/2011/04/becker_1.pdf

[44] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2084–2123, 3rd Quart., 2016.

[45] (May 2018). *Wiki: The Ethereum Wiki*. [Online]. Available: https://github.com/ethereum/wiki

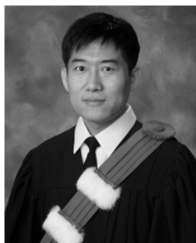
[46] S. Noether, S. Noether, and A. Mackenzie. (Sep. 2014). *A Note on Chain Reactions in Traceability in CryptoNote 2.0*. [Online]. Available: https://lab.getmonero.org/pubs/MRL-0001.pdf

[47] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," in *Proc. IEEE Comput. Soc. Annu. Symp. Emerg. VLSI Technol. Archit. (ISVLSI)*, Mar. 2006, pp. 317–322.

MICHEL GÉMIEUX received the B.E. degree in microelectronics engineering from the University of Quebec at Montreal, Montreal, Canada, in 2012, and the M.S. degree in electrical engineering from Polytechnique Montréal, Montreal, in 2015, where he is currently pursuing the Ph.D. degree in electrical engineering.

His research interests include the development of hardware accelerators in high performance computing servers, techniques for latency reduction of Linux kernel and software, as well as the study of hardware implementations for machine learning applications.

Mr. Gémieux's awards and honors include the Hydro-Quebec's Scholarship of Excellence, the Schneider Electric Student Merit Award (EFC), the Student Merit Scholarship (EFC), and the Doctoral Scholarship (CTM), and was a PERSWADE Trainee (CRSNG) throughout his Ph.D.



MENG LI received the B.E. and M.S. degrees in electronic engineering from the Beijing University of Aeronautics and Astronautics, Beijing, China, in 2004 and 2007, respectively, and the Ph.D. degree in electrical engineering from Polytechnique Montréal, Montreal, QC, Canada, in 2016. Since 2016, he has been with the Department of Electrical Engineering, Polytechnique Montréal, as a Post-Doctoral Fellow. Since 2017, he has been with MISTLab dedicated to the Department of Computer Engineering, Polytechnique Montréal, as a Post-Doctoral Researcher Fellow. Currently, he is also an Associate Professor with the School of Information and Science Technology, Zhejiang Sci-Tech University, Hangzhou, China. His research interests include mobile robot localization, swarm robot control, communication networks, task scheduling, fault tolerance, parallel computing, and real-time systems.

YVON SAVARIA (S'77–M'86–SM'97–F'08) received the B.Ing. and M.Sc.A. degrees in electrical engineering from Polytechnique Montréal in 1980 and 1982, respectively, and the Ph.D. degree in electrical engineering from McGill University in 1985. Since 1985, he has been with Polytechnique Montréal, where he is currently a Professor with the Department of Electrical Engineering.

He has carried work in several areas related to microelectronic circuits and microsystems, such as testing, verification, validation, clocking methods, defect and fault tolerance, effects of radiation on electronics, high-speed interconnects and circuit design techniques, CAD methods, reconfigurable computing and applications of microelectronics to telecommunications, aerospace, image processing, video processing, radar signal processing, and digital signal processing acceleration. He is currently involved in several projects that relate to aircraft embedded systems, green IT, wireless sensor networks, virtual networks, computational efficiency, and application specific architecture design. He holds 16 patents, and he has published 140 journal papers and 440 conference papers. He was the thesis advisor of 160 graduate students who completed their studies.

He has been a Consultant or was sponsored for carrying research by Bombardier, CNRC, Design Workshop, DREO, Ericsson, Genesis, Gennum, Huawei, Hyperchip, ISR, Kaloom, LTRIM, Miranda, MiroTech, Nortel, Octasic, PMC-Sierra, Technocap, Thales, Tundra, and VXP. He is a member of the Regroupement Stratégique en Microélectronique du Québec, Ordre des Ingénieurs du Québec. He has been a member of the CMC Microsystems Board since 1999 and was the Chairman of that board from 2008 to 2010. He was the Tier 1 Canada Research Chair (www.chairs.gc.ca) on design and architectures of advanced microelectronic systems from 2001 to 2015. He also received the Synergy Award of the Natural Sciences and Engineering Research Council of Canada in 2006.

JEAN-PIERRE DAVID (M'05) received the Ph.D. degree from the Université Catholique de Louvain, Louvain-la-Neuve, Belgium, in 2002. He has been an Assistant Professor with the Université de Montréal, Montreal, QC, Canada, for three years and moved to Polytechnique Montréal, Montreal, in 2006, where he has been an Associate Professor since 2013. His research interests include digital system design, reconfigurable computing, high-level synthesis, and their applications.

GUCHUAN ZHU (M'07–SM'12) received the M.S. degree in electrical engineering from the Beijing Institute of Aeronautics and Astronautics, Beijing, China, in 1982, the Ph.D. degree in mathematics and control from the École des Mines de Paris, Paris, France, in 1992, and the Graduate Diploma degree in computer science from Concordia University, Montreal, QC, Canada, in 1999.

He joined Polytechnique Montréal, Montreal, in 2004, where he is currently a Professor with the Department of Electrical Engineering. His current research interests include control of distributed parameter systems, nonlinear and robust control, and optimization with their applications to microsystems, aerospace systems, communication networks, and smart grid.