The Difficult Balance Between Modern Hardware and Conventional CPUs

The Difficult Balance Between Modern Hardware and Conventional CPUs

Fabio Maschi
Systems Group, Dept. of Computer Science
ETH Zurich, Switzerland

Gustavo Alonso
Systems Group, Dept. of Computer Science
ETH Zurich, Switzerland

ABSTRACT

Research has demonstrated the potential of accelerators in a wide range of use cases. However, there is a growing imbalance between modern hardware and the CPUs that submit the workload. Recent studies of GPUs on real systems have shown that many servers are often needed per accelerator to generate a high enough load so the computing power is leveraged. This fact is often ignored in research, although it often determines the actual feasibility and overall efficiency of a deployment. In this paper, we conduct a detailed study of the possible configurations and overall cost efficiency of deploying an FPGA-based accelerator on a commercial search engine. First, we show that there are many possible configurations balancing the upstream system and the way the accelerator is configured. Of these configurations, not all of them are suitable in practice, even if they provide some of the highest throughput. Second, we analyse the cost of a deployment capable of sustaining the required workload of the commercial search engine. We examine deployments both on-premises and in the cloud with and without FPGAs and with different board models. The results show that, while FPGAs have the potential to significantly improve overall performance, the performance imbalance between their host CPUs and the FPGAs can make the deployments economically unattractive. These findings are intended to inform the development and deployment of accelerators by showing what is needed on the CPU side to make them effective and also to provide important insights into their end-to-end integration within existing systems.

CCS CONCEPTS

• **Hardware → Hardware accelerators**; • **Computer systems organization → Cloud computing**; *Client-server architectures*; *Heterogeneous (hybrid) systems.*

KEYWORDS

FPGA, distributed systems, hardware accelerators, cloud computing

ACM Reference Format:
Fabio Maschi and Gustavo Alonso. 2023. The Difficult Balance Between Modern Hardware and Conventional CPUs. In *19th International Workshop on Data Management on New Hardware (DaMoN '23), June 19, 2023, Seattle, WA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3592980.3595314

1 INTRODUCTION

Hardware accelerators are becoming increasingly available in data centres and cloud platforms, introducing a number of heterogeneous processing elements alongside CPUs [1, 9, 12, 26]. Many studies have shown their potential to accelerate a variety of use cases, often on the basis of the much higher throughput that such devices are able to sustain, be it in terms of requests or gigabytes per second. However, such throughput can only be achieved if enough load can be submitted to the accelerator, which is usually beyond the capacity of a single CPU server. In several use cases, a single accelerator node can replace many CPU-based servers, but then the host CPU they are attached to and the upstream system often cannot generate a sufficient high load for the accelerator. The situation creates an optimisation dilemma: as the accelerator improves and yields higher throughput, more CPUs are required to feed enough input for the accelerator to work efficiently.

This imbalance is pervasive and often quite large. For instance, a recent study by Meta of their ML pipelines [33] shows that GPUs used for training ML models are stalled up to 56 % of the time waiting for input data. They also show the increasing amount of compute power, network, and memory bandwidth needed on the CPU side to be able to match the throughput of the accelerator. A number of other studies have also explored this problem [11, 17, 32] confirming that the advantages a hardware accelerator can bring are bound by the ability to generate enough load on it, often leading to a situation where the accelerated system is larger than the initial one. This issue is one of the reasons why new processor architectures are emerging that try to avoid these bottlenecks [7, 8] and new standards for peripheral interconnects are appearing [6, 24, 30].

In this paper, we study this problem in the context of FPGAs. We use a commercial search engine and an FPGA-based accelerator designed for it [21, 25], and explore the interplay between accelerator configuration and requirements on the input throughput. Notably, we demonstrate that the proposed design is capable of accommodating a wide range of trade-offs between latency and throughput, as the fastest solution is not necessarily the most effective one in practical scenarios. We also look into a reasonable estimate of the cost of a deployment involving the accelerators in a variety of settings: data centre, cloud, and with different FPGAs. The results are sobering in the sense that they show how the performance imbalance between accelerator and the rest of the cloud infrastructure limits the potential advantages of the FPGA.

Taken together, these insights provide a very accurate picture of the challenges that arise when trying to deploy accelerators in general, and an FPGA in particular into an existing commercial system. By describing in detail the interactions between the software and the FPGA, the use case we explore can help inform further developments regarding FPGA deployments within distributed systems, as several of the issues we have encountered are structural limitations that will arise in other systems as well. Our results suggest that successful FPGA deployments require a careful software-hardware co-design of the entire system, as approaches that only consider the FPGA side of the system are limited by many of the same design considerations discussed in this paper.

2 BACKGROUND

The starting point for this study is the flight search engine offered by Amadeus to travel companies. Like many other search engines, Amadeus' application is a large-scale distributed system comprising many different components. When a user request looking for flights needs to be processed, a large number of potential routes have to be computed. For all non-direct routes, the Minimum Connecting Time (MCT) between two flights must be ascertained. The MCT module is implemented atop a Business Rule Management System, and is used in the early stages of the search (the *Domain Explorer* component), playing a key role in the performance and total cost of operating the search engine.

The initial implementation of MCT on FPGA (ERBIUM[1][21]) proved to be a significant improvement over the existing system along several dimensions. The proposed solution translates the business rules into a concise Non-deterministic Finite State Automaton (NFA), and exploits the inherent parallelism and pipeline possibilities available on the FPGA to simultaneously traverse active states within the graph. In contrast, when performed on a CPU, the search operation becomes memory bound due to the unpredictable nature of the traversal and the increased likelihood of cache-misses resulting from random memory access.

ERBIUM processes queries three orders of magnitude faster than the fastest CPU implementation, with a throughput of up to 50 million queries per second, 800 times greater than a single CPU instance. Due to this very high throughput, the cost of running ERBIUM in the cloud is 60 billion queries per USD, 20 times cheaper than the most cost-effective CPU deployment. Updating the rules incurs only 500 µs of downtime, 4 orders of magnitude faster than the current procedure, which improves the overall availability of the search engine.

The performance gains and the architectural flexibility offered by FPGAs opened up many different possibilities in terms of how to take advantage of these significant gains. The performance improvements could be leveraged, e.g., to enhance the search quality by processing a larger input than in the current system; to reduce the amount of compute nodes needed to meet the performance requirements; and/or to increase the architecture's flexibility by introducing a new microservice in the engine separated from the other components. In [21, 25] we provide more details and a comprehensive discussion of these possibilities within the flight search engine.

3 SYSTEM INTEGRATION

In this section, we discuss the integration of the hardware accelerator into the existing system and how assumptions made in software and available interfaces impact the resulting performance.

[1]ERBIUM is open source: github.com/fpgasystems/erbium

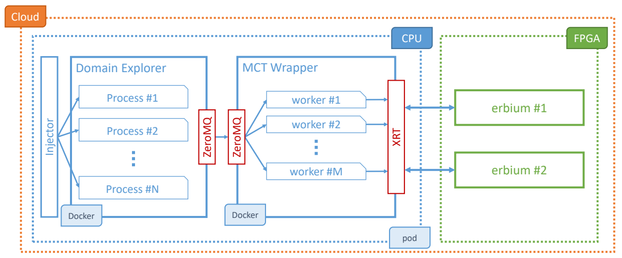

Figure 1: System Integration setup.

3.1 Setup

We integrated ERBIUM into a deployment of the Amadeus flight search engine on AWS F1 instances. The structure of the system is illustrated in Figure 1, and its main components are as follows:

The **Injector** module generates the workload fed to the system by replaying user request traces captured from the real production environment. It distributes the workload to each one of the *Domain Explorer* processes, saturating their computing capacity as much as possible.

A **Domain Explorer** process executes a user request in its entirety. Each user request spawns a variable number of queries to ERBIUM. The number of processes deployed in one machine depends on the memory and number of cores available. In the current setup, there are 48 *Domain Explorer* processes per node and 400 nodes.

The communication between the *Domain Explorer* processes and the *MCT Wrapper* uses the **ZeroMQ** framework, a light-weight networking library that handles concurrent communications using different patterns and protocols. It was already validated and deployed for similar purposes in other Amadeus applications, which facilitated its adoption in this solution. We use a Request-Reply pattern combined with synchronous communications between the *Domain Explorer* processes and the router, and asynchronous communications between the router and the multiple workers.

The **MCT Wrapper** is a multi-threaded module that distributes the incoming queries among the different workers, which uses ZeroMQ *dealers* to support asynchronous communication to ensure the wrapper is always ready to process new queries. Given that libraries used for interfacing the FPGA via PCIe are vendor-specific, the wrapper plays a crucial role in abstracting the FPGA intrinsics by exposing a microservice interface to the rest of the system, thus creating a clear boundary to the tight coupling to specific vendor's tools or boards. In case of a transition from, e.g., Xilinx to Intel FPGA boards, any modifications would be encapsulated within the *MCT Wrapper* as a new docker image. This wrapper is also responsible for managing ERBIUM-specific start-up and update operations, such as programming the device and loading the static input data into the FPGA's internal memory.

The current ERBIUM implementation uses dictionary encoding to reduce both the storage requirement and the online data movement. Therefore, queries must be encoded before being sent to the accelerators, just like data quantisation and normalisation in machine learning pipelines [33]. This process is carried out individually at the worker level in a pipeline manner, while the previous query batch is being executed by the FPGA kernel. The significance of

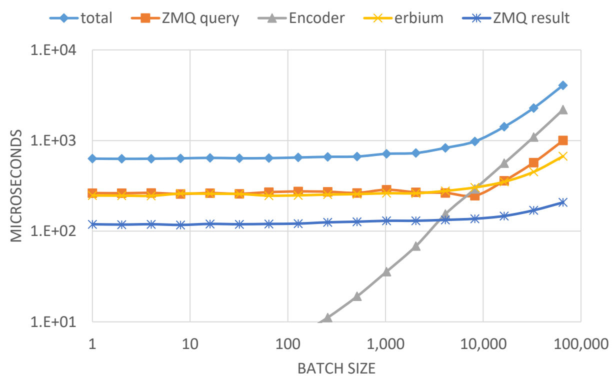

Figure 2: Latency breakdown of a function invocation.

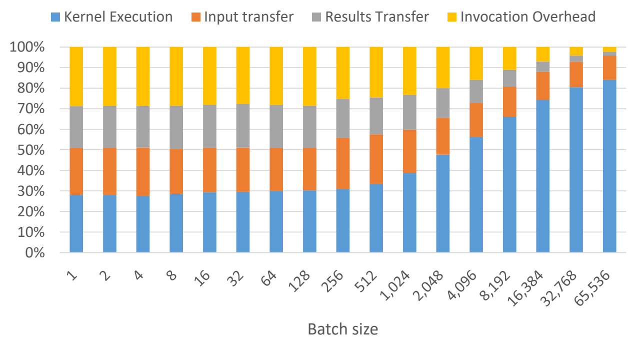

Figure 3: Latency breakdown of the ERBIUM invocation.

this module lies in its ability to transform software data representations into formats that are more suitable for FPGA processing. Its absence would hinder the competitiveness of the FPGA design as it would be mired in the overhead of parsing and processing complex data types.

The communication between the different workers and the FPGA kernels is managed by the Xilinx Runtime Library (**XRT**). It schedules data movement and execution so that while the kernel is processing a batch of queries, a different thread is being served by transferring its query data into the FPGA internal memory. We leverage this mechanism by having at least two threads per FPGA kernel.

3.2 System Overhead Characterisation

We evaluate the simplest scenario where a single *Domain Explorer* process generates queries, a single *Worker* handles data encoding and communication with the FPGA, and a single ERBIUM kernel evaluates the queries. In this configuration, we measure the response time of each element for varying query batch sizes. Reported numbers correspond to the 90th percentile of a series of 1000 iterations.

Figure 2 shows the communication and synchronisation overheads of the different layers of the architecture as a function of the batch size. Due to PCIe bus constraints and FPGA shell protocols, data movement overheads dominate the actual processing time for small batch sizes up to 4096 queries, an input payload of 2 MiB. For larger batch sizes, the encoder introduces a linear and significantly high execution time, exceeding the actual kernel invocation time of ERBIUM. ZeroMQ communication overheads are also noteworthy, accounting for between 60 % to 30 % of the total response time for both query and response data movements respectively. Since the kernel execution time on the FPGA is in the microsecond range, data encoding and data movements become equally or more expensive, a well-known issue in data centre applications [2].

The relative latency incurred inside the *MCT Wrapper* worker for data transfer to and from the FPGA, as well as the *XRT* invocation overheads and the actual kernel execution are plotted in Figure 3. In absolute numbers, the ERBIUM kernel execution time can be as low as 70 µs for batches up to 64 queries, a payload of 32 KiB. Up to this batch size, the invocation overhead, which represents the synchronisation time between host CPU and the accelerator carried out by *XRT*, can be as high as the kernel execution time. The execution latency only becomes dominant for batch sizes equal or greater than 2048 queries, or 1 MiB of input payload. In Section 4, we explore the challenges and current approaches to overcome the PCIe bandwidth limitation in the context of small query payload sizes. We expect that future interconnects such as CXL 3.0 will provide a much higher bandwidth and reduced latency, which will address this concern in the current design.

3.3 Parallel Evaluation

The configuration depicted in Figure 1 is highly amenable to parallelisation, which affords a wide range of deployment choices for each component, but may also crease disparities in workload distribution across the system. To achieve optimal performance, it is necessary to select the most appropriate parameters for each component, taking into account both the latency threshold from SLAs and system utilisation. The system's overall throughput is measured in queries per second, and the execution time of a user request (i.e., a batch of queries) is measured as seen by the *Injector*. In the graphs that follow, series are labelled according to the number of processes (p), number of workers (w), number of kernels (k) and number of engines (e) per kernel.

Varying the number of engines per kernel The number of kernels that can be accommodated on a single FPGA is contingent upon the finite physical resources available on the board and the number of engines they have. The FPGA can fit one *erbium* kernel with four parallel engines, two kernels with two engines each, or four kernels with one engine each. To evaluate the individual throughput gain for a single kernel, we measure performance while varying the number of engines within it, while keeping the upstream system fixed. Figure 4 plots the global throughput of the system when deploying kernels with one, two, and four engines. By design, a kernel allocates the workload of a single user request to all available engines. Increasing the number of engines, thus, reduces the request execution time, resulting in increased throughput in a single-process, single-worker setup. The circuit complexity increases with the number of engines, leading to a 30 % lower operating frequency. As a result, performance metrics do not scale linearly with the number of engines.

Varying the number of parallel components uniformly In a complementary analysis to the previous experiment, we keep

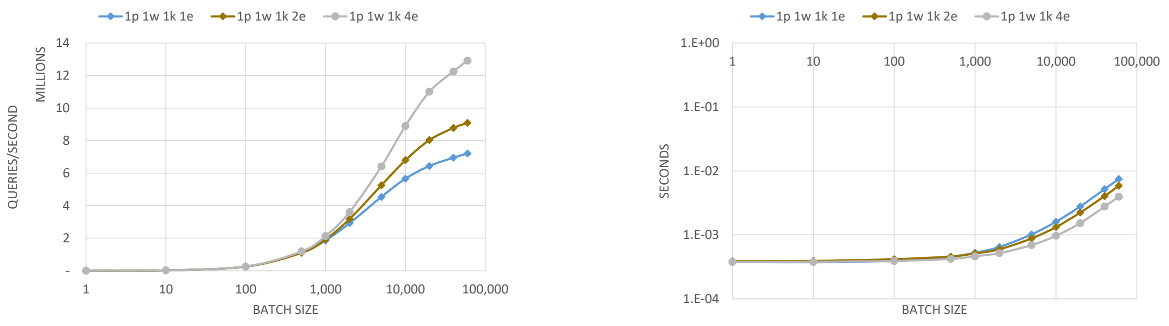

Figure 4: Varying the number of engines per kernel.

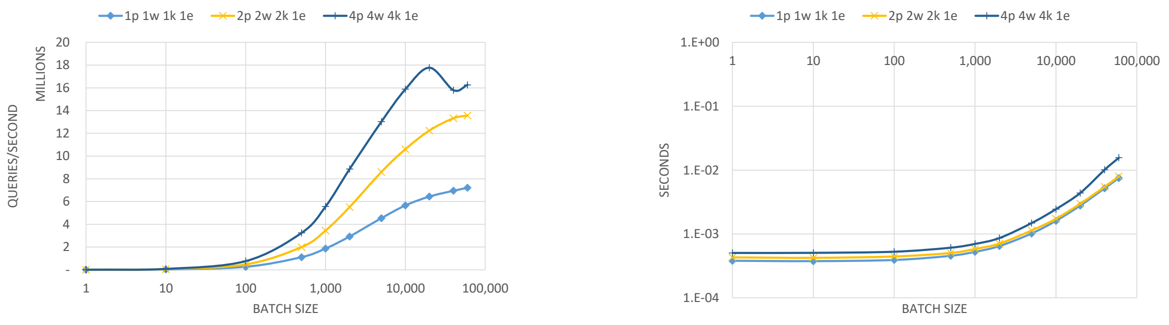

Figure 5: Varying the number of parallel components uniformly.

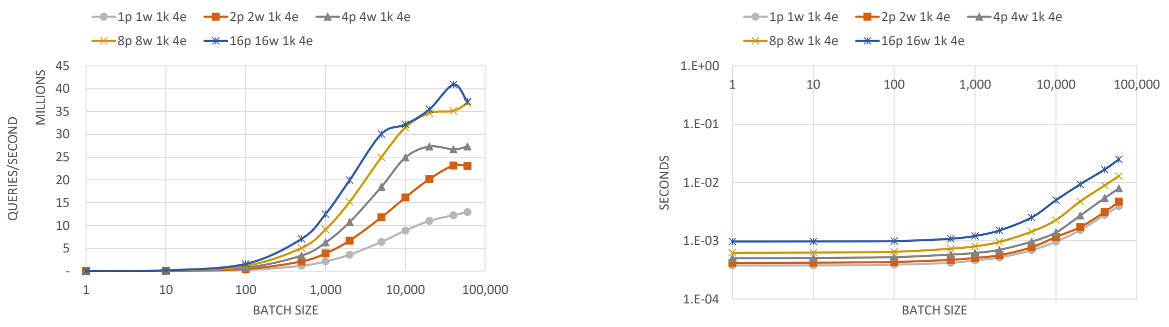

Figure 6: Multiple Process-Worker couple for a single kernel.

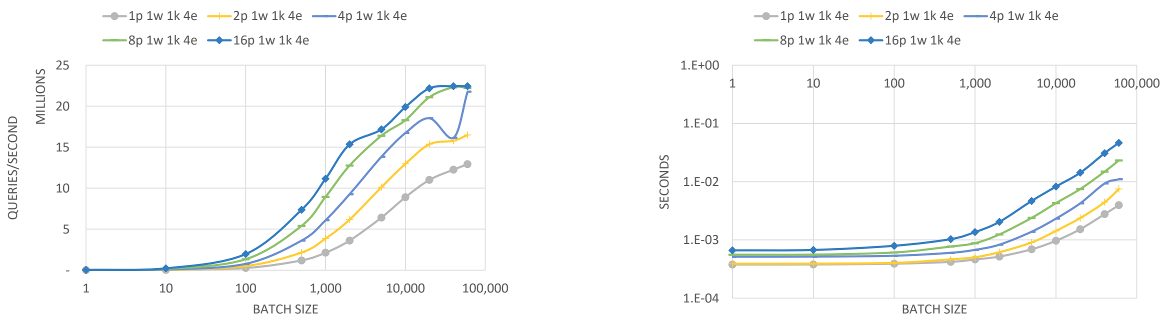

Figure 7: Multiple processes per worker.

the number of engines per kernel fixed and assess the impact of adding new kernels on performance. Each kernel has its own feeding flow, i.e., one process and one worker each. The system capacity to process parallel requests increases as we add parallel processing elements. However, the complexity of the FPGA circuits leads to a slower operating frequency, resulting in slightly increased latency per request. In this setup, global throughput is prioritised at the cost of a slower execution time per request (compare Figure 4 with lower throughput, and Figure 5 with higher throughput but also higher latency).

Multiple Process-Worker couple for a single kernel We measure the computing capacity of a single kernel as the upstream workload increases. Given that the FPGA kernel stand-alone outperforms the global throughput, the stress test of this experiment focuses on how much synchronisation overhead the *XRT* driver imposes, since it schedules different calls from different workers down to a single kernel. As shown in Figure 6, this configuration maximises the global throughput, with a peak of 40 million queries per second. However, the synchronisation overhead at the *XRT* scheduler imposes a latency proportional to the number of feeding threads, while remaining constant with respect to the batch size.

Multiple processes per worker We stress the parallel workers by varying the number of processes a single worker is fed from. In this setup, the worker is responsible for scheduling different user requests and batching them into a single ERBIUM call. The kernel configuration is kept constant as the best-case setup from the first experiment, with four engines per kernel. As depicted in Figure 7, a single worker is not saturated by a single process and can deliver a higher throughput when coupled with several processes, e.g., when connecting from 2 to 8 processes per worker. The gain decreases as we approach 16 processes per worker, indicating saturation at the worker level. In terms of execution time, the scheduling at the worker level imposes a similar latency than the one imposed at the *XRT* level, but the former does depend on the batch size.

Comparison Figure 8 presents the compromise between latency and throughput, and indicates which would be the best configuration according to the performance priority. For a given minimal throughput, e.g., 20 million queries per second, one can identify that a configuration with `4p 4w 1k 4e` would impose the lowest execution time; however if the maximum execution time is fixed at, e.g., 500 µs, the configuration with `2p 2w 1k 4e` would be the one which yields the best throughput. Having such a flexibility in the configuration of the system and the degrees of parallelism is crucial in search engines. Additional performance demands, whether higher throughput or lower latency, can be addressed by choosing the right combination of parameters. This flexibility also helps avoiding over-provisioning by adjusting the size of the system to the target performance and in determining what element to scale when needed.

3.4 Discussion

When configuring a system with parallel processing elements, it is important to consider not only the direct gain in throughput, but also how the different elements are interconnected [3, 23]. Moreover, the trade-off of generality and easy of maintenance at the cost of designs that do not achieve maximum performance is a common one in software, but one that still needs to be considered in many FPGA-based systems where performance often trumps all other considerations [13]. As our use case illustrates, the practical viability of a design hinges not only on its performance, but also on its ability to match changing performance targets.

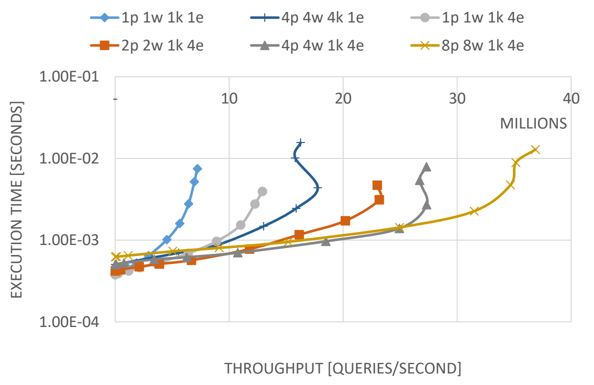

Figure 8: A *pareto* comparison of execution time as a function of the throughput for selected configurations.

The performance comparison shows the importance of considering the life cycle of a real system of this scale. In practice, the load on all components tends to grow over time. Thus, a design whose performance is maximised for a particular problem size might not be the best option on the medium and long term. Instead, a flexible design is preferred as it facilitates maintenance and evolution. The choice of using *ZeroMQ* as the middleware between the *MCT Wrapper* and the *Domain Explorer* emphasises the significance of easy maintenance. Exposing it as a library, rather then using a microservice architecture, would reduce isolation and increase technology stack dependencies. Given that the *MCT Wrapper* encompasses vendor-specific implementations and has a limitation in terms of threads interfacing *XRT*, the reduced messaging overhead of using it as a library would incur an extreme development and maintenance cost, exposing software developers to the intricacies of FPGA development. It is crucial to consider the scalability and adaptability of a system design over time as a key takeaway. Ultra-optimised designs that require being re-designed from scratch to accommodate changes are unlikely to be successful in real-world deployments.

From the research and practitioners perspective, these results yield an important lesson, which is that the overall performance is determined more by how the accelerator is integrated rather than the FPGA design itself. As evidenced by the performance measurements, there is a surprisingly large range of setups that can achieve varying levels of latency and throughput as a result of constraints imposed by the system surrounding the FPGA. These findings support the conclusions drawn earlier that the absolute performance on the FPGA is not the only factor to consider, and that aspects such as flexibility, ease of maintenance, evolution, and integration are equally important.

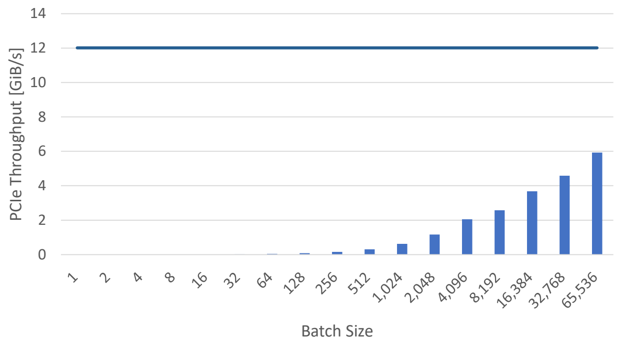

Figure 9: PCIe throughput of data transfer from host to device memory.

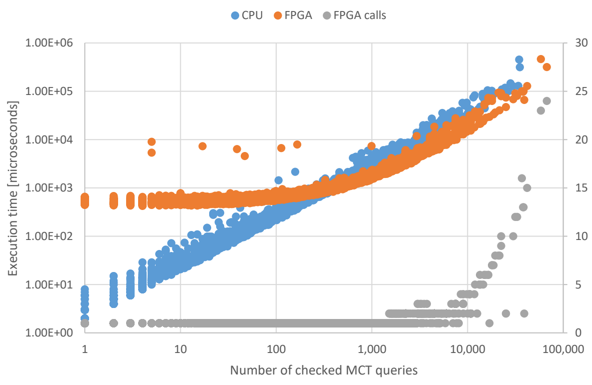

Figure 10: Execution time comparison of CPU and FPGA processing user requests.

4 INVOCATION ANALYSIS

In this section, we explore the performance implications associated with the function invocation of the accelerator, specifically from the perspective of a top-level user request.

4.1 Optimising the input channel

The various setups seen in the previous section stress different components of the system, each one presenting different overhead constraints. Maximising global throughput can come at the cost of longer execution time per user request, while prioritising fast response time may lead to lower throughput. It is important to highlight that this is not a characteristic imposed by the accelerator, but rather by the PCIe bus and the existing synchronisation protocols, which require batching thousands of queries in a single function invocation to be efficient. Batching is a standard workaround for PCIe-attached accelerators to achieve high throughput [3, 16, 25], but it comes at the cost of strict higher latency to the individual queries composing the batch.

Different communication interfaces might play a key role in alleviating this effect, such as streaming [14], where queries can be fed to the accelerator, executed, and returned while the rest of the stream is still being processed. Future developments of interconnects, such as CXL and NVLink, and accelerator interfaces have the potential of lifting the current limitations and enable new use cases [15, 17]. Notably, GPU starvation by lack of incoming data from the CPU is one of the motivations for NVIDIA's Grace CPU architecture, which provides a much higher bandwidth between CPU and GPU than conventional approaches [7, 8].

Differently from GPUs, which by essence process a large amount of data in a *single-instruction-multiple-data* fashion, FPGA applications do not necessarily need to follow the same pattern. FPGA kernels can be designed as a very precise data-flow, where multiple pieces of data are executed independently in parallel. This is the case of ERBIUM, where the query payload is very small (64 B) and the internal graph search [21] is carried out in a *multiple-instruction-multiple-data* fashion. As a consequence, the ratio of memory bandwidth versus computation is flagrantly different compared to GPUs and to how the PCIe bus is traditionally intended to be used [33]. Figure 9 shows the transfer speed of input data, from the host to the device memory. This bandwidth does not depend on the accelerator, but solely on the PCIe bus, device memory characteristics and on the *XRT* driver. Batches with small number of queries are not sufficient to reach a high enough throughput over PCIe. Notice that not even packing 65 536 queries in the same batch, as a payload of 4 MiB, barely reaches half of the maximum speed, so the *XRT*'s DMA engine is always operating below PCIe bandwidth capacity. As seen in the previous section, attempts to achieve higher throughput by batching more queries together incurs higher individual query latency, which is a key performance metric in an online system such as Amadeus' flight search engine.

4.2 Related Query Batching

To mitigate the latency versus throughput dilemma illustrated in Figure 9, each FPGA invocation should be optimised in such a manner that a maximum number of *related* queries are batched together. In our use case, a top-level user request spawns a variable number of ERBIUM queries in the *Domain Explorer*, which can usually range from zero to about 6000 queries, but it is only known during runtime. The current implementation of the *Domain Explorer* is optimised for a CPU architecture, where the notion of batch processing is not required. In this way, the module executes queries sequentially, and stops the search as soon as a sufficient number of valid results are found. In the case of the FPGA, where there is a need of batching, the number of queries needed to be checked is not known *before* aggregating them into an FPGA invocation.

To measure how efficient the current CPU implementation is regarding the evaluation of independent user requests, we conduct an experiment deploying a single *Domain Explorer* process, a single *MCT Wrapper*, and a single ERBIUM kernel with four engines. In this experiment, instead of imposing the batch size, we rely on the workload of individual user requests generated by the *Domain Explorer*. For the FPGA flow, if the number of queries in the first invocation is not enough to satisfy the user request, a new invocation is performed with further queries.

Figure 10 plots the execution time of 6300 individual user requests being processed by the CPU and the FPGA modules as a function of the number of checked queries, which is determined at runtime. We also plot the number of FPGA calls required to fully complete the execution. Given that the CPU is able to finish a user request as early as it finds enough valid queries, but the FPGA invocation is carried out by batches, the FPGA is penalised by having to process on average 23 % more queries than the CPU. For small workloads of up to 400 queries, the CPU implementation presents a smaller execution time. For bigger workloads, the FPGA outperforms the CPU even when called several times in the same request. These latency numbers, however, are not close to the performance speed-up originally achieved when not taking into account the origin of queries and their relation [21].

4.3 Discussion

FPGAs are fundamentally different from CPUs and GPUs. While in all cases a certain amount of tuning to the underlying hardware makes sense, in the case of the FPGAs it is usually crucial to get the necessary performance. We have already seen this effect when considering the impact of the encoder in the overall overhead, which is necessary to adapt the incoming data types to formats the FPGA can process efficiently. In this section, the experimental results indicate that the batch size has to be large enough for the FPGA to be competitive. However, the batch size is dictated by the search engine and how it works. We have been able to find a compromise solution, but the overall result is that the performance on the FPGA suffers. At small batch sizes, there is not enough load for the FPGA to be fully exploited, and the overhead of sending the queries to the FPGA dominates. The size of the batch that can be processed is determined by the business logic of the flight search engine. We can delay submitting queries to batch several requests, but that has an impact on latency and requires additional logic before the FPGA. Moreover, not all user requests result in enough ERBIUM queries, naturally leading to small batch sizes and infra-utilisation of the FPGA. One could argue that the flight search engine could be modified to accommodate the FPGA. This is not a realistic option in the short term as it would be very costly and still does not address the fact that the query load is not uniform across different user requests. These effects play no role in the CPU implementation, but turn the FPGA deployment into a rather complex compromise in terms of design and performance with the added factor that the FPGA cannot always be fully exploited.

5 SYSTEM DEPLOYMENT

In this section, we present a brief analysis of deploying the search engine solution for both on-premises and in the cloud, and an approximate cost that gives an accurate picture of the issues involved.

5.1 Application co-location

The PCIe bus is the traditional interface to interact with FPGAs supported by the major vendors. In this setup, the FPGA acts as a subordinated accelerator attached to a host CPU, which is responsible of provisioning, programming, and executing the FPGA kernels through a vendor-specific driver. In order to maximise server utilisation and reduce communication overheads, upstream applications are often co-located in the host CPU of the FPGA board, which is the case of the Amadeus' flight search engine, as depicted in Figure 1. User requests come from the network and are served by the CPU application, which then generates queries to be accelerated by the FPGA. This setup is the only one available in both AWS F1 and Azure NP-series, as of March 2023.

FPGA wrappers like the *MCT Wrapper* can be seen as a simple middleware, or proxy, between the business application and the accelerator kernel. Its computing demand is very low, so not co-locating any upstream application on the host CPU very often leads to waste of resources. Moreover, the inter-POD communication carried out by *ZeroMQ* is optimised so it is routed internally, without actually going through the network. The latency increase caused by the network communication would only be tolerated if the servers running the upstream application and the host CPU are connected through the same network switch (i.e, top-of-rack switch), which is uncontrollable in cloud deployments. From a throughput perspective, the limit caused by the PCIe-bus and the number of threads that can actually be used per kernel would nonetheless remain constant, making the disaggregation of containers into two PODs unprofitable overall.

5.2 Resource optimisation

Amadeus uses 400 large multi-core servers for the *Domain Explorer*, and the current computing footprint of the MCT module accounts for up to 40 % of CPU time on them. By offloading it to the FPGA, it is theoretically possible to handle 40 % more user requests per server. For a constant global workload of the system, this indicates that the current allocation of CPU servers could drop to only 244 CPU servers attached to an FPGA.

The approach to leverage the new design depends on whether the deployment is on-premises or in the cloud. In on-premises settings, one theoretically has the flexibility to choose a specific combination of CPU and FPGA boards. However, in practice, this choice is driven by not only the acquisition costs, but also the compatibility with existing hardware and the technical maintenance overhead within the organisation. For instance, the existing computing resources for the Amadeus' flight search engine are already defined and accounted for. An eventual re-allocation of resources within the organisation should also be considered, e.g, allocating surplus CPUs to a different service or product.

On the other hand, for deployments in the cloud, the choices are limited to the currently available configurations of CPU and FPGA models, FPGA shells (and hence, communication interface such as batching vs. streaming), and how they are attached together. As of March 2023, both the AWS F1 and Azure NP-series instances offer a relatively large FPGA (UltraScale+ VCU9P and Alveo U250, respectively), PCIe-attached to a relatively small CPU (8 and 10 vCPUs, respectively). Although other instances offer further FPGA and CPU resources, the vCPU-to-FPGA ratio remains the same. This is currently a major handicap in a co-located service architecture, since the workload of a few vCPUs is insufficient to fully utilise the computing capacity of the FPGA. More importantly, to match the CPU current load capacity of a single on-premises CPU-only server, approximately six AWS F1 instances would be necessary. Although a possible solution may come from multiple CPU instances sharing the same FPGA, or FPGA-to-FPGA communication, this is not yet available in cloud environments. Thus, while the FPGA can process

Table 1: Cost estimate of the proof-of-concept deployment.

	Element	Units	Unit Cost[2] (USD)	Total (USD)
On-Premises				
Original engine	CPU	400	10 k	4 M
With ERBIUM	CPU + Alveo U200	244	20 k	4.88 M
With ERBIUM	CPU + Alveo U55c	244	13 k	3.17 M
AWS				
Original engine	c5.12xlarge	400	1.452/h	5.0 M/year
With ERBIUM	f1.2xlarge	1,464	1.2266/h	15.7 M/year
Azure				
Original engine	F48s v2	400	1.2084/h	4.2 M/year
With ERBIUM	NP10s	1,171	1.0411/h	10.6 M/year

[2] Hourly prices for AWS instances in Europe (Ireland) region, and for Azure instances in United States (Washington), both for a savings plan of one year.

a large workload, the CPU side is simply not powerful enough to generate enough load.

5.3 Cost efficiency

Table 1 summarises a simplified cost calculation associated to different deployments based on the current production load. This calculation is not intended to provide a comprehensive comparison between on-premises and cloud deployments, since it does not consider factors such as maintenance, facility, energy and operational on-premises costs. This being said, it does provide a good base of comparison for the different system layouts among each category. Notably, these numbers demonstrate that, on-premises, the new design is cost-effective when using some of the more modern FPGAs (U55c) that provide good performance at a much lower price than previous comparable models (U250, U280). In fact, considering that the co-located application cannot fully utilise the FPGA and the hardware design fits on a smaller board, one could consider even smaller FPGA boards to make the system more cost-efficient and increase resource utilisation. On top of that, FPGA models are evolving fast, offering increasing resources at the same price, or even at a lower cost for a fixed capacity (e.g., the U55c is more powerful than the U200 with a far lower list price). On the other hand, for cloud deployments, the current configurations are simply inadequate for the proposed design. Due to the small vCPU-to-FPGA ration available, 1464 *f1.2xlarge* or 1171 *NP10s* would be necessary to sustain the current load on the rest of the system. The cost increase of 3x for AWS and 2.5x for Azure over a CPU-only design is prohibitive.

5.4 Scalability

Equally important to the availability of the provisioning instances, the flexibility of the scalability plays a major role in the deployment of production systems. In fact, despite the relatively higher development and integration efforts, this is what makes distributed systems attractive [5], as they allow applications to be decomposed into smaller, independent computing nodes, enabling fine-grained scaling according to their individual demand. However, the current offering of PCIe-attached FPGAs tends to favour application co-location, which in turn imposes a tight scalability dependency between the module running on the CPU and the accelerated kernel.

The fact that even small models of FPGA boards can consume the workload of several CPU servers stresses the difficult optimisation equation: in the best scenario of smaller FPGAs PCIe-attached to powerful CPU servers, they are expected to scale in and out together. In realistic scenarios, given the current limited cloud offerings, the FPGA nodes in applications such as the Amadeus' flight search engine are certainly under-utilised. This is a key element that deserves more research attention, as one could easily imagine the FPGA-accelerated function as a microservice that can scale in and out independently from the upstream flow, which would significantly enhance its flexibility and scalability. This research direction is being recognised, e.g., from SAP SE, who recently analysed the tightly coupling of accelerators to hosts as an important limiting factor for data processing [22].

As seen before, the accelerator can achieve high throughput or low latency when the workload is simply fed to the system without concerns of relation (i.e, origin user request) or composition (i.e., batching queries from different user requests in a reasonable size). When these business constraints must be respected, both metrics can simply never be met. To solve this problem, it is not sufficient to shuffle around modules into different servers or try to fully optimise some implementations to earn few microseconds. The solution requires a more disruptive change, mainly in what concerns the critical data path that queries navigate through. This can be achieved in two ways. One is through new interconnects such as CXL, which in future versions (3.0) promises a much higher bandwidth and lower latency than existing PCIe solutions [10, 15, 18]. The other is by making the FPGA directly callable through the network, so as to remove the host CPU and the interconnect completely from the interaction.

Until CXL becomes more established and future versions available, the network channel seems the most promising at the moment and could be used in our use case. This would involve making the ERBIUM engine a microservice that be called using a standard cloud interface like HTTP [20]. This can potentially even be done without a regular server such as what has been done in Amazon's AQUA [1]. AQUA allows Redshift to offload advanced query processing to the FPGA via the network to implement a caching layer with SSDs directly attached to the FPGA that is, in turn, directly connected to the network. In this configuration, the ability of the FPGA to process data at line rate is used to maximise the bandwidth of the storage and of the network without requiring a conventional CPU. In such configurations, it is also possible to incorporate additional functionality such as compression and encryption to make the system even more efficient [4].

5.5 Discussion

Although these results are undoubtedly rough cost calculations and some of the prices of servers, boards, and cloud instances can vary in the near future, they do provide a sobering perspective on large-scale FPGA deployments using today's systems. The main issue that these results bring to the fore is the significant imbalance between the CPU and the FPGA in the cloud instances. We have shown throughout the paper that turning the initial prototype into a real system results in a performance loss over the ideal case, performance that is further reduced by impedance mismatches between the current system and what the FPGA needs in order to provide maximum performance. Add to that the fact that the instances in the cloud do not provide enough computing capacity for the software part of the engine to put enough load on the FPGA, and the result is a high cost for FPGA deployments. The obvious solution to this is to have access to instances with a different configurations. On the research front, these findings point out at the importance of considering how to put enough load on the FPGA and the overhead of getting results back.

As pointed out above, at the moment, a better way to use FPGAs seems to be as first-class processors directly accessible from the network. This could look like the configuration used in Microsoft Azure [9], where the FPGA processes packages at line-rate directly from the network. Alternatively, one could think of having stand-alone FPGAs or FPGA clusters also directly connected to the network, as IBM has proposed on their CloudFPGA [27, 31] or as in Amazon's AQUA. As future work, we plan to explore the impact of different communication interfaces on the overall system, such as using TCP/IP communication [28], as well as RDMA [29].

6 CONCLUSION

Data centres make computing heterogeneity affordable. While research has been showing great potential for the use of hardware accelerators, integrating them into complex and legacy systems is yet a challenge. The acceleration nature of such devices requires a high throughput input, but current interconnects limit this to what conventional CPUs are able to submit. In this paper, we provide an extensive study of such imbalance using a commercial flight search engine as a use case. We took a research prototype of an FPGA-based accelerator and made it a Proof-of-Concept integrated within a realistic search engine under realistic constraints. We show that, while orders of magnitude better performance may be achieved in a stand-alone setup, mismatches between the upstream system and the accelerator significantly reduce the benefits. Additionally, the FPGA offerings currently available in the cloud are still too constrained, preventing a full utilisation of the FPGA computing power due to a mismatch with the CPU power and interconnect bandwidth. We shed light on the current integration and deployment issues of such solutions, and show how scalability, resource optimisation, and cost efficiency remain a challenge if accelerators are subordinated to a conventional host CPU. We hope that these insights will encourage further research on the end-to-end aspects of accelerator deployments and also inform existing deployments to expand the range of options available.

ACKNOWLEDGMENTS

We would like to thank Anthony Hock-Koon, Nicols Bondoux, Teddy Roy, Mourad Boudia, and Matteo Casalino for their contributions, which took place as part of a research project funded by Amadeus. This paper extends and updates results obtained as part of the project [19]. The development of the proof-of-concept was supported by AMD through the HACC cluster (amd-haccs.io) deployed at ETH Zurich (systems.ethz.ch/research/data-processing-on-modern-hardware/hacc.html).

REFERENCES

[1] Jeff Barr. 2021. *AQUA (Advanced Query Accelerator) – A Speed Boost for Your Amazon Redshift Queries.* Amazon Web Services. Retrieved May 12, 2023 from https://aws.amazon.com/blogs/aws/new-aqua-advanced-query-accelerator-for-amazon-redshift

[2] Luiz André Barroso, Mike Marty, David A. Patterson, and Parthasarathy Ranganathan. 2017. Attack of the killer microseconds. *Commun. ACM* 60, 4 (2017), 48–54. https://doi.org/10.1145/3015146

[3] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. 2016. When Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration. In *24th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2016, Washington, DC, USA, May 1-3, 2016.* IEEE Computer Society, 29. https://doi.org/10.1109/FCCM.2016.18

[4] Monica Chiosa, Fabio Maschi, Ingo Müller, Gustavo Alonso, and Norman May. 2022. Hardware Acceleration of Compression and Encryption in SAP HANA. *Proc. VLDB Endow.* 15, 12 (2022), 3277–3291. https://www.vldb.org/pvldb/vol15/p3277-chiosa.pdf

[5] George Coulouris, Jean Dollimore, and Tim Kindberg. 2002. *Distributed Systems: Concepts and Designs.* Addison-Wesley-Longman.

[6] CXL. 2020. *Compute Express Link: The Breakthrough CPU-to-Device Interconnect.* Compute Express Link. Retrieved May 12, 2023 from https://www.computeexpresslink.org/about-cxl

[7] Anne C. Elster and Tor A. Haugdahl. 2022. Nvidia Hopper GPU and Grace CPU Highlights. *Comput. Sci. Eng.* 24, 2 (2022), 95–100. https://doi.org/10.1109/MCSE.2022.3163817

[8] Jonathon Evans. 2022. Nvidia Grace. In *2022 IEEE Hot Chips 34 Symposium, HCS 2022, Cupertino, CA, USA, August 21-23, 2022.* IEEE, 1–20. https://doi.org/10.1109/HCS55958.2022.9895599

[9] Daniel Firestone, Andrew Putnam, Sambrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian M. Caulfield, Eric S. Chung, Harish Kumar Chandrappa, Somesh Chaturmohta, Matt Humphrey, Jack Lavier, Norman Lam, Fengfen Liu, Kalin Ovtcharov, Jitu Padhye, Gautham Popuri, Shachar Raindel, Tejas Sapre, Mark Shaw, Gabriel Silva, Madhan Sivakumar, Nisheeth Srivastava, Anshuman Verma, Qasim Zuhair, Deepak Bansal, Doug Burger, Kushagra Vaid, David A. Maltz, and Albert G. Greenberg. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2018, Renton, WA, USA, April 9-11, 2018*, Sujata Banerjee and Srinivasan Seshan (Eds.). USENIX Association, 51–66. https://www.usenix.org/conference/nsdi18/presentation/firestone

[10] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct Access, High-Performance Memory Disaggregation with DirectCXL. In *2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, Jiri Schindler and Noa Zilberman (Eds.). USENIX Association, 287–294. https://www.usenix.org/conference/atc22/presentation/gouk

[11] Chris Gregg and Kim M. Hazelwood. 2011. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2011, 10-12 April, 2011, Austin, TX, USA.* IEEE Computer Society, 134–144. https://doi.org/10.1109/ISPASS.2011.5762730

[12] Gui Huang, Xuntao Cheng, Jianying Wang, Yujie Wang, Dengcheng He, Tieying Zhang, Feifei Li, Sheng Wang, Wei Cao, and Qiang Li. 2019. X-Engine: An Optimized Storage Engine for Large-scale E-commerce Transaction Processing. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). ACM, 651–665. https://doi.org/10.1145/3299869.3314041

[13] Zsolt István. 2020. *The NOPE Manifesto.* Retrieved May 12, 2023 from https://zistvan.github.io/doc/nope-manifesto.pdf

[14] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. 2020. Do OS abstractions make sense on FPGAs?. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020.* USENIX Association, 991–1010. https://www.usenix.org/conference/osdi20/presentation/roscoe

[15] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift (Eds.). ACM, 574–587. https://doi.org/10.1145/3575693.3578835

[16] Cheng Luo, Man-Kit Sit, Hongxiang Fan, Shuanglong Liu, Wayne Luk, and Ce Guo. 2019. Towards Efficient Deep Neural Network Training by FPGA-Based Batch-Level Parallelism. In *27th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2019, San Diego, CA, USA, April 28 - May 1, 2019.* IEEE, 45–52. https://doi.org/10.1109/FCCM.2019.00016

[17] Clemens Lutz, Sebastian Breß, Steffen Zeuch, Tilmann Rabl, and Volker Markl. 2020. Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1633–1649. https://doi.org/10.1145/3318464.3389705

[18] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit O. Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift (Eds.). ACM, 742–755. https://doi.org/10.1145/3582016.3582063

[19] Fabio Maschi, Gustavo Alonso, Anthony Hock-Koon, Nicolas Bondoux, Teddy Roy, Mourad Boudia, and Matteo Casalino. 2021. From Research to Proof-of-Concept: Analysis of a Deployment of FPGAs on a Commercial Search Engine. *CoRR* abs/2108.09073 (2021). arXiv:2108.09073 https://arxiv.org/abs/2108.09073

[20] Fabio Maschi, Dario Korolija, and Gustavo Alonso. 2023. Serverless FPGA: Work-In-Progress. In *Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies, SESAME 2023, Rome, Italy, 8 May 2023*, Dmitrii Ustiugov, Rodrigo Bruno, Pedro Fonseca, Boris Grot, and Antonio Barbalace (Eds.). ACM, 1–4. https://doi.org/10.1145/3592533.3592804

[21] Fabio Maschi, Muhsen Owaida, Gustavo Alonso, Matteo Casalino, and Anthony Hock-Koon. 2020. Making Search Engines Faster by Lowering the Cost of Querying Business Rules Through FPGAs. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 2255–2270. https://doi.org/10.1145/3318464.3386133

[22] Norman May, Daniel Ritter, Andre Dossinger, Christian Färber, and Suleyman Demirsoy. 2023. DASH: Asynchronous Hardware Data Processing Services. In *13th Conference on Innovative Data Systems Research, CIDR 2023*. www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p6-may.pdf

[23] Bertha Mazon-Olivo, Dixys L. Hernández-Rojas, José Maza-Salinas, and Alberto Pan. 2018. Rules engine and complex event processor in the context of internet of things for precision agriculture. *Comput. Electron. Agric.* 154 (2018), 347–360. https://doi.org/10.1016/j.compag.2018.09.013

[24] Timothy Prickett Morgan. 2021. *Finally, A Coherent Interconnect Strategy: CXL Absorbs Gen-Z.* The Next Platform. Retrieved May 12, 2023 from https://www.nextplatform.com/2021/11/23/finally-a-coherent-interconnect-strategy-cxl-absorbs-gen-z

[25] Muhsen Owaida, Gustavo Alonso, Laura Fogliarini, Anthony Hock-Koon, and Pierre-Etienne Melet. 2019. Lowering the Latency of Data Processing Pipelines Through FPGA based Hardware Acceleration. *Proc. VLDB Endow.* 13, 1 (2019), 71–85. https://doi.org/10.14778/3357377.3357383

[26] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James R. Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 13–24. https://doi.org/10.1109/ISCA.2014.6853195

[27] Burkhard Ringlein, François Abel, Alexander Ditter, Beat Weiss, Christoph Hagleitner, and Dietmar Fey. 2019. System Architecture for Network-Attached FPGAs in the Cloud using Partial Reconfiguration. In *29th International Conference on Field Programmable Logic and Applications, FPL 2019, Barcelona, Spain, September 8-12, 2019*, Ioannis Sourdis, Christos-Savvas Bouganis, Carlos Álvarez, Leonel Antonio Toledo Díaz, Pedro Valero-Lara, and Xavier Martorell (Eds.). IEEE, 293–300. https://doi.org/10.1109/FPL.2019.00054

[28] Mario Ruiz, David Sidler, Gustavo Sutter, Gustavo Alonso, and Sergio López-Buedo. 2019. Limago: An FPGA-Based Open-Source 100 GbE TCP/IP Stack. In *29th International Conference on Field Programmable Logic and Applications, FPL 2019, Barcelona, Spain, September 8-12, 2019*, Ioannis Sourdis, Christos-Savvas Bouganis, Carlos Álvarez, Leonel Antonio Toledo Díaz, Pedro Valero-Lara, and Xavier Martorell (Eds.). IEEE, 286–292. https://doi.org/10.1109/FPL.2019.00053

[29] David Sidler, Zeke Wang, Monica Chiosa, Amit Kulkarni, and Gustavo Alonso. 2020. StRoM: smart remote memory. In *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*, Angelos Bilas, Kostas Magoutis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer (Eds.). ACM, 29:1–29:16. https://doi.org/10.1145/3342195.3387519

[30] Sajjad Tamimi, Florian Stock, Andreas Koch, Arthur Bernhardt, and Ilia Petrov. 2022. An Evaluation of Using CCIX for Cache-Coherent Host-FPGA Interfacing. In *30th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2022, New York City, NY, USA, May 15-18, 2022*. IEEE, 1–9. https://doi.org/10.1109/FCCM53951.2022.9786103

[31] Jagath Weerasinghe, Raphael Polig, François Abel, and Christoph Hagleitner. 2016. Network-attached FPGAs for data center applications. In *2016 International Conference on Field-Programmable Technology, FPT 2016, Xi'an, China, December 7-9, 2016*, Yuchen Song, Shaojun Wang, Brent Nelson, Junbao Li, and Yu Peng (Eds.). IEEE, 36–43. https://doi.org/10.1109/FPT.2016.7929186

[32] Yuan Yuan, Rubao Lee, and Xiaodong Zhang. 2013. The Yin and Yang of Processing Data Warehousing Queries on GPU Devices. *Proc. VLDB Endow.* 6, 10 (2013), 817–828. https://doi.org/10.14778/2536206.2536210

[33] Mark Zhao, Niket Agarwal, Aarti Basant, Bugra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, Sundaram Narayanan, Jack Langman, Kevin Wilfong, Harsha Rastogi, Carole-Jean Wu, Christos Kozyrakis, and Parik Pol. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: industrial product. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*, Valentina Salapura, Mohamed Zahran, Fred Chong, and Lingjia Tang (Eds.). ACM, 1042–1057. https://doi.org/10.1145/3470496.3533044