Memory-Centric Reconfigurable Accelerator for Classification and Machine Learning Applications

ROBERT KARAM, University of Florida
SOMNATH PAUL, Intel Labs
RUCHIR PURI, IBM T. J. Watson Research Lab
SWARUP BHUNIA, University of Florida

Big Data refers to the growing challenge of turning massive, often unstructured datasets into meaningful, organized, and actionable data. As datasets grow from petabytes to exabytes and beyond, it becomes increasingly difficult to run advanced analytics, especially Machine Learning (ML) applications, in a reasonable time and on a practical power budget using traditional architectures. Previous work has focused on accelerating analytics readily implemented as SQL queries on data-parallel platforms, generally using off-the-shelf CPUs and General Purpose Graphics Processing Units (GPGPUs) for computation or acceleration. However, these systems are general-purpose and still require a vast amount of data transfer between the storage devices and computing elements, thus limiting the system efficiency. As an alternative, this article presents a reconfigurable memory-centric advanced analytics accelerator that operates at the last level of memory and dramatically reduces energy required for data transfer. We functionally validate the framework using an FPGA-based hardware emulation platform and three representative applications: Naïve Bayesian Classification, Convolutional Neural Networks, and k-Means Clustering. Results are compared with implementations on a modern CPU and workstation GPGPU. Finally, the use of in-memory dataset decompression to further reduce data transfer volume is investigated. With these techniques, the system achieves an average energy efficiency improvement of 74× and 212× over GPU and single-threaded CPU, respectively, while dataset compression is shown to improve overall efficiency by an additional 1.8× on average.

CCS Concepts: • **Computing methodologies** → **Evolvable hardware**; Neural networks; • **Computer systems organization** → **Parallel architectures**; High-level language architectures; • **Hardware** → **Hardware accelerators**; **Programmable logic elements**; *Simulation and emulation*; *Memory and dense storage*; • **Information systems** → Clustering and classification; • **Applied computing** → Business intelligence;

Additional Key Words and Phrases: Reconfigurable architectures, hardware accelerators, machine learning, memory-centric, parallel processing, energy-efficiency

ACM Reference Format:
Robert Karam, Somnath Paul, Ruchir Puri, and Swarup Bhunia. 2017. Memory-centric reconfigurable accelerator for classification and machine learning applications. J. Emerg. Technol. Comput. Syst. 13, 3, Article 34 (May 2017), 24 pages.
DOI: http://dx.doi.org/10.1145/2997649

1. INTRODUCTION

Big data has become a ubiquitous part of everyday life [Sun and Heller 2012; Garber 2012; Gray 2013]. For companies that amass large datasets—whether from sensor data

Author's addresses: R. Karam and S. Bhunia, Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL.; S. Paul, Intel Labs, Hillsboro, OR.; R. Puri, IBM T.J. Watson Research Lab, Yorktown Heights, NY.

DOI: http://dx.doi.org/10.1145/2997649

or software logs, social networks, scientific experiments, network monitoring applications, or other sources—the problem is not acquiring the data, but rather analyzing it to identify patterns or provide insight for operational decisions [LaValle et al. 2011]. Often, these decisions are time-sensitive, such as identifying network intrusions in real time or reacting to fluctuations in stock indicators. Meeting such low-latency requirements on a limited power budget will become increasingly difficult as data streams increase in size and velocity [Sun and Heller 2012], especially when leveraging Machine Learning (ML) techniques for data processing.

To address this issue, many have turned to hardware acceleration, relying primarily on either the parallel-processing capabilities of General Purpose Graphics Processing Unit (GPGPU) platforms like NVIDIA CUDA [Nickolls et al. 2008; Chen et al. 2012; Araya-Polo et al. 2011] or Field Programmable Gate Array (FPGA) implementations of certain time-intensive kernels [Helmreich and Cowie 2006; Shan et al. 2010; Papadonikolakis and Bouganis 2012]. Such solutions are promising for simple analytics because they generally accelerate relational database queries on platforms implementing the MapReduce framework [Dean and Ghemawat 2008], exploiting the straightforward data parallelism in such applications [Gray 2013]. However, there are two main issues with these techniques:

(1) The problem of accelerating more complex analytics, where the parallelism may not be as readily apparent and the communication requirements vary significantly between applications, remains to be addressed in the context of increasingly stringent power and latency budgets. In fact, it has been shown that conventional multicore (both CPU and GPU) scaling is unlikely to meet the performance demands in future technology nodes, even with a practically unlimited power budget and that additional cores yield little increased performance due to limited application parallelism [Esmaeilzadeh et al. 2013].
(2) While these accelerators generally connect to the host via high-speed interconnects (e.g., PCIe), they are still at the mercy of the data transfer energy bottleneck. Regardless of the speed and/or efficiency of the accelerator, data transfer requirements vastly reduce their efficacy when processing massive datasets [Bergman et al. 2008].

These issues motivate us to find an alternative, ultralow power domain-specific architecture, focusing on energy-efficient execution of ML application, especially in the context of big data analytics. Such an architecture must be scalable, to deal with growing datasets, and reconfigurable, to readily implement a broad range of kernels as needed by the various applications. Furthermore, domain-specificity will improve area efficiency while supporting diverse applications in the target domain [Karam et al. 2016].

In this work, we present a framework that addresses these issues using a memory-centric spatiotemporal reconfigurable accelerator [Paul et al. 2014a, 2014b] customized for analytics applications, which operates at the last level of memory. As a memory-centric accelerator, it leverages the high-speed, high-density cell integration found in current memory devices to efficiently map entire analytics kernels to the accelerator. Furthermore, as a memory-centric framework, advances in promising emerging CMOS-compatible memory technologies can be leveraged to improve performance while reducing power consumption [Paul et al. 2009; Karam et al. 2015a, 2015b].

To validate this approach, three classes of analytics applications are analyzed for their typical operations and communication patterns, then mapped to the framework: classification (Naïve Bayes Classifier [NBC]), artificial neural networks (Convolutional Neural Network [CNN]), and clustering (K-Means Clustering [KMC]). Data is streamed to the framework using a back-buffering technique in both uncompressed and

compressed formats, employing a Huffman-based entropy coding scheme to effectively increase data bandwidth and system storage capabilities while significantly reducing transfer energy. A Verilog model of the framework is synthesized, and the power, performance, and area are extracted, and the energy efficiency is calculated. Finally, these values are compared with CPU and GPGPU implementations using the same datasets.

In summary, this work offers the following novel contributions:

(1) It presents the processing element microarchitecture and hierarchical interconnect for a domain-specific, memory-centric reconfigurable accelerator for ML kernels in the context of big data analytics. PEs support multi-input, multi-output lookup tables and are highly tailored to the application domain.
(2) It explores the processing and communication requirements for three classes of analytics applications and uses these observations in the design of the processing elements and interconnect.
(3) It considers the role of data compression for big data analytics in the context of disk-to-accelerator bandwidth and overall power savings and demonstrates the feasibility of in-memory decompression of datasets in the accelerator framework.
(4) It validates the approach using a multi-FPGA hardware emulation framework which provides functional and timing verification for the design. It studies the power, performance, and area results from synthesis and compares these with implementations on CPU and GPGPU. Overall, the accelerator demonstrates excellent energy efficiency when compared with CPU and GPU for the ML and classification domain of applications.

The rest of this article is organized as follows: Section 2 provides a brief overview on existing work in the related fields and summarizes the domain-specific requirements for the target applications. Section 3 describes the system organization, hardware architecture, the application mapping procedure, parallelism and execution models, and the data management techniques. Section 4 provides accelerator-specific implementation details for the benchmark applications, the dataset compression methodology, and the experimental setup. Sections 5 reports the experimental results, including raw throughput, energy efficiency, and the effects of dataset compression. Section 6 discusses the performance, parallelism, scalability, and application scope. Section 7 covers a broad range of related works for the hardware acceleration field, including FPGA, GPGPU, CGRA, and other many-core architectures, and contrasts these with the proposed accelerator; finally, Section 8 concludes, with future directions for the research.

2. BACKGROUND AND MOTIVATION

In this section, we explore the requirements of a big data analytics accelerator and discuss the state-of-the-art systems. We note the shortcomings of these systems, which motivates us to find a new solution.

2.1. Overview of Big Data Analytics

In general, analytics aims to identify meaningful patterns in data by employing a variety of statistical or ML methods. “Big Data” analytics presents a slew of problems beyond the basic algorithms and methods employed [Jacobs 2009], including storage and retrieval of data, and division and balancing of workloads, among others. These problems are multifaceted and complex, but are wholly contingent on the distributed computing paradigm (e.g., MapReduce) that is commonly used. A growing body of research aims to reduce processing time under these conditions using general-purpose CPUs, GPUs, or FPGAs for acceleration [Dean and Ghemawat 2008; Papadonikolakis and Bouganis 2012; Atasu et al. 2013; Bakkum and Skadron 2010; Bandre and

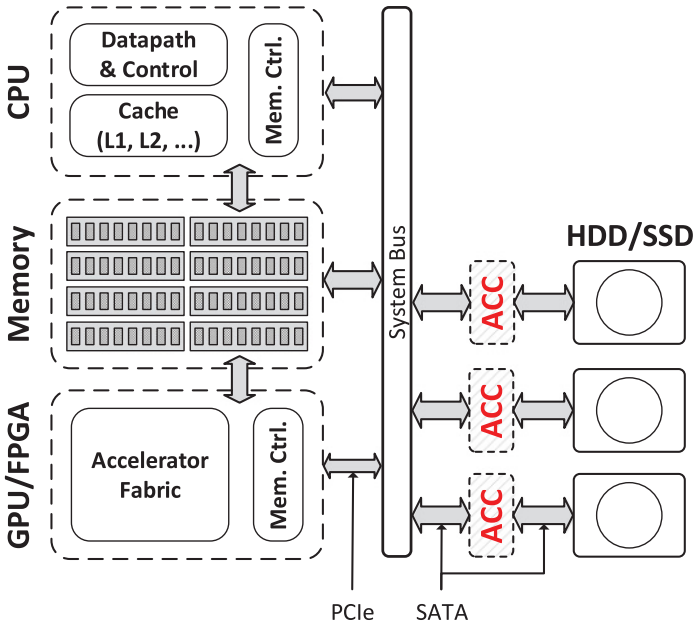

Fig. 1. Proposed reconfigurable accelerator for general analytics kernels. Moving the processing to the last level memory interface reduces data transfer power and results in greater overall efficiency when processing massive datasets.

Nandimath 2015; Chen et al. 2012; Farivar et al. 2008; Fu et al. 2014; Govindaraju et al. 2004; Halstead et al. 2013; Neshatpour et al. 2015; Putnam et al. 2014; Shan et al. 2010; Sukhwani et al. 2012; Wu et al. 2009; Zhao et al. 2009], but generally prioritize raw performance over energy efficiency due to the rapidly increasing data volume and generation velocity. We mainly compare with these platforms, rather than other CGRAs or many-core systems, because CPU, GPGPU, and FPGA tend to be more commonly used in larger scale and enterprise-level big data systems. A more in-depth discussion of these platforms is offered in Section 7.

Although these approaches tend to give excellent results once the data are transferred to the CPU/accelerator memory, the *data transfer* itself cannot be ignored. In fact, for most big data applications, data transfer represents the single largest component of the total processing power, potentially taking half the total processing time just in the transfer [Bergman et al. 2008]. Bringing the processing or acceleration closer to the data is therefore crucial for big data processing. Several examples of integrating processing in the main memory exist [Patterson et al. 1997; Murakami et al. 1997; Guo et al. 2014]. However, this brings up two issues: (i) compared to the last level of memory, main memory is limited in capacity, and (ii) data must still be brought to the main memory, which unnecessarily expends energy. By pushing the processing further down the hierarchy, a system is no longer limited by the main memory capacity, and data movement is greatly reduced. Therefore, an accelerator situated at the last level of memory, as shown in Figure 1, can take full advantage of the analytics domain-specific architecture.

2.2. Analytics Applications

Common analytics applications, including classification [Duda et al. 1973; Friedman et al. 1997] and clustering [MacQueen et al. 1967], are evaluated in order to identify

typical operations and communication requirements. Neural networks, which are capable of both classification and clustering, are also evaluated. Finally, the concept of in-accelerator dataset compression/decompression, which can potentially reduce overall power consumption, is also explored.

2.2.1. Operations. In general, the most common datapath operations were addition, multiplication, shift, and comparison, which were common to classification and clustering applications, as well as to neural network-based approaches. Furthermore, both the neural networks and the clustering algorithms require evaluation of analytic functions that are readily implemented in Lookup Tables (LUTs). In particular, the logistic (sigmoid) and hyperbolic tangent functions are popular for nonlinear activation stages in neural networks. For clustering, the distance computation (e.g., distance from the cluster mean/median/centroid/etc., or locating a nearest neighbor in hierarchical clustering) is critical because it is repeatedly applied to all points in the dataset. Two common distance metrics are Euclidian (LUT, square, and square root) and Manhattan distance (datapath, subtraction). In all instances studied, no more than four different non-datapath functions were required by the applications. In general, a very lightweight datapath, paired with sufficient lookup memory, enables the mapping of a wide range of analytics kernels.

2.2.2. Communication. To some extent, all applications exhibit a mix of Instruction Level Parallelism (ILP) and Data Level Parallelism (DLP) that can be exploited by an amenable multicore environment. Here, we refer to each core more generally as a Processing Element (PE), and we assume inter-PE communication is available on-demand for illustrative purposes.

—*Classification:* For NBC, training can be parallelized at the instruction level among several PEs as long as there is sufficient inter-PE bandwidth to communicate copies of the partial probabilities. Once each PE has a copy of the probabilities, independent classification (DLP) of each vector in a dataset is readily parallelized and requires no inter-PE communication.
—*Clustering:* For KMC, the PE communication requirement is higher than for NBC and could be significantly higher using another technique (e.g., agglomerative hierarchical clustering). From the algorithm definition, we know the number of clusters is predetermined, and we assume that the initial locations of the cluster centers are distributed along with a portion of the data among the PEs. Summary statistics—for example, the local cluster means and number of cluster members—must be communicated among the PEs so that each has a locally up-to-date copy of the current cluster means.
—*Neural Network:* The PE communication requirement is highest for the neural networks. In the convolutional neural network model evaluated, at least two mappings are possible, exploiting either ILP or DLP. For example, individual PEs can perform the two-dimensional convolution and nonlinear activation function independently for each feature map (DLP) given sufficient memory resources. Alternatively, a group of PEs can simultaneously evaluate the same network as long as there is sufficient inter-PE bandwidth for data transfer.

In general, the applications evaluated are highly parallel, with varying communication requirements that depend strongly on the particular mapping strategy. With smaller internal memories, individual kernel iterations must be mapped to multiple PEs, indicating a greater reliance on ILP, PE communication, and, therefore, higher bandwidth requirements. In contrast, larger internal memories allow individual kernel iterations to be mapped in individual PEs, reducing the overall inter-PE bandwidth requirements. For a fixed die area, this translates to fewer PEs. From the communication

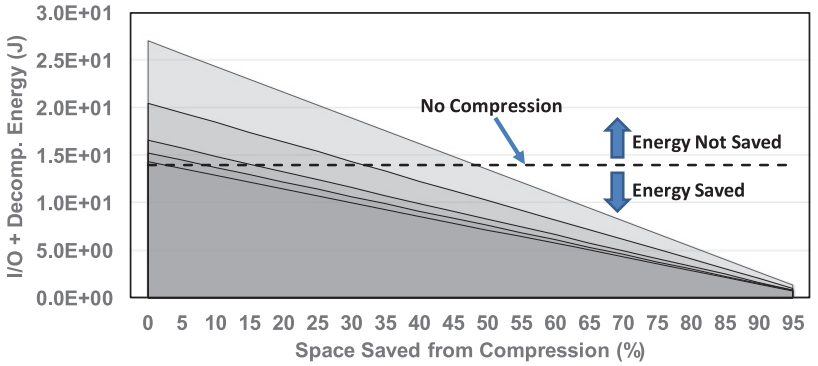

Fig. 2. Area plots represent different compression routines, using (from bottom to top) 15, 50, 100, 250, and 500 cycles on a given platform. Energy savings are achieved when the sum of transfer energy and decompression energy is less than the transfer energy for the uncompressed dataset. In this case, 1GB is transferred at 13 nJ/bit [Bergman et al. 2008].

analysis, the preferred model depends on the application mapping: Classifications are more likely to achieve higher throughput using a large number of small PEs, while KMC and CNN can use either model depending on the desired latency and throughput. Finally, we consider the data network: In the first case, for example, a classifier will only achieve its maximum throughput when sufficient data are available for processing. With a large number of PEs, this data distribution network increases in size and complexity. By having a smaller number of PEs with larger memories, the complexity of this network decreases. For a system to retain high performance and remain energy efficient, these factors must be balanced.

2.3. Disk-to-Accelerator Compression

Next, we consider the role of dataset filtering or compression in the context of big data analytics. Data filtering has been used previously at an enterprise level [Helmreich and Cowie 2006] to intelligently reduce the data volume by selectively transferring rows of structured data that met the query requirements. For unstructured data, it is unknown *a priori* where the salient data resides; this motivates us to instead use dataset compression to increase the effective disk-to-accelerator bandwidth. This is especially important in big data applications, where a reduction in data volume can result in lower transfer energy and latency if certain conditions are met, as discussed here.

Assuming that the data can be effectively compressed, decompression at the accelerator end must be lightweight and not negate the latency and energy saved from transferring less data. Determining if the application can benefit from in-accelerator dataset decompression therefore requires consideration of several variables, including the amount of data to be transferred, the associated transfer energy per bit, and the energy and latency associated with the decompression routine, as summarized by the inequality $P_{cx} + P_{dr} < P_{dx}$, where P_{cx} refers to the compressed dataset transfer power, P_{dr} is the decompression routine power, and P_{dx} is the full dataset transfer power. Furthermore, we assume the data are compressed once, but accessed and processed an indefinite number of times, so the initial data compression energy is amortized and not considered in the inequality.

Using representative values obtained from the synthesis results (Section 5), we can observe the relations among compression routine cycles, compression ratio, and estimated overall energy savings in Figure 2. Here, individual area plots represent

different compression routines, from 15 cycles (lowest) to 500 cycles (highest). Depending on the compression ratio, expressed as a percentage of space saved, the different routines achieve varying levels of total energy savings (I/O +Compute). We use this analysis when choosing the dataset compression for the accelerator.

Huffman coding [Huffman 1952] is chosen to compress the data prior to transfer. Decompression is implemented as a table lookup, and can be completed in a single cycle, whereas data setup and processing contribute an extra 13 cycles/byte. Furthermore, in cases where the data output size is of the same magnitude as the data input size, it is beneficial to compress the result using a viable technique. As a reconfigurable framework, several compression algorithms (among them Huffman coding) are supported.

3. IN-MEMORY ANALYTICS ACCELERATION

In this section, we provide a broad system-level overview and describe in detail the hardware architecture including the PE microarchitecture, the hierarchical interconnect, the application mapping strategy, supported parallelism, execution models, and, finally, system-level memory management.

3.1. System Organization

From a system-level perspective, the in-memory analytics accelerator is situated at the last level of memory, between the last level memory device and the I/O controller, as shown in Figure 1. This is in contrast to previous work that situates processing at the main memory level [Guo et al. 2014] or within the last level memory itself, requiring a memory redesign [Paul et al. 2014b].

Sitting at the I/O interface, the accelerator intercepts and relays I/O commands as needed for typical operation. Acceleration mode is initiated when the host issues a read command to a specific disk location containing configuration information. The host can specify a set of file locations or directories which contain the data to be processed. Using standard I/O transfers, the accelerator can read these files, distribute data, and resume processing as specified by the particular configuration.

3.2. Accelerator Hardware Architecture

The accelerator comprises a set of single-issue RISC-style processing elements. Multiple PEs are interconnected with a two-level hierarchy that balances bandwidth with scalability, as described here:

3.2.1. PE Architecture. Each PE contains lightweight integer datapaths, data memory, scratch memory, an instruction ("schedule") table, and support for multi-input, multi-output lookup operations, as shown in Figure 3. The datapath supports typical operations, including addition/subtraction, shifting, comparison and equality operators, and fixed point multiplication, all of which are common to the evaluated kernel classes, making this lightweight datapath tailored to the analytics domain. Instructions are encoded in 32 bits, and 256 instructions can be mapped to each PE. Based on the communication and data distribution requirements discussed in Section 2.2, a total of 4kB lookup and data memory is available. At least 1kB is reserved for four LUTs, another 1kB each to the dedicated data input and output buffers, and up to 1kB for scratch memory. A portion of the scratch memory can also be used for LUTs if more functions are needed. Data input/output buffers and scratch memories are addressed using a 2-bit bank offset and 8-bit address in memory instructions. Similarly, a LUT offset is determined at compile time by the particular function it implements and its physical placement during kernel mapping. The PE specification is summarized in Table I.

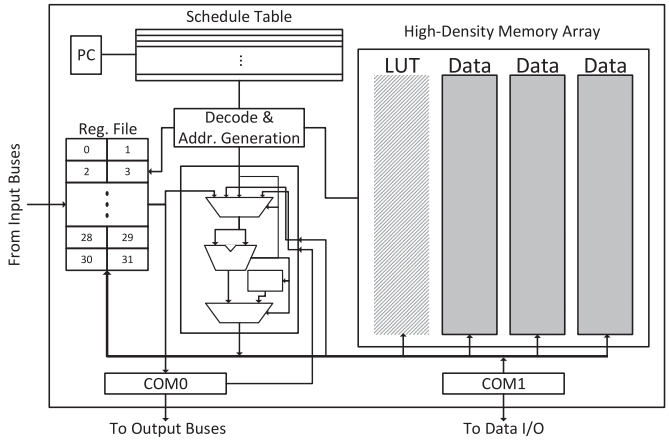

Fig. 3. The processing element architecture. Significant area is dedicated to the lookup and data/scratch memories.

Table I. Accelerator Architecture and Configuration for Processing Element, Memories, and Interconnect

Property	Description
Processing Element	Single Issue Integer ALU (Add/Sub/Shift/Comp.) 8-bit Fixed Point Multiplier
Memory	32×32 bit General Purpose Registers 256×32 bit Instruction Memory 2 kB I/O Memory Buffers 1 kB LUT Memory 1 kB Shared LUT/Scratch Memory
Intcnt	Cluster: 4 PE, Fully Connected Intercluster: Mesh

3.2.2. Interconnect Architecture. From the application analysis, we observed a range of communication requirements and consequently chose a suitable interconnect architecture to match. With 4kB/PE, applications like clustering and neural networks will require significant inter-PE communication, whereas applications like classification can achieve high throughput. To accommodate the various communication requirements, we use a two-level hierarchical interconnect, as shown in Figure 4. At the lowest level, groups of four PEs are fully connected in a *cluster*, which provides maximum on-demand communication between local PEs, thus satisfying the bandwidth requirements for clustering and neural networks. Each PE has a dedicated 8-bit output, to which all other PEs in the cluster connect, giving us a 32-bit intra-cluster interconnect. Data written to the bus are available on the next clock cycle; at the maximum operating frequency of 1.27GHz, this provides a maximum 40.6Gbps local link between PEs.

For the second hierarchical layer, we implement a routerless two-dimensional mesh interconnect; individual PEs within the cluster are responsible for communication with their inter-cluster neighbor. Specifically, the top left PE in one cluster communicates with the bottom right PE in the other cluster; similarly, the top right PE in one cluster communicates with the bottom left PE in the other, as shown in Figure 4. Communication is two-way, with one dedicated input and output bus from each PE to the

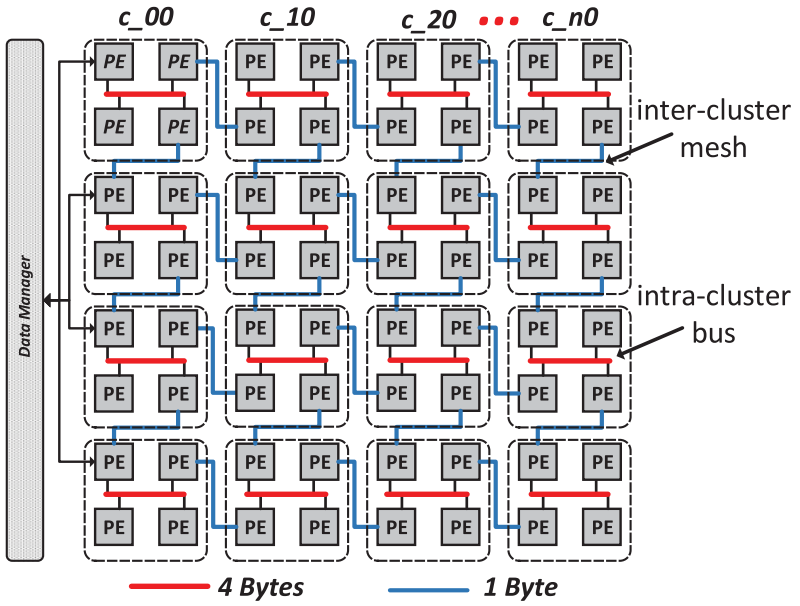

Fig. 4. The two-level hierarchy with a 6,416-cluster mesh. The design is routerless, and relies on individual PEs to communicate with adjacent clusters.

inter-cluster mesh. For the applications evaluated, clustering and neural networks are the most likely to utilize the inter-cluster bus; other analytics applications too large to map into one cluster, or for which lower kernel latency is required, can also make use of this resource. Furthermore, we note that the mesh reduces physical distances and wire length between clusters, making it scalable to larger numbers of clusters. It is preferable to global bus lines, which are commonly used in other frameworks.

3.3. Application Mapping

We developed a software framework for automatic application mapping into the proposed framework by modifying the general purpose MAHA *mapper* tool [Paul et al. 2014b], accounting for differences in the interconnect architecture and supported instruction set. The tool is written in C and maps applications according to the flow shown in Figure 5. An input kernel is first decomposed into a set of subgraphs, which are opportunistically fused to reduce overall instruction count. The fused operations are mapped to each PE under the given resource constraints using Maximum Fanout Free Cone and Maximum Fanout Free Subgraph heuristics [Cong and Xu 1998]. Operation groups (including LUTs and datapath operations) are mapped to individual PEs during placement and distributed in such a way that communication between PEs is minimized, following the specific interconnect architecture. Finally, the communication schedule is generated, which statically schedules intra- and inter-cluster communications among PEs. In this manner, the software tool generates the configuration file used to program the accelerator.

Several operation classes are supported, including bit-sliceable operations such as logic or arithmetic, as well as complex (LUT-based) operations. Memory access instructions, including those for requesting new data from the data manager, are inserted in appropriate locations in individual PEs as well as the gateways between clusters. Thus, this flexible software framework complements the reconfigurable nature of the

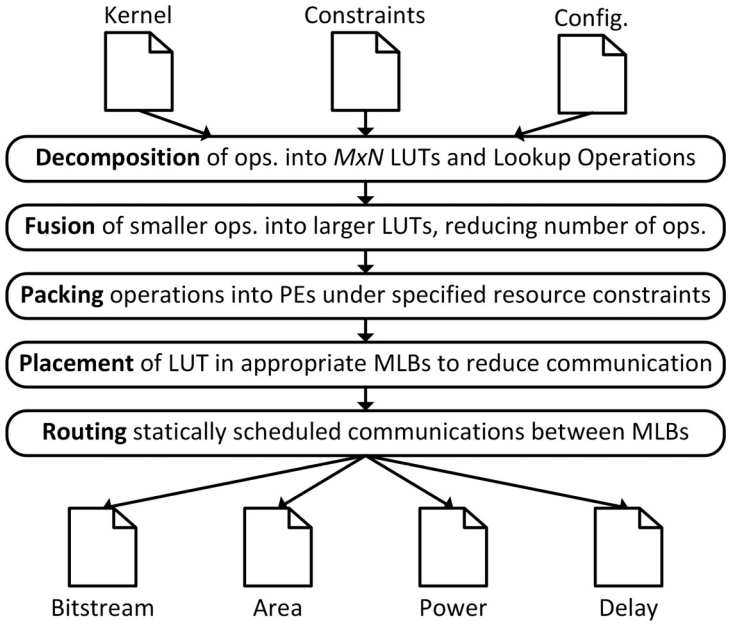

Fig. 5. Software flow for the modified *mapper* tool, which takes as its input a kernel code file, constraints file, and configuration file.

hardware, enabling the mapping of different sizes of applications to different hardware configurations.

3.4. Parallelism and Execution Models

We note that the described hardware architecture offers a high degree of flexibility for kernel execution. First, because each PE retains its own local instruction and data caches, there is implicit support for both SIMD and MIMD execution. Second, there is flexibility in how the analytics applications can be mapped. As noted from the application analysis, these applications can generally exploit different types of parallelism, including instruction-level (ILP) and data-level (DLP). The particular execution model is in part determined by the application mapping. For example, four iterations of NBC can be mapped to a single cluster in several ways; assuming four vectors comprised of four 8 bit variables $\langle A_0B_0C_0D_0, \ldots, A_3B_3C_3D_3\rangle$ executing on $\langle PE_0, \ldots, PE_3\rangle$, we can have:

(1) Each PE independently processes each vector (DLP).
(2) Two PEs $\langle PE_0, PE_1\rangle$ coordinate to process a single vector (ILP/DLP).
(3) Four PEs $\langle PE_0, \ldots, PE_3\rangle$ coordinate to process a single vector (ILP).

Cases 2 and 3 require the use of the intra-cluster bus, whereas Case 1 does not. Furthermore, the MIMD execution model also enables task-level parallelism. For example, multiple PEs can be pipelined, where one is responsible for dataset decompression or preprocessing, and the other three within the cluster handle the primary kernel using any of the preceding execution models.

3.5. System-Level Memory Management

Data management is a crucial aspect of big data processing, and ensuring that PEs are not starved while maintaining efficient and scalable on-chip data transfer is key. In the

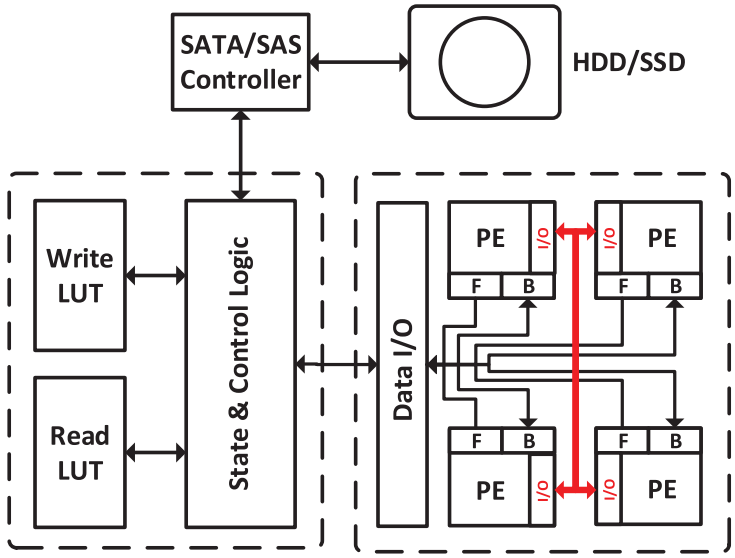

Fig. 6. Schematic of the system-level memory management. A SATA/SAS interface transfers data to/from the LLM. Read and Write LUT controls are configured at compile time and control the behavior of the memory management. Data are distributed through the cluster, separate from the intra-cluster bus, allowing data transfer and computation to overlap.

proposed accelerator, a system memory manager is responsible for transferring data streamed in by the onboard SATA/SAS controller (Figure 6). Configurable Read and Write memories contain the *data schedule*, determined at compile time. The memory manager intercepts SATA commands from the OS, including what files or memory locations to retrieve and where results can be safely written back.

Within each cluster, a simple data router called the Data I/O Block (DIOB) (Figure 6) accepts a directed data transfer and distributes it to the correct Cluster and PE. Directed data transfers are preceded by a 9-bit header containing an MSB of 1 (header ID), a 5-bit Cluster ID, and a 2-bit PE ID; after that, each data byte is automatically forwarded to the appropriate PE. Note that 1 bit is reserved as a "continuation" bit which will allow expansion by using multiple bytes to specify the Cluster ID. Data transfer is localized to rows of Clusters; DIOBs cannot route data to clusters in their columns. Instead, the main data manager, shown in Figure 4, initiates row-wise transfers as data are received.

Internally, dedicated data lines to and from the PEs are used for I/O, rather than reusing the intra-cluster bus, allowing data transfer to overlap with execution. Each PE contains two data buffers, *Front* and *Back* (labeled "F and "B" in Figure 6), which are reserved for I/O. If the data can be processed in a single iteration, and the output data size matches the input data size, the results can be read from the backbuffer while data in the front buffer is processed. When ready, the PE swaps the roles of the buffer. However, for clustering and neural networks, the intermediate values may not be of interest, or, in other cases, the data output sizes may be significantly smaller than data input. In these cases, scratchpad memory can hold intermediate results, and the data manager does not need to read back values between computations. This allows the system to efficiently coordinate the parallel processing of a large volume of data.

This memory management scheme is flexible enough to contend with the various uses of the fabric, including cases where PE data transfer requirements differ between

Clusters or even between PEs (e.g., ILP or TLP modes). Static scheduling is leveraged to ensure PEs have sufficient data to process and that there are no contentions for the data I/O bus. By having a single DIOB per cluster and not relying on global data lines, the architecture can scale to an arbitrary number of clusters.

4. METHODS

In this section, we describe the experimental setup and FPGA-based hardware emulator for functional and timing validation, and we provide implementation details of the three representative kernels on the accelerator.

4.1. Experimental Setup

We obtain initial functional and timing verification of the accelerator using the hardware emulation platform described in Section 4.2. For CPU and GPU implementations, we use a desktop system running Ubuntu server 12.04 x64, with an Intel Core2 Quad Q8200 2.33GHz CPU [Intel Core2 Quad Q8200], 8GB of 800 MHz DDR2 memory, and a 384-core NVIDIA Quadro K2000D GPU [NVIDIA]. The same datasets were used for functional verification on the emulator, the CPU, and the GPU.

For CPU results, NBC and KMC were written in C and optimized with processor-specific MARCH flags and -o3 compiler optimization. Latency results were obtained using the C function clock_gettime, with the CLOCK_PROCESS_ CPUTIME_ID high-resolution clock specifier [CLOCK GET TIME]. CNN code was obtained from Theano [Bergstra et al. 2010] for the sample network provided in the documentation. For GPU results, the NBC and KMC kernels were written in C with the CUDA extensions and compiled with the NVIDIA C Compiler (NVCC). Latency results were obtained using the built-in NVIDIA profiling tools for high-precision kernel and PCIe data transfer timing [NVIDIA]. CNN latency was obtained using high-resolution timing functions available in Python.

Energy results were derived from the latency and estimated power consumption of the CPU based on 25% maximum TDP (95W), scaled by the number of active cores and threads (1 of 4), for a conservative power estimate of 6W. We believe this is estimated to the benefit of the CPU because the actual power consumption is likely to be greater than 6W. Similarly, we estimate GPU power consumption based on the number of active cores and threads; since GPU kernels were launched with sufficient blocks and threads per block to ensure maximum occupancy and core utilization, the TDP (51W) is used. For isoarea comparisons, the CPU area is taken to be 1/4 of the die shown [George et al. 2007], which we estimate to be $25mm^2$ at 45nm.

The accelerator was also implemented in Verilog (the same used in the emulation platform). This was synthesized using Synopsys Design Compiler and a 90nm cell library. The maximum clock frequency of 1.27GHz, as reported by the synthesis tool, was used. Power, performance, and area estimates for the non-memory components were extracted from this model. CACTI [CACTI] was used for the power, performance, and area of the memories, also estimated for a 90 nm process.

4.2. Functional Verification

The proposed accelerator was functionally verified using the hardware emulation platform pictured in Figure 7. Two separate FPGA boards were used: first, a Nios II/f softcore processor [Nios II] was mapped to a Terasic DE0 with Cyclone III FPGA, labeled “Host”; second, the accelerator and flash interface were mapped to a Terasic DE4 with Stratix IV FPGA, labeled “SSD.” The flash interface facilitates communication between the “LLM,” a 2GB SD card, and the accelerator. The two boards communicate over SPI using the GPIO ports.

The hardware was developed in Verilog and compiled using the Altera Quartus II software [Altera 2016] with optimizations for speed. A single cluster (4 PEs) is

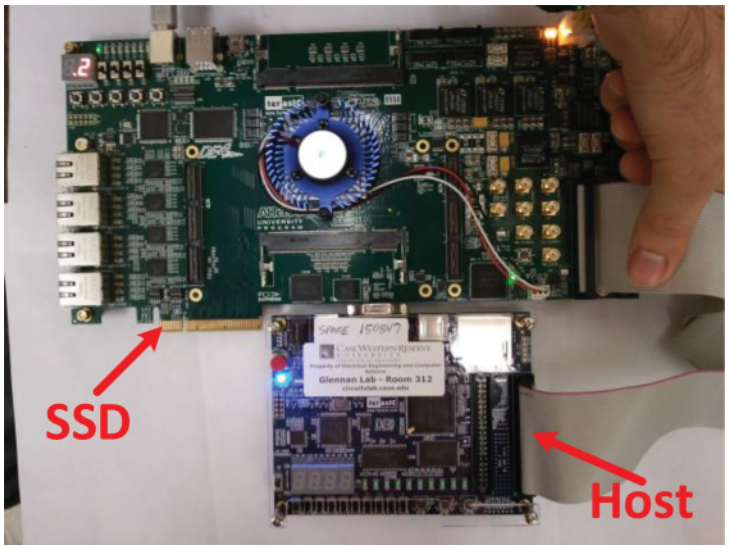

Fig. 7. Photograph of the hardware emulation platform. Two FPGA boards, one "SSD" and one "Host," are used for functional verification of the accelerator.

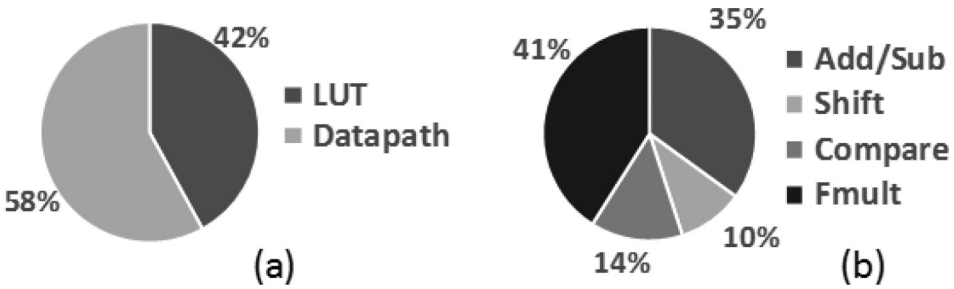

Fig. 8. Breakdown of the average (a) lookup table and datapath instructions for kernels mapped to the framework, and (b) the four most common datapath (non-memory access) operations.

used in the hardware prototype, though for synthesis and simulation results, a two-cluster (8 PE) implementation was used. The three kernels, NBC, CNN, and KMC, were mapped to the framework using techniques highlighted in Section 4. For NBC and KMC, randomly generated 1GB datasets were processed; for CNN, samples from the MNIST handwritten digit dataset were used instead.

For the emulated CPU processing, data are transferred from the "SSD" to the "Host" over the SPI connection, emulating the functionality of the SATA interface. After processing, data are similarly written back to the "SSD." The results are then offloaded to a desktop system, the SD card is reformatted, and the original dataset is reloaded. Accelerator-based processing follows, initiated by the "Host" sending the kernel specification and data location. This initiates the transfer and processing stages. Data are written back to designated locations once processing is complete, and results are again offloaded for analysis. For all three kernels, accelerator output and CPU output were compared and found to match. Figure 8(a) shows the average instruction breakdown for the LUT versus Datapath, and Figure 8(b) shows the same for the four most frequent datapath (non-memory access) instructions.

4.3. Analytics Kernels

4.3.1. Classification. For testing, we implement NBC, although the principles apply similarly to classification using SVM [Boser et al. 1992], especially the linearly separable case. NBC makes heavy use of the available datapath operations but does not require any of the scratchpad memory; the register file is used to store intermediate results. Training is performed offline (e.g., on the host) on a small subset of the data, and the probabilities are stored as 8-bit fixed point. No complex functions are used, and no inter-PE or inter-cluster communications are required.

4.3.2. Neural Network. As with NBC, the neural network implementation (specifically, CNN) also requires two stages, one for training and the other for evaluating the input. The input is a set of small grayscale images containing handwritten digits from the MNIST database [Lecun and Cortes 2016]. The network consists of convolutional layers with nonlinear subsampling operations immediately following. The test network is based on the example CNN from Theano [Bergstra et al. 2010].

Input images are stored in 8-bit grayscale in the data memory, and a copy of each image, along with the filter coefficients, is distributed to each PE. The full convolution operation is evaluated over multiple cycles using a multiply-accumulate unit. A nonlinear activation function, tanh, is approximated with lookups, using the most significant bit as the sign bit, with the remaining bits used for the decimal. Portions of the same image are processed in parallel within a single 4 PE cluster, and output data are shared among the PEs upon completion of a single stage.

Like KMC, a mix of datapath and lookup operations is used. While this implementation requires significantly more intra-cluster communication, individual network evaluation latency is decreased, and clusters can independently evaluate different input data. We note that the approach applied to CNN can be applied to a more general Artificial Neural Network (ANN), where the primary difference will be increased utilization of the intra-cluster bus.

4.3.3. Clustering. Parallel implementations of KMC have been investigated with respect to large datasets using MapReduce [Zhao et al. 2009], GPUs [Farivar et al. 2008], and other multiprocessor, multinode machines [Kraj et al. 2008]. We implement parallel KMC on the accelerator, testing two scenarios, in which (i) one processing cluster is used and (ii) multiple processing clusters are used. The number of processing clusters is not solely a function of the number of vectors in the dataset; additional consideration is given to the desired balance between spatial and temporal computing, in particular, the energy/latency tradeoff. In the first case, communication within the cluster occurs after each point in local PE memory has been assigned to a group and the group summary statistics have been computed. The same principles apply to the second case, except summary statistics must be additionally shared between processing clusters.

During the KMC operations, a mix of datapath and lookup operations are used. Namely, the Euclidian distance measure is used, requiring SQUARE and SQRT functions, which are stored in fixed point in the LUTs, in addition to datapath addition. Magnitude comparisons are also performed using datapath operations.

4.4. Dataset Compression

Dataset compression using Huffman-based entropy encoding is employed to reduce data transfer volume. Decompression is straightforward, implemented as a table lookup, in total requiring 14 cycles: 1 for the actual lookup and the remainder for data management and formatting for the NBC, KMC, or CNN routines. Compression is slightly modified, using padding to enable the efficient lookup. For example, in the

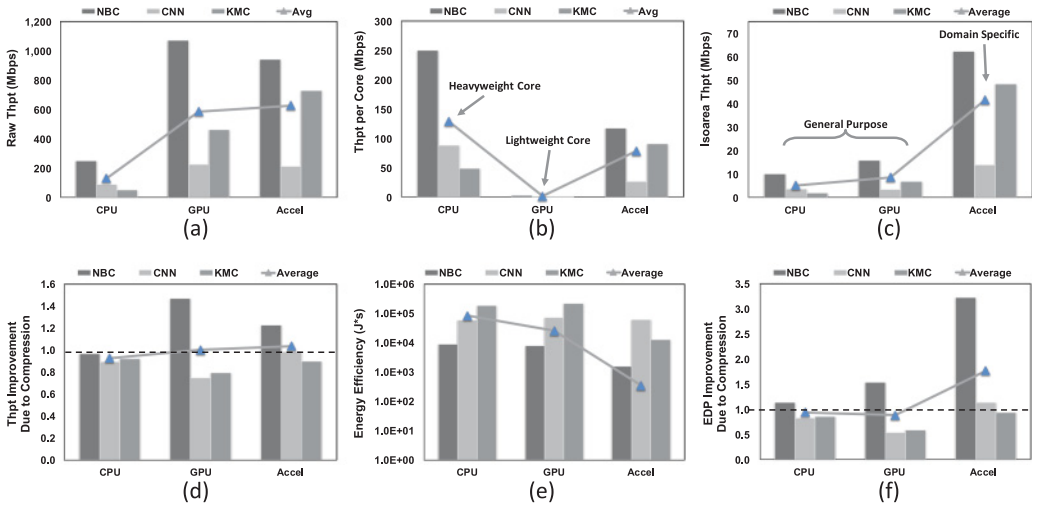

Fig. 9. Comparison of (a) raw throughput, (b) throughput per core, (c) isoarea throughput, (d) throughput improvement from compression, (e) energy efficiency, and (f) improvement in energy efficiency from compression. From (a), we observe that, on average, the accelerator outperforms both the CPU and the GPU in terms of raw throughput. However, architectural differences must be considered, and in (b) we observe how this affects per-core performance. Finally, (c) demonstrates the effect of architecture-level domain customization, which heavily favors the analytics-specific accelerator. We observe in (d) the improvement in raw throughput from compression, which varies based on the dataset and decompression speed. Finally, (e) and (f) demonstrate the significant increase in energy efficiency of the accelerator compared with the CPU and GPU, as well as the improvement in energy efficiency due to compression.

case of the NBC data, the longest Huffman code was determined to be 4 bits, and the shortest was 2 bits. Values shorter than 4 are left-aligned in a half-byte and padded with all remaining combinations. For code lengths greater than 4 bits, an extra 2 cycles are needed to track intermediate values.

As a reconfigurable framework, the user has the freedom to choose one of several compression/decompression algorithms and implementations; the choice depends on the data and which algorithm can provide the best compression ratio. Run length coding or an LZ variant [Ziv and Lempel 1977, 1978] can be used, as long as the inequality (Section 2.3) holds.

5. RESULTS

The implementations of all three applications resulted in identical outputs for the proposed accelerator, CPU, and GPU, confirming functionality of the implementations. The following is a comparison of throughput (at iso-area) and energy-efficiency (at iso-throughput) among the three platforms.

5.1. Throughput

We begin the throughput analysis by comparing the raw throughput of applications on the target platforms (Figure 9(a)). Both the GPU and accelerator outperform the CPU considerably, about 6× on average. While the CPU implementation was single threaded, a multithreaded/multicore implementation is unlikely to bridge the performance gap; even if we assume CPU performance to scale linearly with the number of cores for all applications—an unlikely scenario—this would at most achieve 66% the performance of the other systems (assuming a quadcore CPU) and would certainly increase power consumption.

Interestingly, when considering raw throughput, the GPU outperforms the accelerator by about 1.2× for NBC, and the two are nearly equivalent for CNN; but, for KMC, the accelerator outperforms the GPU by about 1.6×. This can be explained by considering the GPU warp-based execution model: NBC achieves higher performance through the use of coalesced memory accesses; meanwhile, KMC suffers from both uncoalesced memory accesses and significant branch divergence, leading to poor performance [Rogers et al. 2013].

Iso-Comparison: Given the architectural differences between the processing elements in the three platforms, we provide two iso-comparisons in Figure 9(b) and (c), which show Throughput per Core and isoarea throughput, respectively, in Megabits per second (Mbps). The main takeaway is the effect of core size and domain specificity: The CPU (a "heavyweight" core) yields the highest per-core throughput, while the GPU (a "lightweight" core) yields the lowest. Between them is the accelerator, which provides moderate per-core throughput.

Considering area efficiency, we observe relatively low average isoarea throughput for the general purpose platforms (CPU, GPU), whereas the accelerator yields significantly higher results. This is primarily due to the domain specificity, both the inclusion of functional units which are related and useful to the particular domain, and the exclusion of functional units that are less relevant. In other words, the accelerator cores are more focused toward the target domain, resulting in a higher percentage of useful area on the chip.

Application Level: At the application level, we generally observe the highest average isoarea throughput for NBC in all platforms, followed by KMC and finally CNN. There are two reasons for this difference: First, the average number of operations applied to each byte during processing varies from NBC (least) to CNN (most); second, the inter-cluster communication overhead, which similarly varies from NBC (no intercluster communication) to KMC (frequent communication, but a small amount of data), and finally CNN (frequent, with a large amount of data). Generally, applications highly amenable to parallelization (DLP) will benefit most from a larger number of cores, up to a point [Esmaeilzadeh et al. 2013], which in this case favors the GPU. The mesh inter-cluster interconnect supports greater throughput for small, frequent data transfers, leading to a significant performance increase for KMC when using the accelerator.

Effect of Compression: We conclude our discussion of throughput by observing the effect of dataset compression on performance. Figure 9(d) shows that, on average, the CPU was adversely affected, the GPU was unaffected, and only the proposed accelerator experienced an increase in raw throughput by compressing the dataset. Generally, we can expect to see an increase in throughput only when the decompression routine overhead is less than the difference in transfer latency between the compressed and uncompressed datasets (Figure 2). This was not the case for any of the CPU kernels, and therefore it experienced the greatest decline in performance among the three platforms. Note that, among the kernels, the lowest decline was experienced by NBC; in fact, across all three platforms, NBC experienced the highest performance improvement relative to the other kernels while KMC experienced the lowest. This is primarily due to the high compression ratio achieved for the NBC dataset and the low compression ratio achieved for KMC.

The GPU saw a significant increase in throughput for the NBC kernel and a steep decline for the others. Compared with CPU, the decline in performance for CNN and KMC is far more drastic on the GPU. The same factors affecting the raw throughput without compression are exacerbated by the addition of the compression routine, which itself suffers from warp divergence. Finally, the accelerator demonstrates the only positive average improvement due to dataset compression among the three platforms,

which indicates that the decompression routine had sufficiently low overhead on this platform.

5.2. Energy Efficiency

For big data analytics, throughput is only one aspect. Arguably, the more critical design consideration, especially in the coming years, is the energy efficiency of the system. Figure 9(e) shows the Energy Delay Product (EDP) of the applications on the three platforms. Note that a lower EDP implies greater energy efficiency. The CPU appears as the least efficient platform on average, followed by the GPU. The accelerator achieves the design goal of high energy efficiency and is on average two orders of magnitude more energy efficient than either the CPU or GPU.

From an application standpoint, the most energy efficient on all three platforms was NBC, which is primarily due to the small size of the kernel. For the accelerator, this results in 61× and 11× improvement versus the CPU and GPU; for CNN, we observe improvements of 84× and 89× versus CPU and GPU; and for KMC, we observe improvements of 798× and 67× versus CPU and GPU, respectively. The relative EDP improvement over CPU, even when comparing to the GPU, is quite large and primarily stems from the already significant difference in compute energy and, ultimately, from the difference in compute latency between the CPU and other platforms.

On the accelerator, the mix of lookup operations and customized datapath, as well as the reduced transfer energy and latency, result in an average 212× improvement over single-threaded CPU and 74× improvement over GPU. On average, the majority of this improvement (>80%) is attributed to the lowpower, domain-specific architecture and interconnect network, while the remainder derives from energy saved in data transfer.

Efficiency and Dataset Compression: Finally, we consider the effect of dataset compression on the overall energy efficiency of the system (Figure 9(f)). Similar to the effect on raw throughput, we observe that the proposed accelerator is the only platform to benefit from the compression. This is primarily driven by the EDP improvement of NBC, but CNN also demonstrates a slight efficiency improvement. However, the low compression ratio for KMC yields no increase in efficiency on any platform.

6. DISCUSSION

In this section, we discuss different aspects of the experimental results with regards to benchmark performance, the memory-centric architecture, and the application scope, and we provide a thorough differentiation from related work.

6.1. Performance and Parallelism

All three benchmark applications performed well on the accelerator in terms of throughput and energy efficiency. Moreover, the accelerator can rival the much larger GPU, even in raw throughput, while using significantly less power. Depending on the mapping, the application kernels are amenable to ILP and DLP. The high-bandwidth intra-cluster communication network allows greater flexibility in the mapping, so a wide range of kernels can take advantage of any inherent parallelism. This was observed for NBC, which can exploit ILP in training and DLP in classification. CNN and KMC can both benefit from a mix of these two models as well. Finally, task-level parallelism can also be exploited, for example, pipelining the dataset decompression and dataset processing in adjacent PEs or clusters as needed. This task-level parallelism can even be taken one step further, running multiple classifiers (NBC and SVM, for example) or other statistical models simultaneously for forecasting.

6.2. Scalability

The results presented used an 8-PE (two cluster) version of the accelerator. With more clusters, we expect to see a performance increase while maintaining comparable levels of energy efficiency, given the scalable interconnect architecture. In contrast to other Coarse Grained Reconfigurable Arrays (CGRAs) such as RaPiD [Ebeling et al. 1996], MATRIX [Mirsky and DeHon 1996], or MorphoSys [Singh et al. 2000], which contain some form of global bus along rows or columns at the top level of hierarchy, the second-level interconnect of the accelerator uses a routerless mesh to provide adequate bandwidth for the analytics applications while keeping physical connections between PEs short.

Generally, the performance of any accelerator for big data will have some inherent limitations due to available I/O bandwidth since it remains constant even when adding additional memory capacity. However, one can assume that the data already reside in the last level memory of the system. With a typical platform, these data will have to be read out of the disk, passing through the system I/O controller, to be processed by the CPU or GPU. Conversely, the proposed system places the accelerators between the last level of memory and the I/O controller. This results in two use cases: (i) each accelerator can operate independently of the others with limited host intervention, using data on the disk to which they are connected (typical of kernels amenable to map/reduce); or (ii) accelerators must communicate intermediate results, in which case host intervention is required. In the first case, the physical limitations of the system (e.g., pin count) would not cause a bottleneck because processing is distributed among the disks. In the second case, system performance would be reduced only if the communication requirement exceeds that available for the system. If possible, applications should be mapped in such a way that only intermediate results (e.g., summary statistics) need to be communicated in order to minimize the negative impact.

Furthermore, note that effective data compression can go a long way in addressing the limitation imposed by memory bandwidth. Since our reconfigurable accelerator is amenable to efficient implementation of on-chip decompression/compression logic, as described in Section 4.4, it can significantly help in reducing the bandwidth requirement for diverse application kernels. Our results show that even a simple compression routine can significantly reduce the input data size, leading to an improvement in both energy-efficiency and scalability for handling larger datasets.

6.3. Transfer Energy and Latency

Hiding data transfer latency is ubiquitous to modern computing, used in caching systems and employed by off-chip accelerators (FPGA and GPUs) to overlap processing with data transfer. Frameworks like CUDA Streams [NVIDIA] provide higher levels of abstraction to programmers for hiding transfer latency. Modern GPU hardware contains on-board DMA engines which can transfer data directly from the last-level memory without CPU intervention; when paired with streams, this greatly reduces or entirely eliminates transfer latency after the initial transfer. While this does help improve performance, it cannot hide the data transfer *energy*, which has been shown to consume a large percentage of overall power consumption for high-performance systems [Bergman et al. 2008].

Therefore, bringing computation closer to the memory has been investigated as a means to reduce transfer energy and latency. With the advent of 3D integration, the efficiency and cost-effectiveness of Processor-in-Memory (PIM) architectures has improved [Liu et al. 2005]. However, there are several important considerations: (i) data must still be transferred into the memory, which must pass through the I/O controller, and would therefore be subject to the physical bandwidth limitations mentioned in

the previous comment; (ii) the amount of physical memory available to PIM architectures is significantly less than the last-level memory available to a given system through the use of port multipliers or expanders, meaning that applications whose data size exceeds hundreds of gigabytes will need to be processed piecemeal and sequentially in the memory; (iii) while PIM solutions can reduce data transfer latency and energy, they do not aim at developing a tailored accelerator architecture for data-intensive kernels by customizing the datapath elements and interconnection fabric; and (iv) as general-purpose processors, PIM architectures are unlikely to perform sufficiently low-overhead data compression/decompression, making it difficult to reduce the already limited memory bandwidth in modern computing systems. By comparison, the proposed accelerator offers greater scalability and energy-efficiency compared to PIM architectures for future growth of big data analysis.

In the ultimate case, processing can be moved *into* the last-level memory itself, removing the need for external data transfer entirely. This, however, would require a fully custom-designed last-level memory and the replacement of existing drives. The proposed accelerator overcomes this difficulty by operating at the interface, rather than inside the memory itself. Nevertheless, with the headway in "universal memory" research (i.e., a memory technology that can be used for anything from CPU cache to main memory to secondary storage), it is plausible to integrate efficient processing at other levels of the memory hierarchy to further reduce data movement.

6.4. Memory-Centric Processing

The proposed architecture is *memory-centric* in that the primary component within each PE is the memory. Synthesis results estimate that upward of 90% of the PE area and power consumption is taken by the memory array. Therefore, higher density, lower power, and more reliable memories have the potential to vastly improve the energy efficiency and reliability of the accelerator. Many emerging memory technologies are not only promising in terms of integration density, efficiency, and reliability, but also offer nonvolatility.

6.5. Analytics and Machine Learning

Analytics is a rapidly evolving field, and, for many businesses, the ability to make more informed operational decisions is driven by smarter data processing. A reconfigurable accelerator tuned to the analytics domain can potentially reduce the costs associated with big data analysis. The three benchmark applications represent a diverse range of algorithms, from statistical classification to biologically inspired neural networks, but by tailoring the accelerator design to common execution patterns and providing a highly flexible interconnect, the accelerator is general enough to handle other applications from this domain, including basic statistics (e.g., mean, median, variance, histograms, etc.), other classifiers (e.g., SVM and decision trees), and different clustering techniques (e.g., hierarchical), as well as standard ANNs. The framework can also be expanded to include support for single precision floating point operations, catering to applications where more precision is required than can be offered by the fixed point datapath.

7. RELATED WORK

There are many examples in literature of accelerating big data analytics workloads using different hardware frameworks, including FPGA and GPGPU, as well as distributed computing systems with MapReduce. Here, we describe the related work and how the proposed architecture differs, primarily focusing on FPGA and GPGPU rather than other CGRAs or many-core systems, because the former tend to be more commonly used in larger scale enterprise systems.

7.1. FPGA Analytics

One of the most famous uses of FPGA to accelerate analytics is the IBM Netezza platform [Helmreich and Cowie 2006], where database queries are sifted such that only the relevant results are transferred to the host for processing, effectively compressing the data. Other query processing and join processing systems exist [Sukhwani et al. 2012] [Halstead et al. 2013], along with more general regular expression acceleration for text analytics [Atasu et al. 2013] or FPGA systems for general data mining or search applications [Woods and Alonso 2011; Putnam et al. 2014]. Additionally, FPGAs have been used to implement a custom accelerator within a Hadoop MapReduce framework [Neshatpour et al. 2015].

One common theme among FPGA-based systems is the focus on raw performance. In many cases, the significantly reduced latency associated with FPGA acceleration will reduce power consumption over a comparable CPU implementation, but this is not a scalable architecture in the context of overall power draw. More efficient reconfigurable architectures, with domain-specific features such as that presented here, can serve to improve performance while maintaining or even reducing overall power consumption. Finally, the high flexibility of the FPGA systems is such that any future change to the application/kernel/implementation will require a redesign of the underlying hardware description (e.g., Verilog or VHDL). Frameworks with higher-level software support can compile high-level descriptions into hardware, making them somewhat immune from this. Nevertheless, an ideal system would be programmable, as in software, and potentially easier to update and manage.

7.2. GPGPU Analytics

Like FPGAs, GPUs have also been used to accelerate analytics operations in the context of big data. While the architecture is not as flexible as FPGA, the intrinsic parallel processing in GPGPUs makes them attractive for a wide variety of analytics and mining applications, including various clustering algorithms [Farivar et al. 2008; Stoffel and Belkoniene 1999; Wu et al. 2009] and general MapReduce functions [Chen et al. 2012]. In addition, GPU database acceleration has been investigated [Govindaraju et al. 2004; Bakkum and Skadron 2010], as well as accelerating Network Intrusion Detection Systems [Bandre and Nandimath 2015], graph processing [Wang et al. 2015; Fu et al. 2014], and deep learning from neural networks [Wang et al. 2014]. High bandwidth PCIe interconnects, as is standard for modern GPUs, enables multiway data streaming, which is complemented by the GPU hardware architecture and memory hierarchy.

However, like FPGAs, GPUs are not domain-specific. Fitting an analytics problem, including required computations and communication between parallel elements, to the GPU architecture may not result in optimal energy efficiency, as we observed in the KMC and CNN benchmarks.

8. CONCLUSION

We presented a domain-specific, memory-centric, adaptive hardware acceleration frame work that operates at the last level of memory and is capable of running advanced analytics applications on a large volume of data. The architecture is tailored to the functional and inter-PE communication requirements of the common kernels. A software tool for automatic application mapping is developed that takes advantage of the features of the underlying architecture. The framework is functionally validated using a hardware emulation platform with implementations of three common analytics kernels. Synthesis results and subsequent comparison with CPU and GPU implementations of these kernels demonstrate excellent improvements in energy efficiency, which we attribute primarily to the domain-specificity and the proximity to the data.

In-accelerator dataset decompression is also shown to be a viable option by improving overall system energy efficiency. Using the proposed accelerator, reduced latency and power requirements will translate into faster results and lower energy costs, both of which are crucial to many business applications. Finally, as a memory-centric hardware acceleration framework, it can leverage promising properties of emerging memory technologies (e.g., high-density, low-access energy, and nonvolatility) to further improve area- and energy-efficiency. Future work will include extending the automatic application mapping front-end to support kernel description in higher level languages and investigation of other compression techniques, including LZ variants, as well as test chip fabrication for the acceleration framework.

REFERENCES

Altera. 2016. Quartus II Subscription Edition. Retrieved March 2016, from http://www.altera.com.

Mauricio Araya-Polo, Javier Cabezas, Mauricio Hanzich, Miquel Pericas, Felix Rubio, Isaac Gelado, Muhammad Shafiq, Enric Morancho, Nacho Navarro, Eduard Ayguade, and others. 2011. Assessing accelerator-based HPC reverse time migration. *IEEE Transactions on Parallel and Distributed Systems* 22, 1 (2011), 147–162.

Kubilay Atasu, Raphael Polig, Christoph Hagleitner, and Frederick R. Reiss. 2013. Hardware-accelerated regular expression matching for high-throughput text analytics. In *Proceedings of the 23rd International Conference on Field Programmable Logic and Applications (FPL'13)*. IEEE, 1–7.

Peter Bakkum and Kevin Skadron. 2010. Accelerating SQL database operations on a GPU with CUDA. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 94–103.

S. R. Bandre and J. N. Nandimath. 2015. Design consideration of network intrusion detection system using hadoop and GPGPU. In *Proceedings of the 2015 International Conference on Pervasive Computing (ICPC'15)*. 1–6. DOI:http://dx.doi.org/10.1109/PERVASIVE.2015.7087201

Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William H. arrod, Kerry Hill, Jon Hiller, and others. 2008. Exascale computing study: Technology challenges in achieving exascale systems. *Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO)*. Tech. Rep. 15 (2008).

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: A CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy'10)*. Oral Presentation.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*. ACM, 144–152.

CACTI. Online. Retrieved from http://arch.cs.utah.edu/cacti/.

Linchuan Chen, Xin Huo, and Gagan Agrawal. 2012. Accelerating mapreduce on a coupled CPU-GPU architecture. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE Computer Society Press, 25.

clock-gettime(3) - Linux main page. Retrieved from. http://linux.die.net/man/3/clock_gettime.

Jason Cong and Songjie Xu. 1998. Technology mapping for FPGAs with embedded memory blocks. In *Proceedings of the 1998 ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays*. ACM, 179–188.

CUDA Profiling Tools Interface. Retrieved from https://developer.nvidia.com/cuda-profiling-tools-interface.

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

Richard O. Duda, Peter E. Hart, and others. 1973. *Pattern Classification and Scene Analysis*. Vol. 3. Wiley New York.

Carl Ebeling, Darren C. Cronquist, and Paul Franklin. 1996. RaPiDreconfigurable pipelined datapath. In *Field-Programmable Logic Smart Applications, New Paradigms and Compilers*. Springer, 126–135.

Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2013. Power challenges may end the multicore era. *Communications of the ACM* 56, 2 (2013), 93–102.

Reza Farivar, Daniel Rebolledo, Ellick Chan, and Roy H. Campbell. 2008. A parallel implementation of k-means clustering on GPUs. 13, 2 (2008).

Nir Friedman, Dan Geiger, and Moises Goldszmidt. 1997. Bayesian network classifiers. *Machine Learning* 29, 2–3 (1997), 131–163.

Zhisong Fu, Michael Personick, and Bryan Thompson. 2014. MapGraph: A high level API for fast development of high performance graph analytics on GPUs. In *Proceedings of Workshop on GRAph Data Management Experiences and Systems*. ACM, 1–6.

Lee Garber. 2012. Using in-memory analytics to quickly crunch big data. *Computer* 45, 10 (2012), 16–18.

Varghese George, Sanjeev Jahagirdar, Chao Tong, Ken Smits, Satish Damaraju, Scott Siers, Ves Naydenov, Tanveer Khondker, Sanjib Sarkar, and Puneet Singh. 2007. Penryn: 45-nm next generation intel® core 2 processor. In *Proceedings of the IEEE Asian Solid-State Circuits Conference (ASSCC'07)*. IEEE, 14–17.

Naga K. Govindaraju, Brandon Lloyd, Wei Wang, Ming Lin, and Dinesh Manocha. 2004. Fast computation of database operations using graphics processors. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*. ACM, 215–226.

Alexander Gray. 2013. Analyzing Massive Datasets. Retrieved from http://www.skytree.net/resources/.

Qi Guo, Nikolaos Alachiotis, Berkin Akin, Fazle Sadi, Guanglin Xu, Tze Meng Low, Larry Pileggi, James C. Hoe, and Franz Franchetti. 2014. 3d-stacked memory-side acceleration: Accelerator and system design. In *Proceedings of the Workshop on Near-Data Processing (WoNDP) (Held in Conjunction with MICRO-47.)*.

Robert J. Halstead, Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Sameh Asaad, and Brijesh Iyer. 2013. Accelerating join operation for relational databases with FPGAs. In *Proceedings of the IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'13)*. IEEE, 17–20.

Stephen C. Helmreich and Jim R. Cowie. 2006. Data-centric computing with the netezza architecture. (2006).

David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.

Intel Core2 Quad Processor Q8200. Retrieved from http://ark.intel.com/Products/Spec/SLG9S.

Adam Jacobs. 2009. The pathologies of big data. *Communications of the ACM* 52, 8 (2009), 36–44.

Robert Karam, Ruchir Puri, and Swarup Bhunia. 2016. Energy-efficient adaptive hardware accelerator for text mining application kernels. *IEEE Transactions on Very Large Scale Integration* 24, 12 (Dec. 2016).

Robert Karam, Ruchir Puri, Swaroop Ghosh, and Swarup Bhunia. 2015a. Emerging trends in design and applications of memory-based computing and content-addressable memories. *Proceedings of the IEEE* 103, 8 (2015), 1311–1330.

Robert Karam, Kai Yang, and Swarup Bhunia. 2015b. Energy-efficient reconfigurable computing using spintronic memory. In *Proceedings of the 58th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'15)*. IEEE, 1–4.

Piotr Kraj, Ashok Sharma, Nikhil Garge, Robert Podolsky, and Richard A. McIndoe. 2008. ParaKMeans: Implementation of a parallelized K-means algorithm suitable for general laboratory use. *BMC Bioinformatics* 9, 1 (2008), 200.

Steve LaValle, Eric Lesser, Rebecca Shockley, Michael S. Hopkins, and Nina Kruschwitz. 2011. Big data, analytics and the path from insights to value. *MIT Sloan Management Review* 52, 2 (2011), 21–31.

Yann Lecun and Corinna Cortes. 2016. The MNIST database of handwritten digits. Retrieved from http://yann.lecun.com/exdb/mnist/.

Christianto C. Liu, Ilya Ganusov, Martin Burtscher, and Sandip Tiwari. 2005. Bridging the processor-memory performance gap with 3D IC technology. *IEEE Design & Test of Computers* 22, 6 (2005), 556–564.

James MacQueen and others. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. Berkeley, CA, 14.

Ethan Mirsky and Andre DeHon. 1996. MATRIX: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *Proceedings of the 1996 IEEE Symposium on FPGAs for Custom Computing Machines*. IEEE, 157–166.

Kazuaki Murakami, Satoru Shirakawa, and Hiroshi Miyajima. 1997. Parallel processing RAM chip with 256 Mb DRAM and quad processors. In *Digest of Technical Papers of the 43rd IEEE International Solid-State Circuits Conference (ISSCC'97)*. IEEE, 228–229.

K. Neshatpour, M. Malik, M. A. Ghodrat, and H. Homayoun. 2015. Accelerating big data analytics using FPGAs. In *Proceedings of the IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'15)*. 164–164. DOI: http://dx.doi.org/10.1109/FCCM.2015.59

John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. 2008. Scalable parallel programming with CUDA. *Queue* 6, 2 (2008), 40–53.

Nios II Processor: The World's Most Versatile Embedded Processor. Retrieved from http://www.altera.com/devices/processor/nios2/ni2-index.html.

NVIDIA. 2016. *CUDA Toolkit Documentation* (7.5 ed.). NVIDIA. Retrieved from https://docs.nvidia.com/cuda/.

Markos Papadonikolakis and C. Bouganis. 2012. Novel cascade FPGA accelerator for support vector machines classification. *IEEE Transactions on Neural Networks and Learning Systems* 23, 7 (2012), 1040–1052.

David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A case for intelligent RAM. *IEEE Micro* 17, 2 (1997), 34–44.

S. Paul, S. Chatterjee, S. Mukhopadhyay, and S. Bhunia. 2009. Nanoscale reconfigurable computing using non-volatile 2-D STTRAM array. In *Proceedings of the 9th IEEE Conference on Nanotechnology (IEEE-NANO 2009)*. 880–883.

Somnath Paul, Robert Karam, Swarup Bhunia, and Ruchir Puri. 2014a. Energy-efficient hardware acceleration through computing in the memory. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 266.

Somnath Paul, Aswin Krishna, Wenchao Qian, Robert Karam, and Swarup Bhunia. 2014b. MAHA: An energy-efficient malleable hardware accelerator for data-intensive applications. *IEEE Transactions on Very Large Scale Integration Systems*. IEEE, 1005–1016.

Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jordan Gray, et al. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. In *Proceedings of the ACM/IEEE 41st International Symposium on Computer Architecture (ISCA'14)*. IEEE, 13–24.

QUADRO FOR DESKTOP WORKSTATIONS. Retrieved from http://www.nvidia.com/object/quadro-desktop-gpus.html.

Timothy G. Rogers, Mike O'Connor, and Tor M. Aamodt. 2013. Divergence-aware warp scheduling. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 99–110.

Yi Shan, Bo Wang, Jing Yan, Yu Wang, Ningyi Xu, and Huazhong Yang. 2010. FPMR: MapReduce framework on FPGA. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 93–102.

Hertej Singh, Ming-Hau Lee, Guangming Lu, Fadi J. Kurdahi, Nader Bagherzadeh, and M. Chaves Eliseu Filho. 2000. MorphoSys: An integrated reconfigurable system for data-parallel and computation-intensive applications. *IEEE Transactions on Computers* 49, 5 (2000), 465–481.

Kilian Stoffel and Abdelkader Belkoniene. 1999. Parallel k/h-means clustering for large data sets. *European Conference on Parallel Processing*. Springer, 1451–1454.

Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Balakrishna Iyer, Bernard Brezzo, Donna Dillenberger, and Sameh Asaad. 2012. Database analytics acceleration using FPGAs. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*. ACM, 411–420.

Helen Sun and Peter Heller. 2012. Oracle information architecture: An architect's guide to big data. *Oracle, Redwood Shores* (2012).

Yangzihao Wang, Andrew Davidson, Yuechao Pan, Yuduo Wu, Andy Riffel, and John D. Owens. 2015. Gunrock: A high-performance graph processing library on the GPU. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, 265–266.

Yu Wang, Boxun Li, Rong Luo, Yiran Chen, Ningyi Xu, and Huazhong Yang. 2014. Energy efficient neural networks for big data analytics. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'14)*. IEEE, 1–2.

Louis Woods and Gustavo Alonso. 2011. Fast data analytics with FPGAs. In *Proceedings of the IEEE 27th International Conference on Data Engineering Workshops (ICDEW'11)*. IEEE, 296–299.

Ren Wu, Bin Zhang, and Meichun Hsu. 2009. Clustering billions of data points using GPUs. In *Proceedings of the Combined Workshops on UnConventional High Performance Computing Workshop Plus Memory Access Workshop*. ACM, 1–6.

Weizhong Zhao, Huifang Ma, and Qing He. 2009. Parallel k-means clustering based on mapreduce. *IEEE International Conference on Cloud Computing*. Springer, 674–679.

Jacob Ziv and Abraham Lempel. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* 23, 3 (1977), 337–343.

Jacob Ziv and Abraham Lempel. 1978. Compression of individual sequences via variable-rate coding. *IEEE Transaction on Information Theory* 24, 5 (1978), 530–536.

Received March 2016; revised July 2016; accepted September 2016