Accelerating database systems using FPGAs: A survey

Philippos Papaphilippou, Wayne Luk
Department of Computing, Imperial College London, UK
{pp616, w.luk}@imperial.ac.uk

***Abstract*—Database systems are key to a variety of applications, and FPGA-based accelerators have shown promise in supporting high-performance database systems. This survey presents a systematic review of research relating to accelerating analytical database systems using FPGAs. The review includes studies of database acceleration frameworks and accelerator implementations for various database operators. Finally, the survey includes some promising future technologies and discussion on the challenges to be addressed by the future research in this area.**

***Index Terms*—FPGA, Databases, Acceleration, Operators, Computer architecture, Analytics**

I. Introduction

While the use of FPGAs in accelerating database operations has been explored for a long time [1], reports about their application to accelerating industrial-scale databases have only emerged recently. Baidu, for example, has developed SDA, a software-defined accelerator for general-purpose big data analysis systems such as SparkSQL and HIVE, based on Xilinx KU115 FPGAs running at 300MHz [2]. It supports abstract SQL operations. For a real case query based on the TPC-DS query3 benchmark, the FPGA runs 55 times faster than a 12-core server.

FPGAs are an attractive option for accelerating different database systems. FPGAs provide big amounts of flexibility in a relatively cheap package that can be exploited by the programmer or hardware designer to use its resources for building an accelerator. Since the functionality of a database can be specific to the user's needs, developing a general purpose database processor, such as with static ASIC designs, would be expensive. For this reason, FPGAs have found their way into the data centers to become co-processors for specialized applications.

Database systems can be demanding, as their size can easily reach 'Big Data' dimensions, with tables acquiring terabytes of memory. Traditionally, the database systems were mostly On-line Transaction Processing (OLTP) systems. These systems are characterised by simplicity in terms of the transactions. With the rise in popularity of data mining and machine learning in big data, another database category emerged, called On-line Analytical Processing (OLAP) which has more complex queries that need to access and summarise multi-dimensional data. In the latter category, FPGAs have been perceived as an attractive solution as in many operations the data is accessed in a streaming fashion and the queries are sometimes pipelineable.

FPGA development has its own challenges. Different architectures, including CPUs and GPUs, have different inefficiencies and introduce different bottlenecks to performance. In order for an FPGA accelerator to be effective, the problem needs to have certain qualities, such as to be naturally parallel. FPGAs may perform significantly worse than a CPU if the application is control-heavy, i.e. it has a lot of unpredictable branches. The challenges in design may also include the steep learning curve and the immaturity and proprietary nature of the current FPGA development toolchains.

In Section II, there are the challenges of the modern FPGA technology in respect to database operator acceleration. In Section III, we review some representative works on Frameworks and accelerator implementations for database acceleration. Finally, in Section IV, we discuss some alternative and future technologies to overcome the current bottlenecks in performance.

II. Challenges

A. Main memory throughput

In general, database accelerators require streaming big amounts of data and they are bottlenecked by the memory bandwidth [3]. The obtainable throughput can be lower, due to the communication protocol.

Related studies use the DMA bandwidth [4] or the aggregate throughput from other methods, such as with unified virtual addressing (UVA) memory for GPUs [5] or with multiple-FPGA solutions [3], utilizing one or more PCIe links instead, which is currently at around 6.5 GB/s per link. CPU has the highest memory bandwidth to main memory, with modern Intel processors reaching over 80 GB/s read and 38GB/s write speeds [6].

What is usually referred as memory bandwidth is the throughput of the co-processor's bigger memory (DRAM) [7], but the memory access throughput to the main memory is much lower. This is because most platforms are in a non-uniform memory access (NUMA) machine, which means the host processor has a main memory different to that of the FPGA.

In some systems such as Intel's HARP v1, the FPGA does not have its own big memory [4] and the data still needs to be obtained through the processor. Last, there are some SoCs (System on Chip) that combine CPU cores with an FPGA and they share the same speed to DRAM, but they can be low-end in this aspect and they are also often used in a NUMA machine with another CPU being the host.

For the moment, some algorithms allow workarounds. An approach is to use the DRAM of the FPGA more wisely, such as with pre-loading specified datasets or applying optimizations to minimise the bandwidth requirements. The on-chip or DRAM memories of the FPGA can also be used as scratchpads or caches, such as by storing lookup tables. The work of Halstead et al. [8] is an example of a database accelerator that is inspired by the cache hierarchy for multiprocessors. The database processor keeps entries in the on-chip memory of the FPGA and makes multiple requests simultaneously to 'hide' the occurring memory latencies.

In addition, when aggregated, the DRAM of a multi-FPGA server can reach or even exceed the amount of main memory of the CPU. Therefore, it would also make sense to use this memory for an in-memory database system.

B. Latency

In big data streaming situations, the FPGA-induced latency is generally negligible. The latency to access main memory becomes important in access patterns that are more unpredictable, such as with random accesses. A database accelerator whose performance is susceptible to memory latency is hash join [8], as the entries are references to main memory locations for the elements to be joined. For that reason, workarounds inspired from CPU's attempts to hide memory latency are used, such as prefetching and caching. The FPGA-induced latency is also important in accelerators whose performance is affected by the input, such as with some sorting implementations [9].

III. FPGAs and Databases

The works related to database accelerators can be divided into **frameworks** and **specialised accelerators**. The 'frameworks' class includes work that provides a software and hardware stack for accelerating user-defined database operations. In this class we have also included virtualisation frameworks for providing a VM-like functionality in data centers for users to deploy their own accelerators in the cloud. In the 'accelerator' class, we have included database processors that are specifically designed to support a set of common queries. This also includes relatively simplistic FPGA designs to accelerate single operations, but they could easily be used in a framework of the previous class.

A. Database acceleration frameworks

a) Database-specific frameworks: One characteristic work for providing a hardware-software solution is Centaur [4]. Centaur consists of two parts, the hardware part called the "FThreads manager" and the software part called the "Application interface". The software supports API calls in C++ through the UDF (user defined function) functionality of MonetDB [10]. The hardware component is responsible for managing the different user-defined hardware in terms of resource utilisation, memory accesses, and also pipelining.

Centaur implements a dynamic pipelining functionality. The FThreads manager connects consecutive hardware operators with FIFO queues for fast intermediate communication to enable pipelining in hardware. This is abstracted in the software, as the user is able to pipeline the operators in source code. The user can define it programmatically by using the FPipe object to pipe the output from one operator to the other. The UDFs also allow pipelining software and hardware components. This is useful because the CPU is shown to perform better on some operators and the FPGA resources might be too limited for all the required operators to co-exist. The addition of more heavily-optimised accelerators (and UDFs in this case) is left to the user.

DoppioDB [11] extends Centaur. DoppioDB uses MonetDB along with a number of optimized hardware designs for the FGPA, exploiting the UDF functionality for making calls to the hardware accelerator. DoppioDB implements regular expression, skyline and stochastic gradient descent on the FPGA and the rest of the workload is left for the CPU to process. This resulted in a system that can execute SQL queries for an analytical relational database with over 3 times speedup when compared to the baseline, which was to execute the queries on a high-performance CPU without acceleration.

The selection of MonetDB was a good choice of a database to accelerate in these works, as it supports giving access to whole tables, as well as adding the UDFs in plain C++. Giving access to whole tables is an advantage over the more traditional row-by-row processing, because it allows the table to be accessed in a streaming fashion by the FPGA. MonetDB is a column-store database, as it uses Binary Association Tables (BATs) as a data representation [10], to be more focused on high performance with big data. This is due to the natural ability to ignore fields who belong in infrequently accessed columns, having therefore more efficient memory access patterns. Other advantages of MonetDB for future research is that is open-source and it also approaches an in-memory database, which is an increasingly popular storage model and is here to stay [12].

b) General-purpose frameworks: There is also a selection of frameworks that are not directly related to databases, but could be a useful component for a complete system for accelerating database operations on FPGAs.

One example framework that targets a wider range of applications is an attempt to provide virtualisation services for FPGAs in data centers [13]. The main contribution of this paper is the extension of the OpenStack framework with tools to enable FPGA resource assignment to clients without making compromises on performance and ease of use. The solution is based on partial reconfiguration, as the main component of the FPGA design is a static logic to manage the client-programmable VFRs (Virtualized FPGA resources). The static logic mainly consists of input and output arbiters that ensure fast and secure communication and eliminate unpredictable states. There are contributions with both hardware designs and software, as a complete compile flow and boot sequence is provided, with all the required parts, such as a netlist file for the Xilinx tools to connect client designs in Verilog and produce valid bitstream files. The added latency for arbitration between different FPGA designs is negligible.

High-level synthesis tools could also be seen as frameworks for database acceleration, as they provide means for easier software-hardware interaction. One relatively popular general purpose framework is the MaxCompiler. Simple accelerators like some of the 'arithmetic' type below are just a few lines in MaxCompiler. Maxeler systems provide a high-level synthesis tool for FPGA accelerators along with their own dedicated high-performance servers. Database accelerators have already been deployed on a Maxeler system [3]. The server configurations have multiple FPGAs connected through the multiple sets of PCI Express lanes, which can lead to higher aggregate throughput and enable pipelining of different tasks across different FPGA boards. Some disadvantages of HLS are that they may provide less flexibility to the programmer, some designs may not be able to be represented efficiently with the provided functionality and there might be less opportunities for manual optimisation and root cause analysis.

B. Operator accelerators

In this subsection, we review a series of accelerators specific to databases or applicable to databases. Most of these are inspired from SQL queries, since it is an almost universal language for expressing database operations, more extensively than relational algebra, a more theoretical representation. Database operator accelerators can be divided into one of the following categories: **sort**, **select**, **join**, **string matching**, **filtering** and **arithmetic**. In Table I, there is an overview of the accelerated operators mentioned in this section. The access pattern attribute describes the patterns with which the data is moving and the data placement describes the original location of the data at the time of request.

a) ***Sort:*** A sorting module can be very useful in a database processor as many queries perform aggregations or table merging and according to their implementation and requirements it can be their costliest operation [3]. It can be used for sort-merge joins, which is a join algorithm easily applicable to FPGAs [3]. This is one of the cases where different algorithms perform better on different platforms [14]. Quicksort and variations are currently preferred on CPUs. Optimisations of quicksort can be found in the Oracle database implementation [18] and this research area is still fairly active. Other uses of sorting on databases include the analytics-related queries, hence the tau notation in relational algebra which represents sorting.

The current state-of-the-art sorting algorithms on FPGAs are not based on quicksort. In 2005, J. Harkins et al. [14] evaluated the performance of five different sorting algorithms on an FPGA: Quick sort, Heap sort, Radix sort, Bitonic sort, and Odd-even mergesort. The conclusion was that Radix sort was the winner but the it was only twice as fast as quicksort on the CPU, due to the technology limitations of that time, such as the lower memory throughput, low FPGA clock and fewer compiler optimisations. Mueller et al. reached similar observations for sorting networks with 2012 technology [9]. They found that an FPGA consumes much less power than a processor for similar performance and also that more tight collaboration of an FPGA with a CPU can actually yield better performance. They studied the performance of sorting networks on FPGA (odd-even sort and bitonic sort) and evaluated heterogeneous mergesort, which is a realistic use-case that performs the last levels of the merge operation on the CPU for data that does not fit the FPGA memory.

In a more database-centric setting, Casper et al. [3] in 2014 have created a mergesort implementation on the FPGA that focuses on minimizing the memory bandwidth requirements. The inclusion of a sorting accelerator was crucial for implementing equi-join entirely in hardware. They showed that this implementation is memory-bound, meaning that with advances in memory system technology it has the potential to further exceed the performance of a CPU-only solution.

There are also works that focus more on memory and energy efficiency. Chen et al.[19] in 2015 showed that a bitonic sort implementation can be 2 times more memory efficient and 5 times more energy efficient when compared to a CPU implementation. They also showed ways of trading performance for more memory and energy efficiency by changing the throughput with which the input data are accessed from the FPGA.

Srivastava et al. [20] have further optimised bitonic sort for accelerating big data applications. They showed that the last merge operations of mergesort are a bottleneck for large-scale sorting and replaced them with a heavily-pipelined bitonic network for higher throughput. It was inspired by the two works mentioned above [3], [19]. The result was a considerable speedup over the then current mergesort implementations.

b) ***Join:*** The join operation is a frequent operation of relational databases. The main functionality of join is to combine the columns of two tables by using different criteria. One common join operation is equi-join, which takes as input two tables and performs an equality comparison for merging rows with common values in their equivalent columns. There are multiple join algorithms, such as nested-loop join, sort-merge join and hash join, all of which have FPGA implementations.

Casper et al. [3] created a full equi-join implementation in hardware by pipelining different modules in a multi-FPGA solution, as found in a high-performance Maxeler system. Their approach was to do the sort-merge completely in hardware for minimizing the bandwidth bottleneck and therefore including a merge sort module. The main idea behind a regular sort-merge join is that the two tables that need to be merged, they must first be sorted by the respective key/column on which the comparison will be performed.

Hash joins have also been successfully implemented on FPGAs. The main consideration with hash joins is that hash operations may perform suboptimally when the hashed locations are not provided efficiently in a memory hierarchy. For example, when using a CPU, the multiple-level cache system will help hide the memory latencies from accessing the data if the working set is sufficiently small. Halstead et

TABLE I: Comparison of different accelerators for database operations in terms of data movement

Accelerator type	Algorithm	Ref.	Access pattern	Data placement
Sort	Quicksort, Heapsort, Radix sort	[14]	Streaming controlled[b]	FPGA[a]
	Odd-even mergesort, Bitonic sort	[9], [14]		FPGA
	Merge sort (as an intermediate step)	[3]		FPGA
	Heterogeneous mergesort	[9]		main memory (NUMA)
Selection	Selection (as an intermediate step)	[3]	Streaming read/write	FPGA
Join	Sort-Merge Join	[3]	Streaming read/write	main memory (NUMA)
	Hash Join	[8]	Random Access	main memory (NUMA)
String Matching	Regular expressions	[11], [15]	Streaming read/write	main memory (NUMA)
Filtering	Logical expressions	[16]	Streaming read/write	FPGA
	Skyline	[4], [11]	Streaming iterative	main memory (NUMA)
Arithmetic	Mullad (`a[i] = b[i]*C+D`)	[4]	Streaming read/write	main memory (NUMA)
	Percentage (`sum(b[i:j])/sum(b[:])`)	[4]	Streaming read, single value write	main memory (NUMA)
	Testcount (`if cond(a[i]): count++`)	[4]	Streaming read, single value write	main memory (NUMA)
	Stochastic Gradient Descent	[11], [17]	Streaming iterative	main memory (NUMA)

[a]When 'FPGA', the data reside on FPGA's DRAM or BRAM, ignoring initial copy operations, which is less realistic for a NUMA system
[b]The data is mostly accessed sequentially, often in iterations, but it is not clearly streaming. It is either fetched from multiple DRAM banks/ FIFO queues in a MIMD or SIMD fashion from independent sorting modules, or sent through commands upon algorithmic request.

al. [8] showed that by mimicking multi-tasking as found in multi-threaded implementations for multiprocessors, there is a smarter memory hierarchy utilisation resulting is increased performance. They have implemented a hardware-aware join operation to eliminate the effects of the absence of cache in the FPGA.

Recently, Roozmeh et al. [7] demonstrated how a heterogeneous system that combines an FPGA, a CPU and a GPU can accelerate database join operations. Their approach was to compare the performance of nested-loop join and sort-merge join for FPGA and GPU and they showed that a high-end FPGA can outperform the GPU test platforms.

*c) **String matching**:* Another focus of database acceleration research work is string matching operations. One of Centaur's[4] and also DoppioDB's [11] hardware user-defined functions (HUDFs) that were provided for evaluating the performance of the frameworks was a regular expression engine.

Regular expressions can be seen as short scripts which are well-defined for complex string matching capabilities. With the rise of machine learning and its integration in relation algebras for analytic operations, the non-strictly formatted data, such as string, can become the bottleneck in analytical databases [11]. The claimed speedup is up to 3 times in comparison with a high-performance CPU implementation. The accelerator implementation was described in more detail in [15]. This work focuses in database applications and provides more flexibility and resource utilisation by breaking down the regular expressions into parameters. This is achieved by having a flexible state graph that changes according to the automatically–generated parameterisation configuration vector.

Accelerating regular expressions in FPGAs is not only found in database analytics. The work of Sidhu et al. [21] dates back to 2001 and demonstrated non-deterministic finite automaton implementations on an FPGA working with O(n) speed, where n is the number of characters in a string.

*d) **Filtering**:* The most common way of filtering is by using logical expressions, as found in SQL's WHERE clause, which is a number of conditions for each row. One accelerator designed to do this in hardware is a database processor from 2016 [16], designed for fast database operations and is implemented on an FPGA. This work takes advantage of a highly-coupled SoC architecture that provides high-throughput communication between two arm cores and the FPGA, residing on the same die (Altera Arria V SoC FPGA). In contrast with other accelerators, it focuses on small and embedded devices. The main idea was to use a Content Addressable Memory (CAM), with which a series of queries will be executed in parallel but inside the chip. The queries are encoded in a format that reminds an Instruction Set Architecture (ISA) found in processors.

One more complex way of filtering is by using the Skyline operator [22]. Skyline is a family of queries that filter multi-dimensional data using multiple criteria. This requires streaming the same data multiple times and it can be expensive for CPUs. Centaur [4] and DoppioDB [11] explore an FPGA implementation of a skyline query engine and demonstrate a considerable performance improvement over CPUs.

*e) **Arithmetic**:* Other accelerators that would help increase modern database systems are a set of operators that are related to big data analytics and machine learning/statistics. The list could be very extensive, due to the many different use-cases and therefore we present only some that were seen in a database acceleration context.

Simpler examples such as mullad (scale one column by a constant and add another constant to each element) and percentage (the percentage of the sum of a subsequence of a column over the sum of all elements) and testcount (count all rows satisfying a common condition) are included in the exploration of the Centaur framework [4]. CPUs can also be efficient with such operations, if vectorisation is exploited by the compiler and the architecture supports them.

A more complex accelerator is a Stochastic Gradient Descent (SGD) implementation [17]. SGD is used for building linear models in machine learning and is an iterative operation, until a set of stopping conditions are met. Through quantization, the operation can be optimised for different accuracy needs. It is shown to outperform a high-end CPU implementation by an order of magnitude for the same level of accuracy, despite the memory bandwidth limitation.

IV. Alternative and future architectures

A. Other co-processors

Graphics Processing Units (GPUs) are currently a competitor to FPGAs for database acceleration. However, they also suffer from the main bottleneck as FPGAs when doing memory accesses, which is the limited memory bandwidth. There is an exception to this and it is a new interconnection protocol for NVidia GPUs called NVlink [23] and can yield a main memory access speed of 25 GB/s which is close to that of the CPU.

Roozmeh et al. [7] compared bleeding edge FPGA technology with GPUs and found that FPGAs can actually be faster than GPUs for some database operators. Both the FPGA platforms were 40% more energy efficient than the GPUs. However, this work highlights that the high-end FPGA test platform (Virtex UltraScale VU440) that was the best performance-wise in comparison with the GPUs, costed $37000, which was two orders of magnitude more expensive.

Many-core processors, such as Intel's Xeon Phi, is another competitor to FPGAs. As a co-processor, Xeon Phi suffers from similar disadvantages for databases. Cheng et al. [6] have optimised state-of-the-art hash join algorithms for better utilisation of the resources of the latest Xeon Phis. The local high-bandwidth Multi-Channel DRAM (MCDRAM) is found better to work as a big cache for the main memory and combined with software prefetching and NUMA-aware partitioning, Xeon Phis can give an important speedup for hash joins. Some of the optimisations can be found in a smaller scale on the FPGA competitor for hash joins by Halstead et al. [8].

Additionally, some innovative many-core architectures, apart from leaving more control to the programmer for simpler hardware logic and flexibility [24], they can support direct inter-core communication which can be used for pipelining operations [25]. This is something very desirable in database queries [4] and it is not supported in mainstream CPUs.

B. Faster interconnection protocols

Moussalli [26] discussed the future directions for accelerating databases on GPUs and FPGAs in 2017 and suggested that various new technologies could eliminate the memory bandwidth limitation of accelerators. Among NVlink [23], which is for GPUs, the IBM POWER8 processor, with its Coherent Accelerator Processor Interface (CAPI) was also mentioned for faster database acceleration.

CAPI promises higher memory bandwidth from the accelerator (including FPGAs) by eliminating software abstractions for better bandwidth utilisation. This is currently implemented on top of PCIe Gen. 3 and has a memory bandwidth of 16GB/s bidirectional [27] (16 lanes of 8 Gbps each, for one device), which is low compared to CPU's speeds. However, future processors of the IBM POWER9 architecture will have up to 48 PCIe lanes and OpenCAPI, the successor to CAPI, which can yield performance of 150GB/s aggregate throughput (48 lanes of 25Gbps each) [28]. Such protocols are expected to improve the FPGA performance for database operations, including multi-way joins [29].

The latest advancements in FPGAs revolve around System-On-Chip (SoC) packages, such as with the MPSoC platform [30] that combines an FPGA with ARM and GPU cores. This could be a pivotal point for heterogeneous workloads, but in regards the bandwidth performance it is relatively low-end [16]. An interesting platform is the Intel HARP v1 prototype [4]. It allows an FPGA to co-exist in a second socket in a multi-socket server. While this might not be an important change over FPGAs connected with PCIe in high-performance servers, it could be considered as an early step for the introduction of higher-performing Xeon+FPGA chips [31] and hopefully with the highest main memory bandwidth for the FPGA.

C. Near-data processing/ processing in memory

In the datacenter scale, near-data processing (NDP) is promising to overcome the power and performance implications of processing huge amounts of data. By moving computation to memory, NDP avoids chip-to-chip transfers and yields efficient in-memory computation. Due to the recent advances in 3D-stacking technology, NDP is predicted to gain more attention as a solution [32].

More related to FPGAs and databases [26], ConTutto is a novel platform to bring FPGAs close to the memory by incorporating them with DRAM in the form of DIMM cartridges [33]. There are working prototypes of such DIMMs and plug into an IBM POWER8-based server. Acting as a memory buffer, the FPGA can have a memory access bandwidth of 11 GB/s per DIMM port, or an aggregate of 35 GB/s for a single memory channel.

V. Conclusions

FPGAs are an interesting platform for hardware acceleration of a considerable amount of database operations. It is clear that FPGAs are superior for energy efficiency, but their advantage in overall performance for non-local data is sometimes questionable.

The main bottleneck is the limited main memory bandwidth found in current hardware, which is around 6 GB/s and is low in comparison with that of CPUs, reaching over 35 GB/s. There are clever algorithms or modifications that make better usage of the available memory throughput, such as by caching entries, compression, improving the inter-chip communication protocol and data localities. One possible workaround could be to develop in-memory databases for FPGAs.

This limitation is likely to be solved in future architectures, such as with IBM's POWER9 servers that will allow up to 150 GB/s main memory throughput over 48 PCIe lanes using the OpenCAPI protocol, Near-data processing and high-end SoC platforms, such as Intel's Xeon+FPGA, that may provide main memory bandwidth close to that of the CPU.

Acknowledgments

This research was supported by dunnhumby. The authors would like to thank Chris Brooks and Rosie Prior from dunnhumby for their valuable involvement in the partnership program. The support of the United Kingdom EPSRC (grant number EP/I012036/1, EP/L00058X/1, EP/L016796/1, EP/N031768/1 and EP/K034448/1), European Union Horizon 2020 Research and Innovation Programme (grant number 671653) is gratefully acknowledged.

References

[1] N. Shirazi, D. Benyamin, W. Luk, P. Y. Cheung, and S. Guo, "Quantitative analysis of FPGA-based database searching," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 28, no. 1-2, pp. 85–96, 2001.

[2] J. Ouyang, W. Qi, W. Yong, Y. Tu, J. Wang, and B. Jia, "SDA: Software-defined accelerator for general-purpose big data analysis system," in *Hot Chips: A Symposium on High Performance chips, Hotchips*, 2016.

[3] J. Casper and K. Olukotun, "Hardware acceleration of database operations," in *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, ACM, 2014, pp. 151–160.

[4] M. Owaida, D. Sidler, K. Kara, and G. Alonso, "Centaur: A framework for hybrid CPU-FPGA databases," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*, IEEE, 2017, pp. 211–218.

[5] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk, "GPU join processing revisited," in *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, ACM, 2012, pp. 55–62.

[6] X. Cheng, B. He, X. Du, and C. T. Lau, "A study of main-memory hash joins on many-core processor: A case with intel knights landing architecture," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ACM, 2017, pp. 657–666.

[7] M. Roozmeh and L. Lavagno, "Implementation of a performance optimized database join operation on FPGA-GPU platforms using OpenCL," in *Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC), 2017 IEEE*, IEEE, 2017, pp. 1–6.

[8] R. J. Halstead, I. Absalyamov, W. A. Najjar, and V. J. Tsotras, "FPGA-based Multithreading for In-Memory Hash Joins," in *CIDR*, 2015.

[9] R. Mueller, J. Teubner, and G. Alonso, "Sorting networks on FPGAs," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 1–23, 2012.

[10] S. I.F.G. N. Nes and S. M.S.M. M. Kersten, "MonetDB: Two decades of research in column-oriented database architectures," *Data Engineering*, vol. 40, 2012.

[11] D. Sidler, Z. István, M. Owaida, K. Kara, and G. Alonso, "doppioDB: A hardware accelerated database," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 2017, pp. 1659–1662.

[12] T. Lahiri, S. Chavan, M. Colgan, D. Das, A. Ganesh, M. Gleeson, S. Hase, A. Holloway, J. Kamp, T.-H. Lee, *et al.*, "Oracle database in-memory: A dual format in-memory database," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, IEEE, 2015, pp. 1253–1258.

[13] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "Fpgas in the cloud: Booting virtualized hardware accelerators with openstack," in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*, IEEE, 2014, pp. 109–116.

[14] J. Harkins, T. El-Ghazawi, E. El-Araby, and M. Huang, "Performance of sorting algorithms on the SRC 6 reconfigurable computer," in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, IEEE, 2005, pp. 295–296.

[15] D. Sidler, Z. István, M. Owaida, and G. Alonso, "Accelerating pattern matching queries in hybrid CPU-FPGA architectures," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ACM, 2017, pp. 403–415.

[16] X.-T. Nguyen, H.-T. Nguyen, T.-T. Hoang, K. Inoue, O. Shimojo, T. Murayama, K. Tominaga, and C.-K. Pham, "An efficient FPGA-based database processor for fast database analytics," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*, IEEE, 2016, pp. 1758–1761.

[17] K. Kara, D. Alistarh, G. Alonso, O. Mutlu, and C. Zhang, "FPGA-accelerated Dense Linear Machine Learning: A Precision-Convergence Trade-off," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*, IEEE, 2017, pp. 160–167.

[18] R. Neininger and J. Straub, "Probabilistic analysis of the dual-pivot quicksort counti," in *2018 Proceedings of the Fifteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, SIAM, 2018, pp. 1–7.

[19] R. Chen, S. Siriyal, and V. Prasanna, "Energy and memory efficient mapping of bitonic sorting on FPGA," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, 2015, pp. 240–249.

[20] A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, "A hybrid design for high performance large-scale sorting on FPGA," in *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*, IEEE, 2015, pp. 1–6.

[21] R. Sidhu and V. K. Prasanna, "Fast regular expression matching using FPGAs," in *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*, IEEE, 2001, pp. 227–238.

[22] L. Woods, G. Alonso, and J. Teubner, "Parallel computation of skyline queries," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, IEEE, 2013, pp. 1–8.

[23] D. Foley and J. Danskin, "Ultra-performance Pascal GPU and NVLink interconnect," *IEEE Micro*, vol. 37, no. 2, pp. 7–17, 2017.

[24] D. Bates, A. Chadwick, and R. Mullins, "Configurable memory systems for embedded many-core processors," *ArXiv preprint arXiv:1601.00894*, 2016.

[25] D. Bates, A. Bradbury, A. Koltes, and R. Mullins, "Exploiting tightly-coupled cores," *Journal of Signal Processing Systems*, vol. 80, no. 1, pp. 103–120, 2015.

[26] R. Moussalli, "Tradeoffs and considerations in the design of accelerators for database applications," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, IEEE, 2017, pp. 1615–1615.

[27] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel, "CAPI: A coherent accelerator processor interface," *IBM Journal of Research and Development*, vol. 59, no. 1, pp. 7–1, 2015.

[28] T. P. Morgan, *Opening Up The Server Bus For Coherent Acceleration*, https://www.nextplatform.com/2016/10/17/opening-server-bus-coherent-acceleration/, [Online; accessed 30-March-2018], 2016.

[29] K. Huang, "Multi-way Hash Join Based on FPGAs," 2018.

[30] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 13, no. 3, p. 40, 2008.

[31] N. Hemsoth, *Intel Marrying FPGA, Beefy Broadwell for Open Compute Future*, https://www.nextplatform.com/2016/03/14/intel-marrying-fpga-beefy-broadwell-open-compute-future/, [Online; accessed 12-March-2018], 2016.

[32] R. Balasubramonian and B. Grot, "Near-data processing," *IEEE Micro*, vol. 36, no. 1, pp. 4–5, 2016.

[33] B. Sukhwani, T. Roewer, C. L. Haymes, K.-H. Kim, A. J. McPadden, D. M. Dreps, D. Sanner, J. Van Lunteren, and S. Asaad, "Contutto: a novel FPGA-based prototyping platform enabling innovation in the memory subsystem of a server class processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2017, pp. 15–26.